Merton's portfolio problem, constant fraction investment strategy and frequency of portfolio rebalancing

by

JOACHIM HOLTH

THESIS

for the degree of

MASTER OF SCIENCE

(Modelling and data analysis)

Faculty of Mathematics and Natural Sciences
University of Oslo

November 2011

Det matematisk- naturvitenskapelige fakultet
Universitetet i Oslo

Acknowledgements

I would like to thank my supervisor professor Fred Espen Benth at the department of mathematics at the University of Oslo, for giving me an interesting task to work with. I would also like to thank the students of study room B 802 in Abel's tower for their good company. Finally I would like to thank my friends and family for supporting me in stressful times. A special thanks to my sister Ida in that regard.

Contents

Chapter 1

Introduction

Banks, investment funds and insurance companies are examples of investors that invest money in the financial markets. Naturally, they want to make as much money as possible on their investments, but any serious investor also need to consider the risk involved. Normally, an investor is to a certain degree risk averse, that is, the investor is reluctant to invest in an asset with a potentially high upside if it means that the risk of loosing money is high as well. For example, because of their obligations towards their customers, a traditional bank or an insurance company, which invest funds on behalf of their customers in the financial market, cannot allow themselves to take too much risk. The aim of such investors is to maximize the expected returns on their investments while at same time limiting the risk involved. One way of modelling such behaviour is through the theory of stochastic control and the maximization of expected utility.

Potential objects of investment can basically be divided into two categories: risky assets, which are assets with an uncertain future return, and risk-free assets, which are assets with a beforehand known future return. Examples of risky assets are stocks, derivatives, real estate, raw materials et cetera. Examples of risk-free assets are bonds and t-bills. Depending on the degree of risk aversion, an investor may compose an investment portfolio as a mix of both risky and risk-free assets to match the level of risk the investor is comfortable with. For such a risk averse investor it is natural to ask: which allocation strategy or investment strategy will maximize the expected utility of the portfolio? This is the question that Nobel laureate in economics Robert C. Merton addressed and mathematically solved in a paper [15] in 1969 by using stochastic control. The problem is popularly known as "Merton's portfolio problem", which has become a well-studied problem in articles and literature.

The most basic version of the problem gives an investor the limited choice of investing her wealth in a risky asset and a risk-free asset. Given some additional assumptions, Merton found that the optimal allocation strategy or trading strategy is to keep a constant fraction of the wealth in the risky asset (and hence, a constant fraction in the risk-free asset). This can be generalized to a situation with several risky assets and one risk-free asset and the conclusion is basically the same, that is to keep a constant fraction of the wealth in the risky assets. This strategy is indeed a frequently used strategy among investors. For example, the norwegian pension fund, with an approximate value of NOK 3,000 billion, uses this strategy to control risk.

From a realistic point of view, the conclusion of "Merton's portfolio problem" is based on rather stylized mathematics as well as stylized assumptions. For example, one such assumption is that the dynamics of the risky assets are assumed to follow geometric Brownian motions, implying normally distributed log returns. With real stock prices, this is usually not the case. Analysis of the distributions of real stock returns shows that the distributions have heavier or fatter "tails", which means there is a higher chance of large price changes than one would expect with the normal distribution [7].

Another problem is that the conclusion is based on a continuous mathematical framework. It is also a fact that in today's extremely liquid financial markets, stocks and other risky assets change value almost continuously in time. This means that to follow the optimal strategy an investor has to rebalance her portfolio at the same rate as the prices changes. This is obviously not very realistic seen from a practical point of view. Also, transaction costs would make such a behaviour extremely expensive.

In this thesis we will address this problem by discretization. Wikipedia defines discretization as the process of transferring continuous models and equations into discrete counterparts [5]. The discretization of the model allows for simulation. Through the simulations we want to simulate the portfolio of an investor making investment decisions according to the optimal investment strategy of constant fractions. The investor will only be allowed to rebalance her portfolio at certain discrete time points. These discrete time points will be chosen in such a way as to reflect different types of rebalancing strategies, such as daily rebalancings or monthly rebalancings.

The design of simulation models as well as the discussion of the resulting simulation runs of these models is the main focus of this thesis. Through the simulations we want to investigate how the optimal strategy performs in a more realistic setting. To compare the impact of discretization with the original continuous model, we will among other things measure the difference in utility or the loss of utility. The loss of utility will also be related to different rebalancing strategies. Regarding the different rebalancing strategies we will also calculate the Sharpe ratio for each strategy. The Sharpe ratio relates portfolio return with portfolio risk.

Basically, we will consider three different simulation models. The first model, which will serve as a basis for the other models, is a simple and rather unrealistic model, where the main purpose is to look at the impact of discretization itself. In the second model we will increase the complexity and hopefully the realism of the model by adding transaction costs. Finally, in the third simulation model, we will assume stochastic volatility. So the basic idea is to start out with a relatively simple simulation model and then gradually add more complexity, and with that, more realism.

Chapter 2

Background theory

2.1 Stock price model

We will in this thesis consider two stock price models for the modelling of risky asset prices. The basic structure of the models are similar. The difference between them lies in the assumptions about volatility. In the first model we will make the rather naive assumption of constant volatility. In the second model we will make the more realistic assumption of stochastic volatility.

2.1.1 Constant volatility

A frequently used model for modelling risky asset prices is the geometric Brownian motion. If S_t denotes the price of a risky asset at time t, then S_t will follow a geometric Brownian motion if it satisfies the following stochastic differential equation (abbreviated SDE),

$$dS_t = \mu S_t dt + \sigma S_t dB_t, \tag{2.1}$$

where μ is the drift and σ is the volatility of the risky asset, which we assume is constant. B_t is the stochastic process known as Brownian motion. Benth [1] defines Brownian motion as follows,

Definition 2.1.1 Brownian motion B_t is a stochastic process starting at zero, i.e. $B_0 = 0$, and which satisfies the following three properties:

1. Independent increments: The random variable $B_t - B_s$ is independent of the random variable $B_u - B_v$ whenever $t > s \geq u > v \geq 0$.

2. Stationary increments: The distribution of $B_t - B_s$ for $t > s \geq 0$ is only a function of $t - s$, and not of t and s separately.
3. Normal increments: The distribution of $B_t - B_s$ for $t > s \geq 0$ is normal with expectation 0 and variance $t - s$.

The probability density function of a normally distributed variable X is

$$f_X(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right).$$

Using Ito's formula the explicit solution of the SDE of the geometric Brownian motion can be shown to be

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma B_t\right). \tag{2.2}$$

2.1.2 Stochastic volatility

Assume instead that the volatility is non-constant and stochastic. A popular model for modelling stochastic volatility is the Heston model, proposed in 1993 by the American mathematician Steven Heston [9]. The Heston model can be stated as follows,

$$dS_t = \mu S_t dt + \sqrt{\nu_t} S_t dB_t^S, \tag{2.3}$$
$$d\nu_t = \kappa(\theta - \nu_t)dt + \xi\sqrt{\nu_t} dB_t^\nu \tag{2.4}$$
$$dB_t^S dB_t^\nu = \rho dt. \tag{2.5}$$

The SDE (2.4) is also known as the SDE of a CIR-process [3]. The CIR-process is mean-reverting, which means that in the long run, the process tends to drift towards its long-term mean θ. The intensity of this mean-reverting tendency is scaled by the parameter κ. Similarly to the stochastic stock price dynamics of the constant volatility model, the stochastic behaviour of the stock price of the Heston model is driven by a Brownian motion B_t^S. Additionally, we have that the volatility process [1] ν_t is driven by a Brownian motion B_t^ν. The Brownian motion is scaled by the parameter ξ, which often is referred to as the volatility of the volatility. The last expression (2.5) tells us that these Brownian motions are assumed to be correlated with correlation coefficient ρ. This means that the

[1] Note that the process ν_t is a variance process, not a volatility process per se. The volatility process itself is of course given as $\sqrt{\nu_t}$, but given the context, we will refer to (2.4) as an SDE modelling stochastic volatility.

joint distribution of the Brownian motions is described by a bivariate normal distribution with mean vector $\boldsymbol{\mu}$ and convariance matrix $\boldsymbol{\Sigma}$ given as

$$\boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} dt.$$

2.2 The Sharpe ratio

The Sharpe ratio, which was introduced by Nobel laureate William F. Sharpe in 1966, is a measure of portfolio performance and as such a measure of the performance of an investor or portfolio manager. The original name of the Sharpe ratio is the reward-to-variability ratio and it measures the excess return per unit of risk of a portfolio [18]. According to Sharpe [17], there are two versions of the Sharpe ratio. We have the ex ante version, which is calculated through expected values by assuming that the future returns on the portfolio are distributed according to some known statistical distribution, and hence, is prospective, and we have the ex post version where the calculation of the ratio is based on historical portfolio returns, and hence, is retrospective. The following definition of the ex ante Sharpe ratio is based on the definition in Wikipedia [18], but with slightly altered notation to better fit into the notational scheme of this thesis.

Definition 2.2.1 If X_t is the return on an investment portfolio and X_t^f is the return on a benchmark asset at time t, then the ex ante Sharpe ratio at time t can be defined as

$$SR_t^{\text{ea}} = \frac{E[X_t - X_t^f]}{\sqrt{\text{Var}[X_t - X_t^f]}}. \tag{2.6}$$

We observe that the nominator of the ratio is a measure of the excess return on the portfolio, whereas the denominator is a measure of the risk of the portfolio. A positive excess return means that the we expect our investment portfolio to perform better than the benchmark asset and vice versa. As such, the ex ante Sharpe ratio may serve as a guide as to where we should invest our money. We also observe that an increase in the risk of the portfolio is associated with a decrease in ex ante Sharpe ratio. This is based on the common assumption that a high-risk investment should yield high profits compared to a low-risk investment. Note that if $x_t^f = X_t^f$ is a deterministic quantity or a constant it follows that the ex ante Sharpe ratio can be formulated as

$$SR_t^{\text{ea}} = \frac{E[X_t] - x_t^f}{\sqrt{\text{Var}[X_t]}}.$$

Sharpe [17] gives the following definition of the ex post Sharpe ratio (with slightly altered notation):

Definition 2.2.2 Given a time series of historical returns on a portfolio $\{x_t\}_{t=1,\ldots,T}$ and a time series of historical returns on a benchmark portfolio or asset $\{x_t^f\}_{t=1,\ldots,T}$, the ex post Sharpe ratio is defined as

$$SR_T = \frac{\bar{x} - \bar{x}^f}{\hat{\sigma}_x},$$

where $\bar{x} = \sum_{t=1}^{T} x_t$ is the sample mean of the portfolio returns, $\bar{x}^f = \sum_{t=1}^{T} x_t^f$ is the sample mean of the returns of the benchmark portfolio or asset and $\hat{\sigma}_x = (T-1)^{-1/2}(\sum_{t-1}^{T}(x_t - \bar{x})^2)^{1/2}$ is the sample standard deviation of the portfolio returns.

2.3 The Euler-Maruyama method

The following presentation of the Euler-Maruyama method is based on the presentation of Kloeden and Platen [12]. Consider an Ito process

$$dX_t = a\,(t, X_t)\,dt + b\,(t, X_t)\,dB_t,$$

defined on a time interval $[0, T]$ with initial value x_0. B_t is Brownian motion at time t. An approximate solution to this Ito process can be found through a so-called Euler approximation, also known as an Euler-Maruyama approximation. The approximation method requires the time interval to be divided into smaller subintervals, that is we need to construct a time discretization of the time interval:

$$0 = t_0 < t_1 < \cdots < t_n = T.$$

According to Kloeden and Platen, the Euler approximation is a continuous time stochastic process $\{Y_t\}_{t\in[0,T]}$. However, the process is only calculated at the discrete time points given by the time discretization. The Euler approximation of X_{k+1} ($X_k = X_{t_k}$) is defined recursively as

$$Y_{k+1} = Y_k + a\,(Y_k)\,\Delta t_k + b(Y_k)\Delta B_k,$$

with $Y_0 = x_0$ and where $\Delta t_k = t_{k+1} - t_k$ and $\Delta B_k = B_{k+1} - B_k$. We see that the Euler approximation describes a simple, iterative approximation scheme.

Chapter 3

Merton's portfolio problem

3.1 Introduction

Consider a scenario where an investor has the limited choice of investing his wealth in only two different assets: a risky asset (for example a stock) and a risk-free asset (for example a bank account). Given a limited time horizon, the goal of the investor, who is avert to risk, is to maximize the expected utility of his wealth at the end of the time horizon. How should the investor allocate and reallocate his wealth at each time point to achieve this goal? Stated a bit differently, what is the optimal investment strategy at each time point that will maximize the expected utility of the wealth at some terminal time?

3.2 Solution to the problem

Let the price of the risky asset at time t be denoted by S_t. The dynamics of the risky asset price is given by (2.1), which is the stochastic differential equation also known as geometric Brownian motion. The parameters μ and σ represent respectively the drift and the volatility of the risky asset. B_t is the stochastic process known as Brownian motion. The price of the risk-free asset at time t is denoted by R_t and satisfies the following deterministic differential equation:

$$dR_t = rR_t dt. \tag{3.1}$$

The parameter r represents the risk-free continuously compounding interest rate. It is natural to assume that $E[S_t] > E[R_t]$ which means that we assume $\mu > r$.

Let the wealth of the investor at time t be denoted by V_t. At each time point t the investor must invest a fraction u_t of his wealth in the risky asset. The remaining wealth $1 - u_t$ is invested in the risk-free asset. This means that the value of the risky investment at time t is $u_t V_t$ and that the value of the risk-free investment is $(1 - u_t)V_t$. The stochastic differential equation of the wealth or portfolio value is then simply

$$\begin{aligned} dV_t &= du_t V_t + d(1-u_t)V_t = \mu u_t V_t dt + \sigma u_t V_t dB_t + r(1-u_t)V_t dt \\ &= (\mu u_t + r(1-u_t))V_t dt + \sigma u_t V_t dB_t. \end{aligned} \tag{3.2}$$

The object now is to find the optimal allocation strategy u_t at each time point t, which gives the best possible outcome at some future terminal time T for the investor. Assume that no borrowing or short selling is allowed, which means that we require that $0 \leq u_t \leq 1$. As already stated, the investor is risk averse. One way of modelling risk aversion is through expected utility theory. Introduce an increasing and concave utility function $U(x)$. Instead of maximizing the expected portfolio value itself, the investor wants to maximize the expected utility of the wealth at terminal time T. Assume a time horizon restricted by an initial time t_0 and a terminal time T, i.e. $t_0 < t < T$, and assume an initial portfolio value V_{t_0}. The maximization problem can be stated as

$$I(t,x) = \max_{u_t} \mathrm{E}[U(V_T) | t_0 = t, V_{t_0} = x].$$

This constitutes an optimal control problem[1] , where the allocation strategy u_t is the actual control function. Define

$$\begin{aligned} \phi(t,x) &= \frac{\partial I(t,x)}{\partial t} + (\mu u_t + r(1-u_t))\frac{\partial I(t,x)}{\partial x} + \frac{1}{2}\sigma^2 u_t^2 x^2 \frac{\partial^2 I(t,x)}{\partial x^2} \\ &= \frac{\partial I(t,x)}{\partial t} + (r + (\mu - r)u_t)\frac{\partial I(t,x)}{\partial x} + \frac{1}{2}\sigma^2 u_t^2 x^2 \frac{\partial^2 I(t,x)}{\partial x^2}. \end{aligned} \tag{3.3}$$

The optimal solution must satisfy [15]

$$\max_{u_t} [\phi(t,x)] = 0, \quad t \in [t_0, T] \tag{3.4}$$

and $I(T, V_T) = U(V_T)$. (3.4) is a continuous-time version of the Bellman-Dreyfus fundamental equation of optimality. This requirement also gives the optimal solution to the problem. To find a solution that is compatible with the utility function $U(x)$ (increasing and concave), we require that $I_x = \partial I(t,x)/\partial x > 0$ and $I_{xx} = \partial^2 I(t,x)/\partial x^2 < 0$. Also, a first-order condition for finding a maximum is [15]

$$(\mu - r)I_x + \sigma^2 u_t x I_{xx} = 0,$$

[1] In this slightly simplified version of the problem, we do not consider the possibility that the portfolio value could reach zero.

which is equivalent to

$$u_t = -\frac{(\mu - r)I_x}{\sigma^2 x I_{xx}}. \tag{3.5}$$

Substituting this expression into (3.3) yields

$$\begin{cases} \begin{cases} I_t + x\left(r + (\mu - r)\left(-\frac{(\mu - r)I_x}{\sigma^2 x I_{xx}}\right)\right) I_x \\ +\frac{1}{2}\sigma^2\left(-\frac{(\mu - r)I_x}{\sigma^2 x I_{xx}}\right)^2 x^2 I_{xx} = 0 \end{cases}, & t < T \\ I(t,x) = U(x), & t = T \end{cases}$$

$$\Leftrightarrow \begin{cases} I_t + rxI_x - \frac{(\mu - r)^2 I_x^2}{\sigma^2 I_{xx}} + \frac{1}{2}\frac{(\mu - r)^2 I_x^2}{\sigma^2 I_{xx}} = 0, & t < T \\ I(t,x) = U(x), & t = T \end{cases}$$

$$\Leftrightarrow \begin{cases} I_t + rxI_x - \frac{(\mu - r)^2 I_x^2}{2\sigma^2 I_{xx}} = 0, & t < T \\ I(t,x) = U(x), & t = T \end{cases} \tag{3.6}$$

with $I_t = \partial I(t,x)/\partial t$.

3.3 Power utility

In this thesis we will model the utility of wealth x by the power function

$$U(x) = x^\gamma, \quad 0 < \gamma < 1. \tag{3.7}$$

This choice of utility function is compatible with the assumptions of the previous section, that is increasing and concave utility. This choice also allow us to find a closed form solution of the optimal control function. We will refer to γ as the risk aversion parameter. We see that a low value of the risk aversion parameter is associated with high aversion to risk and vice versa. To find a solution, we need to guess a solution, so we try

$$I(t,x) = f(t)x^\gamma. \tag{3.8}$$

Substituting this expression into (3.6) yields

$$\begin{cases} f'(t)x^\gamma + rxf(t)\gamma x^{\gamma-1} - \frac{(\mu - r)^2 f^2(t)\gamma^2 x^{2(\gamma-1)}}{2\sigma^2 f(t)\gamma(\gamma-1)x^{\gamma-2}} = 0, & t < T \\ f(t)x^\gamma = x^\gamma, & t = T \end{cases}$$

$$\Leftrightarrow \begin{cases} -\frac{f'(t)}{f(t)} = r\gamma + \frac{(\mu - r)^2\gamma}{2\sigma^2(1-\gamma)}, & t < T \\ f(t) = 1, & t = T. \end{cases}$$

Solving these equations with respect to $f(t)$ yields

$$f(t) = \exp\left(\left(r\gamma + \frac{(\mu - r)^2\gamma}{2\sigma^2(1-\gamma)}\right)(T-t)\right).$$

Substituting this solution into (3.8) gives

$$I(t,x) = \exp\left(\left(r\gamma + \frac{(\mu - r)^2\gamma}{2\sigma^2(1-\gamma)}\right)(T-t)\right)x^\gamma. \tag{3.9}$$

Finally, we find the optimal control u_t^* by solving (3.5) with respect to (3.9),

$$u_t^* = -\frac{(\mu - r)\exp\left(\left(r\gamma + \frac{(\mu-r)^2\gamma}{2\sigma^2(1-\gamma)}\right)(T-t)\right)\gamma x^{\gamma-1}}{\sigma^2 x \exp\left(\left(r\gamma + \frac{(\mu-r)^2\gamma}{2\sigma^2(1-\gamma)}\right)(T-t)\right)\gamma(\gamma-1)x^{\gamma-2}} = \frac{\mu - r}{\sigma^2(1-\gamma)}, \tag{3.10}$$

which is in fact a constant independent of time. We can conclude that the optimal allocation strategy is to hold a constant fraction u^* of the wealth in the risky asset, and hence, a constant fraction $1 - u^*$ in the risk-free asset.

The ratio (3.10) is also known as the Merton ratio. The numerator of the ratio is the difference between the risky asset drift and the risk-free rate of return. Under the assumption that no short selling is allowed, it is clear that if $\mu - r \leq 0$ an investor will invest all of her money in the risk-free asset. For a rational and risk-averse investor, this is the obvious allocation strategy since it means the highest expected return combined with no risk at all. If $\mu - r > 0$ the picture becomes more complex. A positive difference implies that the investor will invest at least a fraction of her wealth in the risky asset. This fraction is in part determined by the size of the difference between the risky asset drift and the risk-free rate of return, but it is also scaled by the parameter values of the denominator. The denominator is the product between the square of the volatility of the risky asset and one minus the risk aversion. Keeping all other parameters of the Merton ratio constant, we see that an increase in volatility leads to a decrease of the Merton ratio itself, and vice versa. This property of the Merton ratio is quite logical considering the fact that a risk-averse investor would be more reluctant to invest in the risky asset if the volatility increases. One minus the risk aversion can be interpreted as a scaling parameter that scales the impact of the volatility on the Merton ratio. We see that a low value of the risk aversion parameter γ, in relative terms, scales the impact of the volatility up, and vice versa. This is also a quite logical property since a low risk aversion parameter value is associated with high risk aversion.

Chapter 4

Estimation of parameters

4.1 Estimation of the risky asset and riskfree asset parameters

The SDE describing the dynamics of the risky asset has two parameters or constants, the drift μ and the volatility σ. The differential equation describing the risk-free asset has only one parameter, the continuously compounding interest rate r. To estimate the risky asset parameters, we will use a time series consisting of daily closing index prices of the norwegian stock market index OBX to act as a proxy for stock investments. The plot of figure 4.1 shows the development

Figure 4.1: OBX index price, 3rd January 1996 - 9th March 2009.

of the OBX index price. The Lehman Brothers bankruptcy of 15th September 2008, which many count as the start of the financial crisis, is indicated by the dotted vertical line.

Due to the fact that the wealth process (5.2) describing the solution of the SDE (3.2) is a lognormal process it is natural to consider the log returns of the price data [1] when we want to estimate μ and σ. Given a time series of n daily prices $\{s_k\}_{k=1,\ldots,n}$, the log return of the time interval $[t_k, t_k + 1)$ is defined as

$$x_k = \log\left(\frac{s_{k+1}}{s_k}\right), \quad k = 1, \ldots, n-1,$$

where log is interpreted as the natural logarithm. Using the estimation method of maximum likelihood, we can, according to Benth [1], estimate the drift μ and the volatility σ by using

$$\hat{\mu} = \frac{1}{N\Delta t}\sum_{k=1}^{N-1} x_k \tag{4.1}$$

$$\hat{\sigma} = \sqrt{\frac{1}{(N-1)\Delta t}\sum_{k=1}^{N-1}(x_k - \hat{\mu})^2}. \tag{4.2}$$

This means that the risk of the risky asset is measured as the variability of the OBX log returns. Using the convention of 252 trading days in one year, to estimate the annual drift and volatility we must choose $\Delta t = 1/252$ since the log returns are sampled on a daily basis.

To estimate the continuously compounding interest rate we will use historical data of the effective annual interest rate of norwegian twelve month treasury bills. More specifically, the treasury bill time series consists of daily recordings of the syntetic annual interest rate. For easier comparison with the OBX log returns, given a time series of M annual treasury bill interest rates $\{b_k\}_{k=1,\ldots,M}$ and $\Delta t = 1/252$, the daily log returns can be calculated by the transformation

$$y_k = \Delta t \log(1 + b_k), \quad k = 1, \ldots, M.$$

Analogously to the estimation of the risky asset drift, the continuously compounding interest rate r can then be estimated by using

$$\hat{r} = \frac{1}{M\Delta t}\sum_{k=1}^{M} y_k.$$

Initially, the OBX log return and treasury bill time series intended used for parameter estimation were time series covering the period from the start of 1996 until the end of 2010. However, by including OBX and treasury bill log return data for 2010 and most of 2009 the estimated difference between the risky asset drift and the continuously compounding interest rate becomes so large that (3.10) tells me to invest all of the wealth into the risky asset, i.e. $u^* = 1$, even for $\gamma > 1$. For the sake of an interesting simulation scenario and discussion, $u^* = 1$ is not desirable. It turns out that estimates based on 3308 OBX log returns and 3117

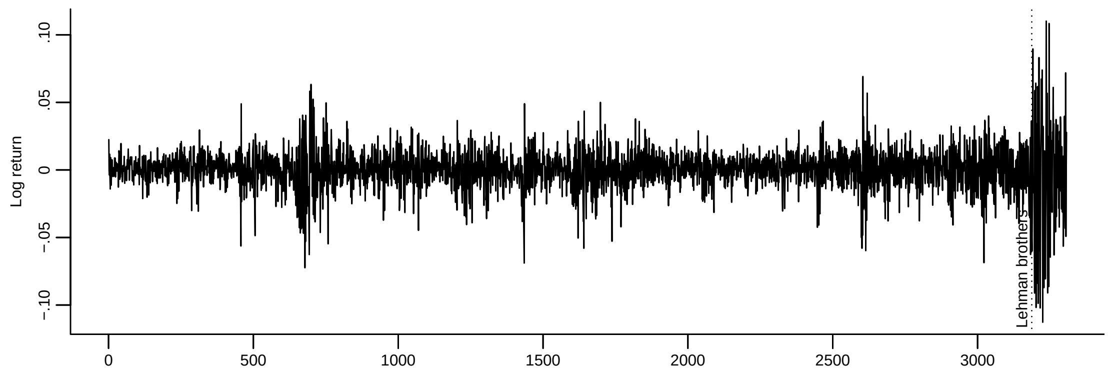


Figure 4.2: OBX log returns, 3rd January 1996 - 9th March 2009.

treasury bill interest rates in the time period from 3rd January 1996 until 9th March 2009 do not give undesirable estimates. The estimates are summarized in table 4.1. The plot of figure 4.2 shows the development of the log returns of the OBX index. The size of the variations of the log returns reflects the amount of uncertainty in a market. We see how the uncertain economic times of the financial crisis has an impact on the variations of the log returns of the OBX index.

4.2 Estimation of risk aversion through VaR

The utility function (3.7) measures the investor's relative satisfaction with a given wealth x. The parameter γ is still to be interpreted as a risk aversion parameter. The utility function is usually assumed to be increasing and concave [14], which implicates that $0 < \gamma < 1$. This means that the investor becomes relatively less satisfied with increasingly bigger wealth, i.e. the investor is risk averse. For example, a low risk aversion parameter value would indicate a high aversion to risk.

To estimate the risk aversion parameter we will in this thesis employ the method of value at risk, abbreviated VaR. VaR gives us a simple way to measure the risk of losing money [8]. Jorion [11] gives the following definition: Value at risk is the worst loss over a target horizon such that there is a low, prespecified probability that the actual loss will be larger. In mathematical terms, by combining the definitions of Jorion and Benth, value at risk can be defined as follows:

Definition 4.2.1 Define L as the loss, measured as a positive number, and $\text{VaR}_{1-\alpha}$ as the value at risk at confidence level $1-\alpha$. Then, value at risk is defined as the loss, in absolute value, such that

$$P(L > \mathrm{VaR}_{1-\alpha}) = \alpha.$$

There are different ways to measure the loss of the portfolio, for instance by looking at the actual portfolio value itself. But to achieve simplicity in the calculations we will choose the portfolio's log returns as our measure of loss. The log returns are defined as

$$X_k = \log\left(\frac{V_{k+1}}{V_k}\right). \tag{4.3}$$

Let $x^*_{1-\alpha}$ denote the value at risk at confidence level $1-\alpha$, then by definition

$$P(-X_k > x^*_{1-\alpha}) = \alpha. \tag{4.4}$$

If the dynamics of the wealth follows the SDE (3.2), it can be shown that the log returns are normally distributed with expectation $(\mu u^* + r(1-u^*) - .5\sigma^2 u^{*2})\delta$ and standard deviation $\sigma u^*\sqrt{\delta}$. With the probability distribution of the log returns known it is possible to solve (4.4) with respect to γ. The solution, which involves a quadratic equation, is

$$\gamma = 1 + \frac{(\mu - r)\left(\mu - r + \frac{q_\alpha \sigma}{\sqrt{\delta}} \pm \sqrt{\left(\mu - r + \frac{q_\alpha \sigma}{\sqrt{\delta}}\right)^2 + 2\sigma^2\left(\frac{x^*_{1-\alpha}}{\delta} + r\right)}\right)}{2\sigma^2\left(\frac{x^*_{1-\alpha}}{\delta} + r\right)}. \tag{4.5}$$

With values given for μ, σ, r, δ and $x^*_{1-\alpha}$ and with q_α defined as the α-quantile of the standard normal distribution, (4.5) gives us a way to estimate γ.

To be able to estimate γ we will also need to estimate the VaR. There are several different methods for estimating the VaR, but here we will use historical data as my method of estimation. Specifically, the historical data used for estimation of the VaR are the same historical log returns as were used for the estimation of the risky asset drift and volatility and the historical treasury bill rents as were used for the estimation of the risk-free rent. Given a confidence level $1-\alpha$, an estimate for the VaR is simply the α-quantile of the historical data. To take into account that the portfolio consists of investments both in a risky and a risk-free asset we will estimate the VaR by a weighted sum of the OBX and the treasury bill α-quantiles. Choosing a conventional confidence level of .99, a time horizon of one day and multiplying the OBX and the treasury bill log return α-quantiles with equal weights, that is weights equal to .5, we estimate that $x^*_{.99} = .0252$. The insertion of this estimate along with the other parameter estimates into (4.5) yields two solutions. Naturally, we choose to keep the solution, $\hat{\gamma} = .5255$, which is compatible with the assumption of an increasing and concave utility function. The complete set of parameter estimates required for the calculation of the optimal investment strategy u^* is summarized in table 4.1.

Parameter	Estimate
μ	.0657
σ	.2537
r	.0449
γ	.5255

Table 4.1: The parameter estimates.

4.3 Calibration of the Heston model

4.3.1 Introduction

In this section we will estimate the parameters of the Heston stochastic volatility model, or in other words, calibrate the model. The parameters that need to be estimated are given in the set $\Omega^H = \{\nu_0, \kappa, \theta, \xi, \rho\}$. The calibration of the Heston model is not as straightforward as the calibration of the risky asset model (2.1). In fact, the calibration of stochastic volatility models can, according to some, be notoriously difficult. There are many different methods of calibration available, each with its own advantages and disadvantages. The different methods can be divided into two categories based on the underlying set of data used for the calibration. According to Javaheri [10] there are two possible sets of data that we can use for calibration: option prices or historical stock prices.

Using option prices, the goal is to find the set of parameter estimates that most accurately reproduces the volatilities that are implied by the real market prices of vanilla options. As such, the calibration problem that this approach entails, constitutes an inverse problem. According to Moodley [16] the most popular way of solving this inverse problem is to minimise the squared differences between the option prices implied by the model and the market prices over the parameter space. This method is also known as least squares estimation. For example, given a set of n call option market prices $\{C_j(K_j, T_j)\}_{j=1,\ldots,n}$ with strike K_j and maturity T_j and n model estimated call option prices $\{\hat{C}_j(K_j, T_j)\}_{j=1,\ldots,n}$ with stochastic volatility based on the Heston model, the least squares scheme could be formulated as

$$\min_{\Omega^H} \sum_{j=1}^{n} \left(\hat{C}_j(K_j, T_j) - C_j(K_j, T_j) \right)^2 .$$

Alternatively, in conjunction with model calibration based on stock prices, there exists different estimation methods based on maximum likelihood. The basic idea with maximum likelihood estimation is to maximize the likelihood function (which is defined as a conditional joint probability function) over the model parameter set. Stated a bit differently, the goal is to find the most likely model parameter set given the stock price data.

What are the advantages and disadvantages of the two different approaches? According to Javaheri [10], the advantage of using calibration methods based on option prices is that it guarantees that the modelled option prices will match the option market prices within a certain tolerance. The disadvantage is the limited availability of option price data. With stock prices, the situation is opposite: we have no guarantee that the estimated option prices based on the model will match option market prices, but the availability of stock price data is usually plentiful. We will however not use any of these methods in this thesis.

4.3.2 Estimation of ν_0, θ and κ through linear regression

For the calibration of the Heston model we will apply a simpler and more hands-on approach. As stated in subsection 2.1.2, the volatility process ν_t is a CIR-process. The CIR-process is a popular model for modelling stochastic short term interest rates. To calibrate the CIR model, Wikipedia suggests discretizing the SDE and then to fit the discretized model to a set of short term interest rate data by using linear regression. To calibrate the Heston model, we will use a similar approach. The Euler approximation of the SDE of the volatility process of the Heston model can be expressed as

$$\nu_{k+1} = \nu_k + \kappa(\theta - \nu_k)\Delta t_k + \xi\sqrt{\nu_k}\Delta B_k^\nu. \tag{4.6}$$

This is equivalent to

$$\frac{\nu_{k+1} - \nu_k}{\sqrt{\nu_k}} = \kappa\theta\Delta t_k \frac{1}{\sqrt{\nu_k}} - \kappa\Delta t_k\sqrt{\nu_k} + \xi\epsilon_k^\nu, \tag{4.7}$$

where $\epsilon_k \sim N(0, \Delta t_k)$. We recognize this expression as a linear model suitable for linear regression.

Assume equidistant time increments, that is $\Delta t_k = \delta$. The linear model (4.7) can be reformulated as

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \epsilon_i,$$

with

$$y_i = \frac{\nu_{i+1} - \nu_i}{\sqrt{\nu_i}}, \quad \beta_1 = \kappa\theta\delta, \quad x_{i1} = \frac{1}{\sqrt{\nu_i}}, \quad \beta_2 = -\kappa\delta, \quad x_{i2} = \sqrt{\nu_i}, \quad \epsilon_i = \xi\epsilon_i^\nu.$$

We can now apply the ordinary least squares estimators to find estimates for the β's. From the above equations it is clear that

$$\hat{\theta} = -\frac{\hat{\beta}_1}{\hat{\beta}_2}, \quad \hat{\kappa} = -\frac{\hat{\beta}_2}{\delta}. \tag{4.8}$$

There is however a problem with this approach: we will require a data set of historical short term variances. Initially we do not have such a set of data, but given a set of historical log returns, we can construct a set of short term variances by calculating the variances over short subsections of the log return data. The basic idea is to let a narrow "window" move discretely from the beginning to the end of the log return data and to construct a variance data point each time the window moves up one notch. Given a time series of n log return data $\{x_k\}_{k=1,\ldots,n}$ and assuming a moving window of length l, a time series of short term variances can be constructed in the following fashion:

$$\begin{aligned}
\nu_1 &= \frac{1}{(l-1)\Delta t}\sum_{j=1}^{l}\left(x_j - \bar{x}_1\right)^2, \quad \bar{x}_1 = \frac{x_1 + \cdots + x_l}{l} \\
\nu_2 &= \frac{1}{(l-1)\Delta t}\sum_{j=2}^{l+1}\left(x_j - \bar{x}_2\right)^2, \quad \bar{x}_2 = \frac{x_2 + \cdots + x_{l+1}}{l} \\
&\vdots \\
\nu_{n-l+1} &= \frac{1}{(l-1)\Delta t}\sum_{j=n-l+1}^{n}\left(x_j - \bar{x}_{n-l+1}\right)^2, \quad \bar{x}_{n-l+1} = \frac{x_{n-l+1} + \cdots + x_n}{l}.
\end{aligned}$$

We see that the moving window estimation method results in a new time series of $n-l+1$ short term variances. This way of constructing a new time series of short-term variances is quite simple and straightforward. However, it is not clear what the optimal choice of the window length l is. Different choices of l will yield somewhat different variance time series and as a consequence, different parameter estimates. We will get back to this problem when we start the actual parameter estimation.

In addition we need to estimate the initial volatility data point ν_0, which is required in connection with simulation of the volatility process of the Heston model. There are at least two possible solutions to this problem. One solution is to use the estimated variance of the first window of the moving window estimation process. A problem with this approach is that the estimate we obtain, could turn out to be quite a long distance from the estimate of the long term mean θ. Since the volatility process of the Heston model is a mean reverting process, this could lead to undiserable initial behaviour of a discretized simulation of the volatility process. A better solution is based on the fact that a CIR-process has a stationary distribution. The stationary distribution of the volatility process can be shown to be a gamma distribution with shape parameter $2\kappa\theta/\xi^2$ and scale parameter $\xi^2/2\kappa$ [2]. This implies an expected value of θ, which is the long-term mean of the variance process, as could be expected. As stated in subsection 2.1.2, because of the way a CIR-process is constructed, it always has a tendency to drift towards its long-term mean. As such, an estimate of the long-term mean θ is also a neutral estimate of the initial volatility ν_0.

4.3.3 Estimation of ξ and ρ

The parameter ξ is the so-called volatility of the volatility. Given a time series of short term volatilities, a natural estimate of ξ is simply the sample standard deviation or the volatility of this time series.

The parameter ρ determines the correlation between the Brownian motion of the risky asset and the Brownian motion of the stochastic variance. As such, ρ represents the relationship between the price change of the risky asset and the change of volatility, or in other words, the relationship between the derivatives (in the discrete sense). For parameter estimation, we will use the index price data of the OBX index. A measure of the index price changes of the OBX index are the log returns, and a measure of the changes of the variance time series are the first order differences of the series. An estimate of ρ will be given as the correlation between the log returns and the first order differences.

Regarding the correlation between risky asset price change and volatility change, what can we expect? The plot of figure 4.3 shows the 1st order differences of the 5-

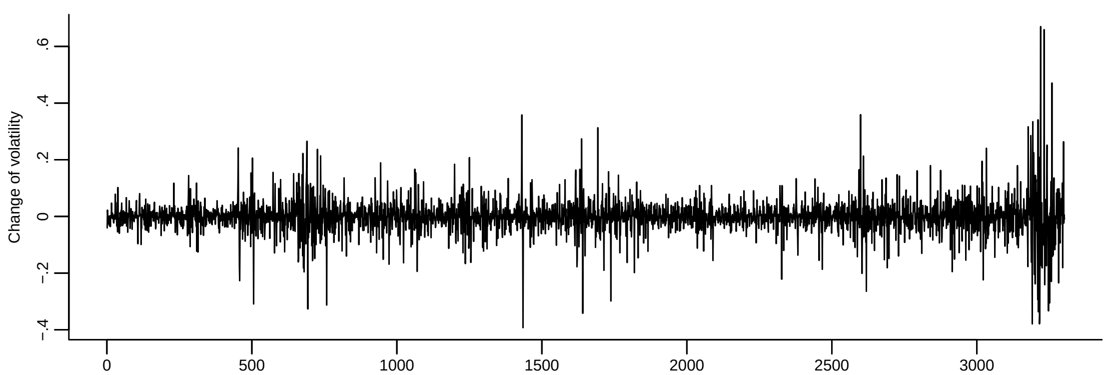


Figure 4.3: 1st order differences of annualized 5-day volatilities.

day volatilities of the OBX log returns. If we compare this plot with the OBX log returns of figure 4.2, it becomes clear that there is a positive correlation between the absolute sizes of change. If there is a correlation between the directions of change, is however not clear. Research suggests that in most of the industrialized countries, the relationship between stock price returns and volatility is weak [13].

4.3.4 Doing the calibration

The Euler approximation 4.6 of the volatility process is also the model that we will use for simulating the stochastic volatility of simulation model IV in the next chapter. What is the right choice of window length? The author of this thesis did unfortunately not succeed in finding any articles or other sources that address this problem. As a consequence we need to make an uneducated a priori choice of window length and five seems like a conservative choice. Other choices of window length are however available. A small range of window lengths along with the corresponding parameter estimates are given in table 4.2.

Window	Parameter estimate				
length	ν_0	κ	θ	ξ	ρ
2	6.3212×10^{-2}	1377.4886	6.3212×10^{-2}	.2292	3.7391×10^{-2}
3	6.4767×10^{-2}	844.6233	6.4767×10^{-2}	.1165	1.9363×10^{-2}
4	6.6105×10^{-2}	599.2981	6.6105×10^{-2}	.0775	12.9317×10^{-2}
5	6.7456×10^{-2}	320.1192	6.7456×10^{-2}	.0590	2.6706×10^{-2}
6	6.8752×10^{-2}	214.3306	6.8752×10^{-2}	.0511	4.2394×10^{-2}
7	6.9074×10^{-2}	170.0703	6.9074×10^{-2}	.0409	7.7981×10^{-2}

Table 4.2: Results of the calibration of the Heston model.

Table 4.2 summarizes the results of the calibration of the Heston model. We observe that the estimates of the parameters ν_0, θ and ρ are not very sensitive to the choice of window length. The estimates for κ and ξ are on the other hand, very sensitive. In other words, there is a clear relation between choice of window length and the intensity of the mean reversion tendency and the volatility of the volatility. Short window lengths are associated with high estimates of κ and ξ. As a direct consequence of the way that the SDE (2.4) of the volatility of the Heston model is defined, higher estimates of κ will result in a more volatile behaviour of the volatility process ν_t itself, since the tendency to revert towards the mean θ will be stronger. As for the volatility of the volatility ξ, higher estimates of this parameter will obviously result in a more volatile process. These facts along with the plots of figure 4.4 explain why there is a negative correlation between window length and the estimates of κ and ξ. The plots of figure 4.4 show the estimated short term volatilities as a result of (a) window length equal to one, and (b) window length equal to seven. It is clear that the short term volatilities that results from a choice of window length equal to two are more spiked and volatile,

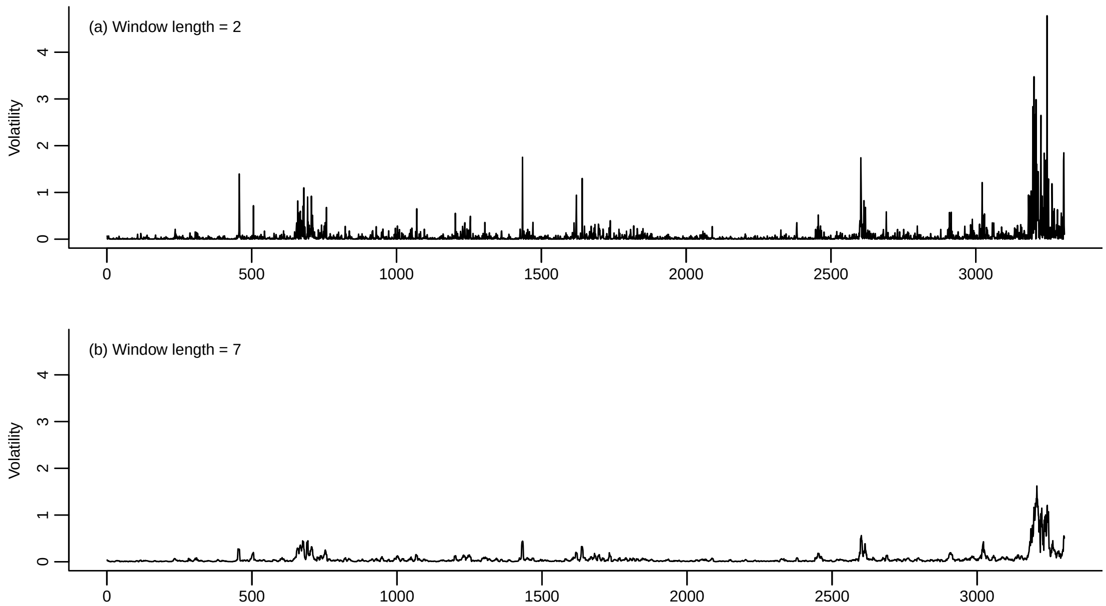


Figure 4.4: Short-term volatilities as a result of (a) window length equal to two, and (b) window length equal to seven.

whereas the short term volatilities that results from a choice of window length equal to seven are more smoothed out and less volatile. We observe how these features of the choices of window length are reflected in the parameter estimates of table 4.2.

Note that in relation with simulation model IV in the next section, we will simulate the stochastic volatility process using the same Euler approximation (4.6) of the SDE of the volatility as was used to create the linear regression model (4.7) of this section. The Euler approximation (4.6) is dependent on the size of the time increment $\Delta t_k = \delta$, which in turn implies that the linear regression model and the estimator of κ (4.8) also are time dependent. The estimate of κ needs to be scaled according to the size of the time increment. As stated earlier, we measure time in years. In the simulations, the variables will be updated hourly. Assuming 252 trading days in one year, hourly updates imply $\delta = 1/6048$. So, the estimates of κ of table 4.2 need to be interpreted in light of the size of the equidistant time increment.

Chapter 5

Simulation

5.1 Introduction

As mentioned in the introduction chapter (chapter 1), the goal of this thesis is to simulate the development of the value of a portfolio with two investment options, namely a risky asset and a risk-free asset. As already stated, the optimal strategy is for the portfolio manager or the investor to keep a constant fraction of her wealth in the risky asset and consequently a constant fraction in the risk-free asset. In Merton's portfolio problem, the investor is allowed to rebalance the portfolio continuously in time. The question is, how will this strategy perform in a more realistic, discrete time scenario?

5.2 Basic simulation model

5.2.1 Introduction

In this section we will consider the most basic portfolio model, that is a portfolio model with constant parameters and no transaction costs. This means that we assume that the dynamics of the value of the risk-free asset follows the deterministic differential equation (3.1) and that the dynamics of the value of the risky asset follows the SDE (2.1). As shown in chapter 3, by assuming these dynamics for the risky and risk-free asset, we obtain an SDE for the portfolio value given by equation (3.2), where u_t is the control function at time t. The control function is the actual trading strategy or allocation strategy, that is, at time t, the investor must allocate a fraction u_t of the total wealth V_t in the risky asset and $1 - u_t$ in the risk-free asset. The optimal strategy, which we will use, is to hold a constant fraction u^* of the wealth in the risky asset, that is we assume that $u_t = u^*$. The dynamics of the value of the optimal portfolio is then given by

$$dV_t = (\mu u^* + r(1 - u^*))V_t dt + \sigma u^* V_t dB_t. \tag{5.1}$$

It can be shown that the solution of this SDE is

$$V_t = V_0 \exp\left(\left(\mu u^* + r(1 - u^*) - \frac{1}{2}\sigma^2 {u^*}^2\right) t + \sigma u^* B_t\right). \tag{5.2}$$

This is the exact solution of the portfolio value and we will refer to V_t as the theoretical portfolio value at time t. The theoretical portfolio value will serve as a baseline for comparison.

The time domain in which we want to simulate the development of the portfolio value, is constrained by an initial time $t_0 = 0$ and a terminal time $t_n = T$. Let

$$0 = t_0 < t_1 < t_2 < \cdots < t_n = T \tag{5.3}$$

be the time discretization of this time domain and let $\mathcal{T} = \{t_0, t_1, \ldots, t_n\}$ denote the complete set of time points within the time interval. The time increments are defined as $\Delta t_k = t_{k+1} - t_k$. We will assume equidistant discretization times, i.e. $\Delta t_k = \delta$. The Euler-Maruyama approximation of the SDE (5.1) is defined as

$$V_{k+1} = V_k + (\mu u^* + r(1 - u^*))V_k \delta + \sigma u^* V_k \Delta B_k.$$

Observing that $V_k = u^* V_k + (1 - u^*)V_k$, the approximation can be rewritten as

$$V_{k+1} = \underbrace{u^* V_k}_{(i)} \underbrace{(1 + \mu\delta + \sigma \Delta B_k)}_{(ii)} + \underbrace{(1 - u^*)V_k}_{(iii)} \underbrace{(1 + r\delta)}_{(iv)}.$$

We recognize (i) as the value of the risky asset investment at time t_k and (ii) as one plus the return on the risky asset between time t_k and t_{k+1}. Likewise, we recognize (iii) as the value of the risk-free asset investment at time t_k and (iv) as one plus the return on the risk-free asset. This approximation will serve as a template for the simulation models. The approximation describes a recursive method of simulation. It is the correct method for simulating the portfolio value at discrete time points, because the portfolio value at each time point is the wealth at the preceding time point plus the return from the amount invested in the risk-free asset plus the return from the amount invested in the risky asset.

The amount invested in the risk-free and the risky asset will follow the optimal trading strategy, but the rebalancings of the portfolio will not necessarily happen at each and every time point. In the simulations one important task is to compare different rebalancing strategies, such as daily rebalancings, monthly rebalancings et cetera. Given a time interval and a set of time points according to a discretization of the time interval, we will achieve this by rebalancing the portfolio at time points according to a subset of the time points. Because of this the simulated portfolio value will be calculated by using a somewhat modified Euler-Maruyama approximation scheme, which will be formulated in the next section.

To make a notational distinction between theoretical quantities and simulated quantities where it is necessary, simulated quantities will be indicated with a tilde. For example, the simulated portfolio value at time t_k will be given as $\tilde{V}_k$. The set of rebalancing time points is given by $\mathcal{T}^{\text{reb}} = \{t_0, t_\epsilon, t_{2\epsilon}, \ldots, t_n\}$ which constitutes a subset of the complete set of time points, i.e. $\mathcal{T}^{\text{reb}} \subseteq \mathcal{T}$. The positive integer ϵ denotes the distance between rebalancing time indices and for simplicity we will assume that ϵ is a divisor of n. Assume also that the last rebalancing time point relative to the time point in which we want to simulate the wealth is given by t_{k^*}. The total portfolio value can be seen as a sum consisting of two values: the value of the investment in the risky asset and the value of the investment in the risk free asset. The value of the risky asset investment at time t_k is denoted by $\tilde{V}_k^S$, the value of the risk free asset investment is denoted by $\tilde{V}_k^R$ and the total portfolio value is denoted by $\tilde{V}_k$. In addition, Q_k denotes the amount that needs to be subtracted from the risky asset investment and added to the risk free asset investment, that is the transaction quantity, at each rebalancing time point to rebalance the portfolio in accordance with the optimal strategy. This implies that Q_k also can be negative. A negative transaction just means that the risk-free investment needs to be reduced and the risky investment increased, to put the portfolio in a state of balance according to the optimal strategy.

5.2.2 Simulation model I

We will refer to the basic and initial simulation model as simulation model I. The model is defined by the following set of equations:

Simulation model I
Transaction costs: none
Volatility: constant

$$
\begin{aligned}
\tilde{V}_k'^S &= u^* \tilde{V}_{k^*} \prod_{j=k^*}^{k-1} (1 + \mu\delta + \sigma \Delta B_j) \\
\tilde{V}_k'^R &= (1-u^*) \tilde{V}_{k^*} (1 + r\delta)^{k-k^*} \\
Q_k &= (1-u^*) \tilde{V}_k'^S - u^* \tilde{V}_k'^R \\
\tilde{V}_k^S &= \begin{cases} \tilde{V}_k'^S - Q_k, & t_k \in \mathcal{T}^{\text{reb}} \\ \tilde{V}_k'^S, & \text{otherwise} \end{cases} \\
\tilde{V}_k^R &= \begin{cases} \tilde{V}_k'^R + Q_k, & t_k \in \mathcal{T}^{\text{reb}} \\ \tilde{V}_k'^R, & \text{otherwise} \end{cases} \\
\tilde{V}_k &= \tilde{V}_k^S + \tilde{V}_k^R.
\end{aligned}
$$

$\tilde{V}_k'^S$ represents the value of the risky asset investment at time t_k. At rebalancing time points, $\tilde{V}_k'^S$ will represent the value of the risky asset investment before the portfolio is rebalanced. It is defined as the value of the risky asset investment at the preceding rebalancing time point t_k^* times the product of one plus the return on the risky asset of each time interval since the preceding rebalancing time point, that is the value after compounding. The value of the risk-free investment $\tilde{V}_k'^R$ at time t_k is calculated using the same rationale. What about Q_k? Assume that t_k is a rebalancing time point. For the portfolio to become rebalanced according to u^*, it is required that $\tilde{V}_k^S = u^* \tilde{V}_k = u^*(\tilde{V}_k'^S + \tilde{V}_k'^R)$. From this it is clear that

$$
\begin{aligned}
Q_k &= \tilde{V}_k'^S - u^* \left(\tilde{V}_k'^S + \tilde{V}_k'^R \right) = (1-u^*) \tilde{V}_k'^S - u^* \tilde{V}_k'^R \\
&= u^*(1-u^*) \tilde{V}_{k^*} \left(\prod_{j=k^*}^{k-1} (1 + \mu\delta + \sigma\Delta B_j) - (1 + r\delta)^{k-k^*} \right) \\
&= u^*(1-u^*) \tilde{V}_{k^*} \left(\left(\prod_{j=k^*}^{k-1} (1 + \mu\delta + \sigma\Delta B_j) - 1 \right) - \left((1 + r\delta)^{k-k^*} - 1 \right) \right).
\end{aligned}
\tag{5.4}
$$

Notice that the sign of Q_k is only determined by the difference between the returns on each asset investment since the last rebalancing time point t_{k^*}, which reflects the fact that the balance of the portfolio is preserved as long as the returns are equal. Hence, a difference in returns at a rebalancing time point requires the portfolio to be rebalanced.

Since Q_k is both added and subtracted at the same time at each rebalancing time point, it doesn't affect the total value of the portfolio. For the sake of the simulation of the portfolio value it is not even necessary to calculate Q_k because we know that $\tilde{V}_k^S = u^*\tilde{V}_k$ and that $\tilde{V}_k^R = (1-u^*)\tilde{V}_k$. What this means is that the simulation model can be stated in a more compact way:

$$\tilde{V}_k = u^*\tilde{V}_{k^*}\prod_{j=k^*}^{k-1}(1+\mu\delta+\sigma\Delta B_j) + (1-u^*)\tilde{V}_{k^*}(1+r\delta)^{k-k^*}. \tag{5.5}$$

This compact restatement of the simulation model is more ideal as a basis for implementation of fast simulation routines in R.

To illustrate how the simulation model works we will look at an example.

Example 5.2.1 Assume that the portfolio is rebalanced at every 3rd time point, which implies $\epsilon = 3$. The subset of rebalancing time points is as a result given as $\mathcal{T}^{\text{reb}} = \{t_0, t_3, t_6, \ldots, t_n\}$. Also assume that $\tilde{V}_0 = V_0$ and that $y_k = \mu\delta + \sigma\Delta B_k$ which is the return on the amount invested in the risky asset between time points t_k and t_{k+1}. Then according to (5.5) we have that

$$\begin{aligned}
&\tilde{V}_1 = u^*V_0(1+y_0) + (1-u^*)V_0(1+r\delta)\\
&\tilde{V}_2 = u^*V_0(1+y_0)(1+y_1) + (1-u^*)V_0(1+r\delta)^2\\
&\begin{cases} Q_3 = (1-u^*)u^*V_0(1+y_0)(1+y_1)(1+y_2) - u^*(1-u^*)V_0(1+r\delta)^3\\ \tilde{V}_3 = u^*V_0(1+y_0)(1+y_1)(1+y_2) + (1-u^*)V_0(1+r\delta)^3\end{cases}\\
&\tilde{V}_4 = u^*\tilde{V}_3(1+y_3) + (1-u^*)\tilde{V}_3(1+r\delta)\\
&\vdots\\
&\begin{cases} Q_n = (1-u^*)u^*V_{n\text{-}3}(1+y_{n\text{-}3})(1+y_{n\text{-}3})(1+y_{n\text{-}3}) - u^*(1-u^*)V_{n\text{-}3}(1+r\delta)^3\\ \tilde{V}_n = u^*\tilde{V}_{n\text{-}3}(1+y_{n\text{-}3})(1+y_{n\text{-}3})(1+y_{n\text{-}3}) + (1-u^*)\tilde{V}_{n\text{-}3}(1+r\delta)^3.\end{cases}
\end{aligned}$$

5.2.3 Loss of utility

The portfolio manager's utility of the wealth is given by a utility function (3.7), which is a utility function from the family of power functions. To measure the loss of utility at terminal time T, we simply calculate the difference between the utility of the theoretical wealth $U(V_T)$ with the utility of the simulated wealth $U(\tilde{V}_T)$, that is, the measure of the loss of utility will be given by

$$U(V_T) - U(\tilde{V}_T). \tag{5.6}$$

5.2.4 Simulation test run

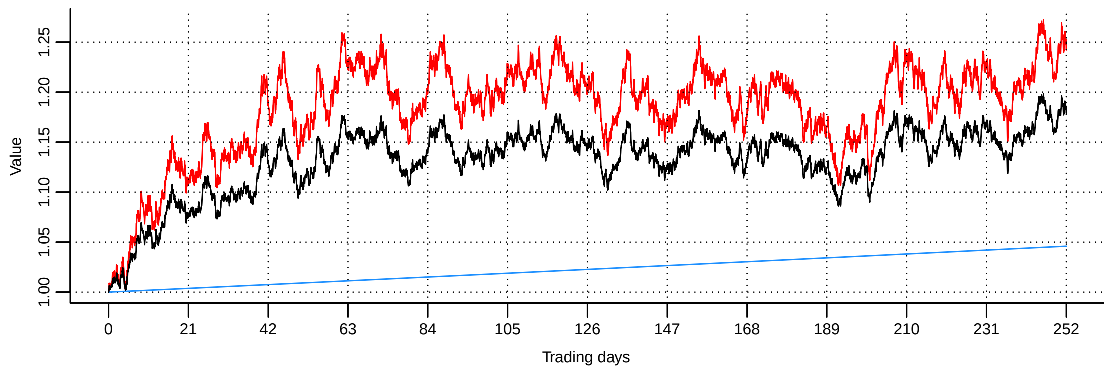
Parameter	Value	Parameter	Value
V_0	1	c_B	24
μ	.0657	c_P	12/252
σ	.2537	n	6048
r	.0449	δ	1/6048
γ	.5255		

Table 5.1: Example of a complete set of simulation input parameter values.

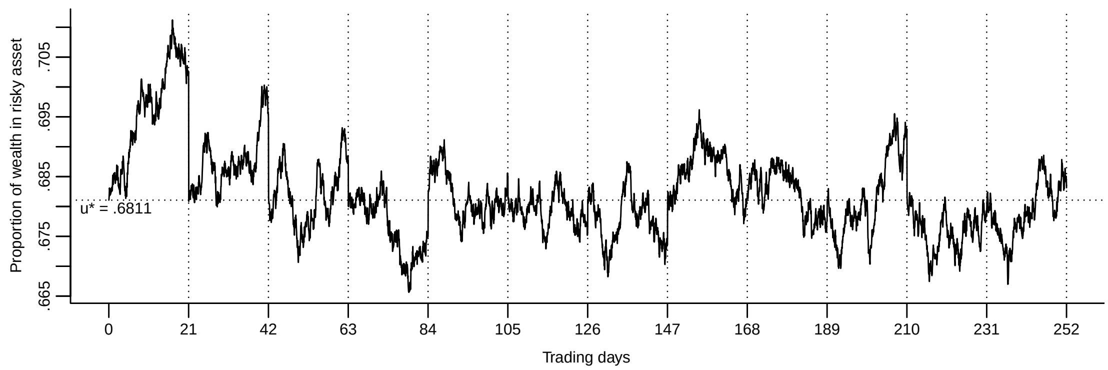
All of the simulations in this thesis were implemented and executed in the statistical language R. Initially, to get a feel for the behaviour of the simulations, we will implement a single simulation test run. The simulation function has several different input parameters: The parameters in "Merton's portfolio problem", that is, the initial wealth V_0, the continuously compounding interest rate r of the risk-free asset, the drift μ and volatility σ of the risky asset, the risk aversion parameter γ and the optimal investment strategy u^*. As for the specific choices of these parameter values, these are of course the parameter estimates calculated in chapter 4. These estimates yield $u^* = .6811$.

For the simulations we also need to define additional parameters. These parameters are n, which denotes the total number of time points in one year, δ, which denotes the size of the equidistant time increments, c_B, which denotes the number of daily changes of the risky asset, that is, the number of daily increments of the simulated Brownian motion underlying the stochastic dynamics of the risky asset, and c_P, which denotes the number of daily portfolio rebalancings the portfolio manager may do. This implies $\epsilon = c_B/c_P$. These are basically simulation specific parameters. Concerning the choices of these parameter values, the time will be measured in years and we will follow the conventional assumption of 252 trading days in one year. This means that one year will be discretized into $n = 252c_B$ time points and that $t_n = T = 1$, which implies $\delta = 1/n$. Assume that $c_B = 24$, which means that the risky asset will change value at an hourly basis. A consequence of this choice is $n = 6048$ and $\delta = 1/6048$. We will in the test run assume monthly rebalancings, that is a total 12 rebalancings in one year, which implies $c_P = 12/252$. The complete set of parameter values required for a simulation run, are given in table 5.1.

Figure 5.1 shows the results of a single simulation run with parameter values according to table 5.1. The vertical dotted lines indicate the rebalancing time points related to the number of trading days. Monthly rebalancings imply that the portfolio is rebalanced every 21st day. The plot of subfigure (a) shows the development of the risky asset value (red), the risk-free asset value (blue) and



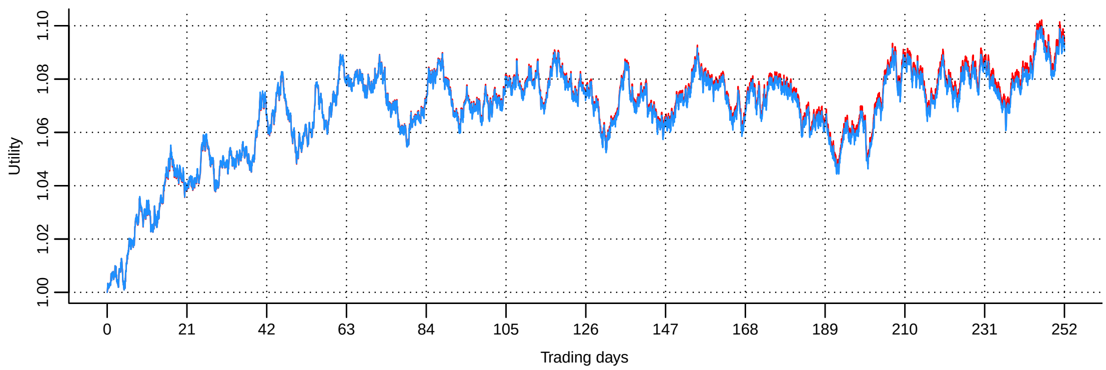
(a) Development of risky asset value (red), risk-free asset value (blue) and portfolio value (black).



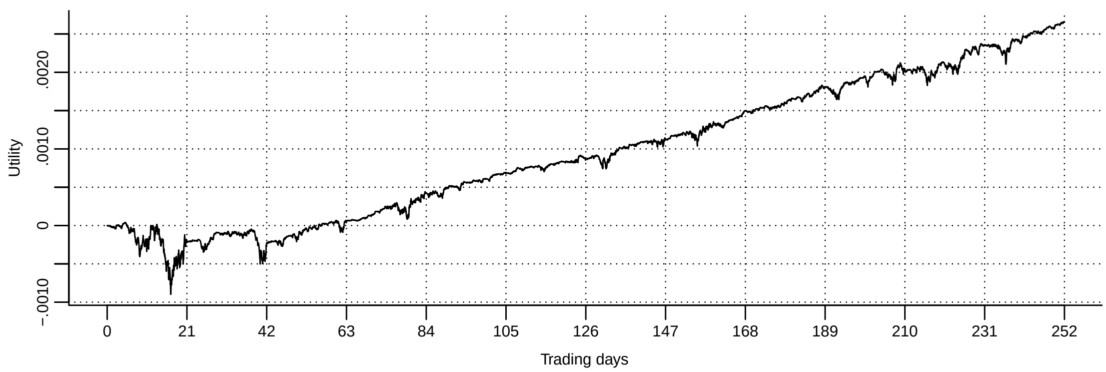
(b) Proportion of the wealth invested in the risky asset.

Figure 5.1: Results of the test run.

the simulated portfolio value (black). It is clear that for this particular simulation run, the development of the risky asset is far superior compared to the development of the risk-free asset. This is reflected in the plot of subfigure (b), which shows the size of the proportion of the wealth invested in the risky asset. We see that just before the first rebalancing of the portfolio at trading day 21, the strong development of the risky asset causes the proportion of the risky asset investment to deviate considerably from the optimal proportion u^*. We also see how the portfolio is adjusted at each rebalancing time point, to match the optimal allocation proportion. Figure 5.2 shows plots concerning the utility of the wealth of the investor. In subfigure (a) the utility of the simulated wealth (5.5) is plotted in blue on top of the utility of the theoretical wealth (5.2), which is plotted in red. As the plots of subfigure (a) shows, the value of the simulated wealth follows the theoretical wealth very closely, but clearly, there are small differences. These differences are magnified in the in subfigure (b), which shows the difference in utility at each time point. We observe that for this specific simulation run, the difference is relatively small but that it is increasing with time. In



(a) The utility of the theoretical (red) and the simulated wealth (blue).



(b) The difference in utility.

Figure 5.2: One single simulation run over 252 trading days.

some time intervals, there also seems to be a correlation not only with time, but also with the utility of both the theoretical and the simulated wealth. However, one simulation is of course not sufficient to draw any serious conclusions about the loss of utility. To be able to do that, it is a good idea to consider the sample mean of the simulated loss of utilities, which is exactly what we will do in the next section.

5.2.5 Mean loss of utility

Figure 5.3 shows the results after calculating the terminal losses of utility (5.6) of one million simulation runs with parameter values according to table 5.1. The plot of figure 5.3 suggests that the mean loss of utility might be slightly less than zero due to the fact that many of the negative losses are larger in absolute value compared to the positive losses. However, the histogram of figure 5.4 (a) shows that the distribution of losses of utilities is skewed to the left with a global maximum in the positive region and with the sample mean close to zero. Also, the left

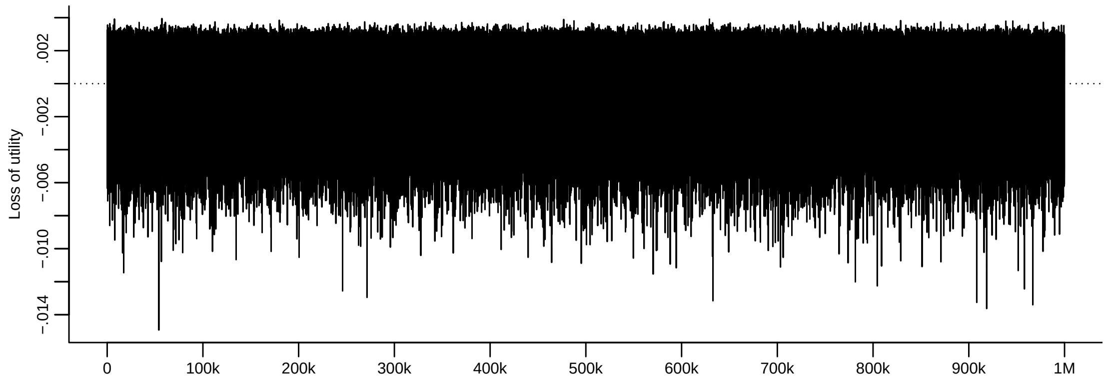
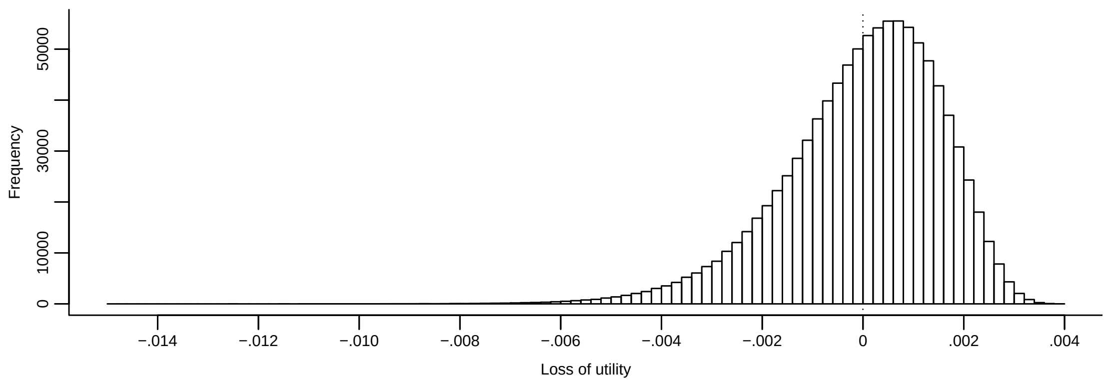
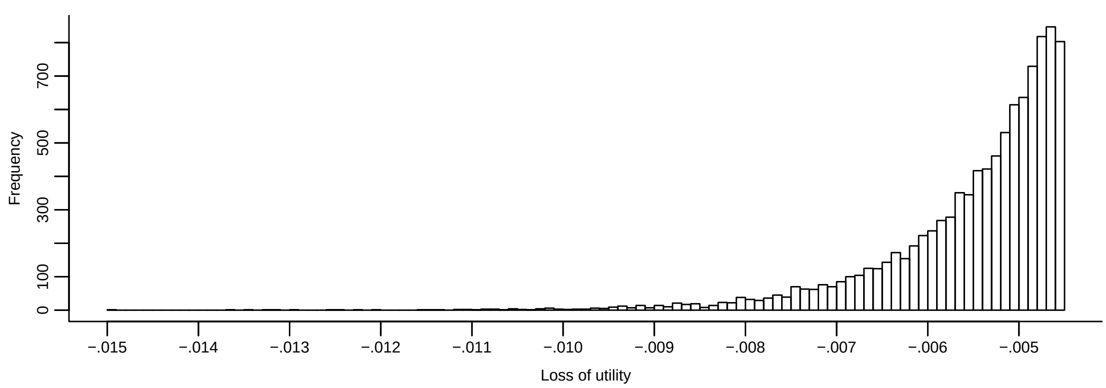


Figure 5.3: One million simulation runs.



(a) The distribution of the losses of utility.



(b) The distribution of the lower one percent of the losses of utility.

Figure 5.4: Distributions of one million simulation runs.

tail is extremely long and narrow. This left tail behaviour is magnified in figure 5.4 (b) and shows that in relative magnitude, some of the negative losses are very large compared to the main bulk of losses, but that they are extremely rare. The histograms of figure 5.5 show how this left tail behaviour is related to the choice

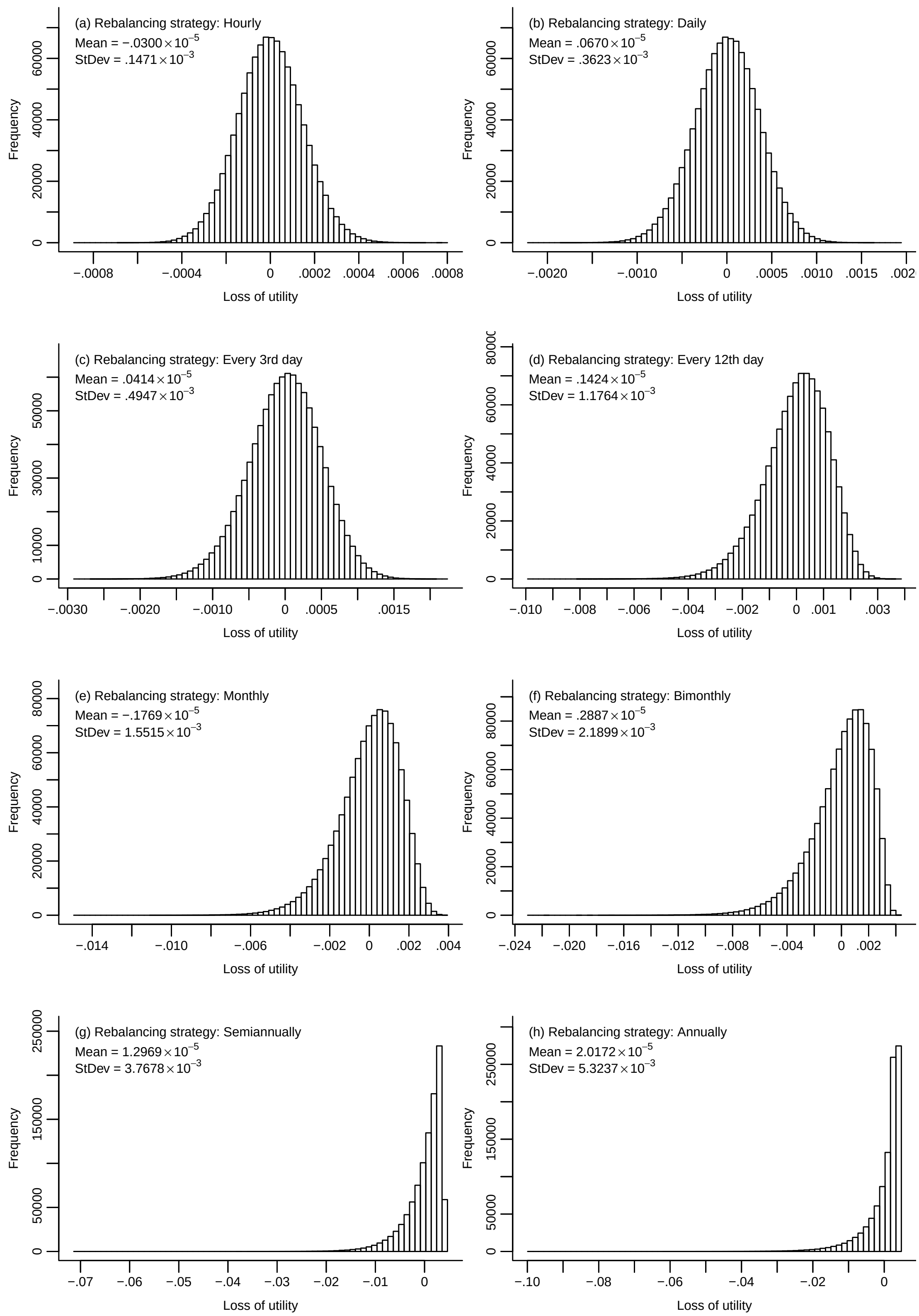


Figure 5.5: The distributions of the losses of utility of the different rebalancing strategies.

of rebalancing strategy. The distribution of the losses of utility of the hourly-rebalancing strategy is similar to a normal distribution, whereas the distribution of the losses of utility of the annual-rebalancing strategy is extremely skewed to the left. As for the distributions of the intermediate rebalancing strategies, they describe an evolution from gaussian symmetry towards negative skewness. Remember that the loss of utility is the utility of the theoretical portfolio value minus the utility of the simulated portfolio value. This means that a negative loss of utility is equivalent to a gain of utility. We observe that on rare occasions, the gain of utility for the annual-rebalancing strategy can be quite large. .1 is a large gain of utility considering that the initial utility is equal to one. On the other, the maximum loss of utility is also larger for the annual-strategy: 4.7914×10^{-3} versus 1.9437×10^{-3} for the daily-strategy. On average, the daily-strategy seems to be a little bit better.

The plots of figure 5.6 show how the simulated losses of utility sample means develop as the number of simulations increases. The outer grey lines mark the lower and upper limits of a 95% confidence interval of the estimated mean, calculated under the assumption of a normally distributed mean in accordance with the central limit theorem. We observe that for all the different rebalancing strategies, the mean loss of utility seems to converge towards a value very close to zero. Considering that the strategy of hourly rebalancings is in fact the direct Euler-approximation of the portfolio value, it is not surprising that the mean loss of utility for this specific strategy is close to zero. The mean loss of utility is however very small for all the rebalancing strategies and for all practical purposes approximately equal to zero. With a significance level of 5%, the mean losses of utility for the semiannual and the annual strategy are significantly different from zero, but they are still extremely small. This result suggests that by the law of large numbers, the sample mean utilities of the simulated portfolio values will converge toward the true expected utility of the theoretical portfolio value, that is,

$$\mathrm{E}[U(V_t)] = V_0^\gamma \exp((\mu u^* + r(1-u^*) - \frac{1}{2}\sigma^2 u^{*2})t)^\gamma c$$

with

$$c = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} \exp(\sigma u^* \sqrt{t} x)^\gamma \exp\left(-\frac{1}{2}x^2\right) dx.$$

The plot of figure 5.7 and table 5.2 confirms the picture we have seen so far. The plot also provides more detail into the relationship between rebalancing strategy and the measuring uncertainty of the loss of utility. The plot shows the mean losses of utility as the curve in the middle with accompanying confidence limits of 95% confidence intervals. We observe that frequent rebalancings are associated with narrow confidence intervals and that strategies with just a few

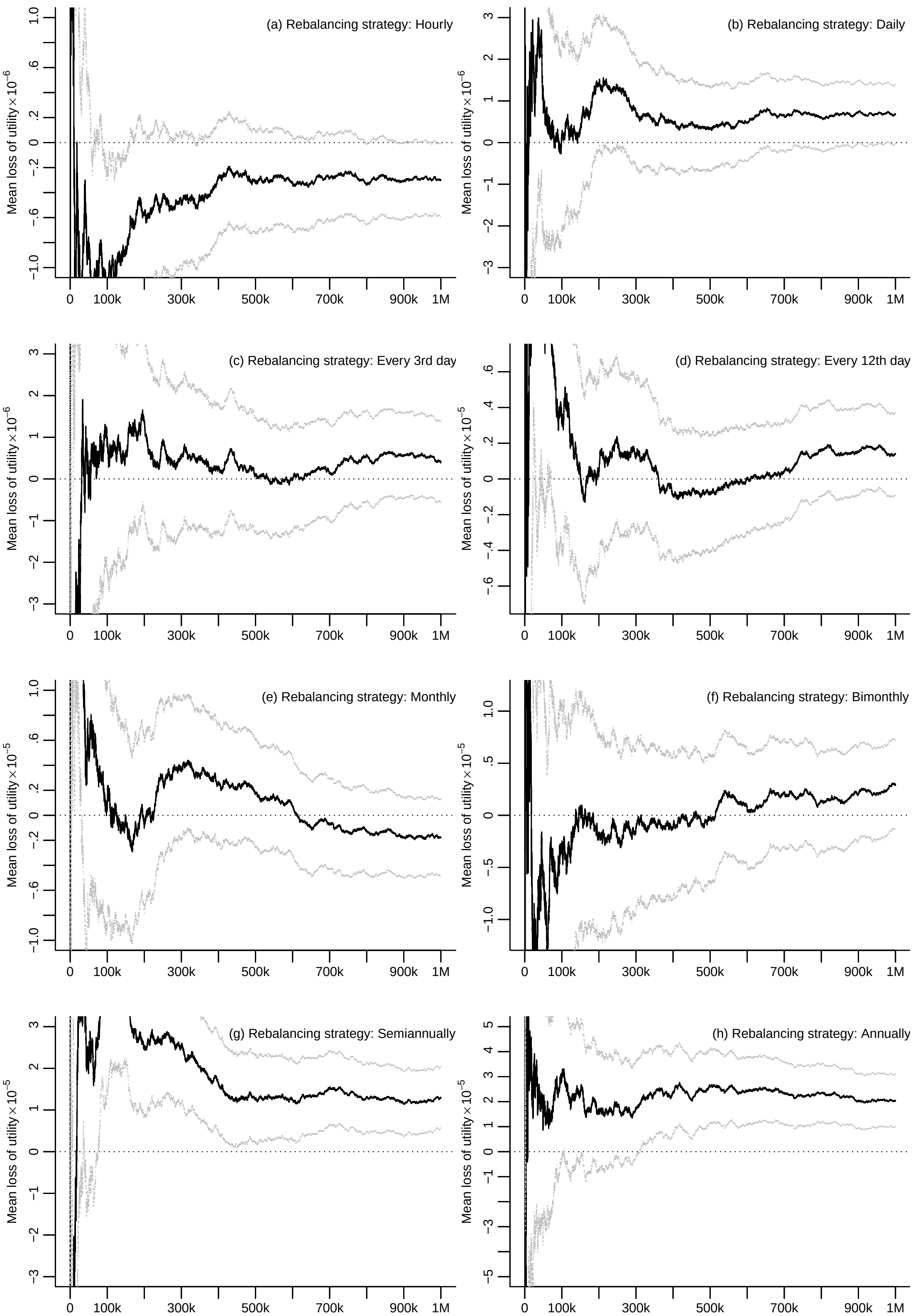


Figure 5.6: The mean losses of utility plotted against rebalancing strategies.

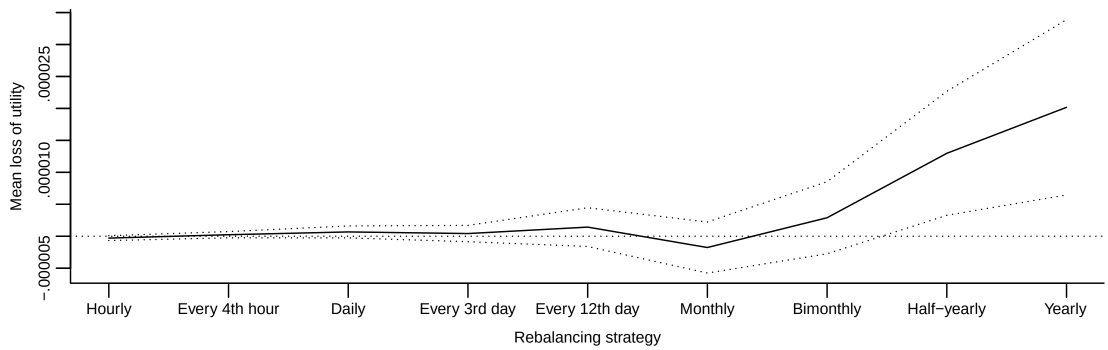


Figure 5.7: Mean loss of utility vs rebalancing interval.

	Simulation model		Sample means				StDev
			Term. wealth	Loss of wealth	Term. utility	Loss of utility	Loss of utility
Rebalancing strategy	Hourly	Th	1.0609	0	1.0277	0	0
		Sim	1.0609	$-.0476\times10^{-5}$	1.0277	$-.0300\times10^{-5}$	$.1471\times10^{-3}$
	Every 4th hour	Th	1.0609	0	1.0277	0	0
		Sim	1.0609	$.0539\times10^{-5}$	1.0277	$.0233\times10^{-5}$	$.1894\times10^{-3}$
	Daily	Th	1.0610	0	1.0278	0	0
		Sim	1.0610	$.1044\times10^{-5}$	1.0278	$.0670\times10^{-5}$	$.3623\times10^{-3}$
	Every 3rd day	Th	1.0607	0	1.0276	0	0
		Sim	1.0607	$.0364\times10^{-5}$	1.0276	$.0414\times10^{-5}$	$.4947\times10^{-3}$
	Every 12th day	Th	1.0607	0	1.0276	0	0
		Sim	1.0607	$-.0534\times10^{-5}$	1.0276	$.1424\times10^{-5}$	1.1764×10^{-3}
	Monthly	Th	1.0608	0	1.0277	0	0
		Sim	1.0608	$-.9413\times10^{-5}$	1.0277	$-.1769\times10^{-5}$	1.5515×10^{-3}
	Bimonthly	Th	1.0610	0	1.0278	0	0
		Sim	1.0610	$-.6428\times10^{-5}$	1.0278	$.2887\times10^{-5}$	2.1899×10^{-3}
	Seminannualy	Th	1.0606	0	1.0276	0	0
		Sim	1.0606	-1.0460×10^{-5}	1.0275	1.2969×10^{-5}	3.7678×10^{-3}
	Annually	Th	1.0607	0	1.0276	0	0
		Sim	1.0607	-3.1841×10^{-5}	1.0276	2.0172×10^{-5}	5.3237×10^{-3}

Table 5.2: Mean losses of utility and other related statistics.

rebalancings during one year are associated with wider confidence intervals. This is of course a rather obvious feature considering that the potential size of the difference in utility will increase as the time since the last rebalancing took place, increases, leading to potentially larger differences in utility and hence, larger variance. This tells us that although the expected utility of an investors portfolio value will lie close to the expected utility of the theoretical wealth, as concluded above, the uncertainty of this prediction will increase as the time interval between rebalancings increases. This also points to the fact that strategies of infrequent rebalancings involve higher risk to the investor. This is not surprising considering the fact that the optimal strategy of holding a constant fraction of the wealth in the risky asset, is meant to limit the risk.

5.2.6 Portfolio return and Sharpe ratio

To compare the performances of the different rebalancing strategies we will employ the Sharpe ratio. As described earlier in section 2.2, the Sharpe ratio measures the excess return per unit of risk of an investment portfolio. Also, there are two versions of the Sharpe Ratio, the ex ante version and the ex post version. To compare the rebalancing strategies we must use the ex post version. The ex ante version will serve as a baseline for the ex post Sharpe ratios. For both versions, the natural benchmark is the risk free rate of return, r. It can be shown that

$$\mathrm{E}[X_t] = \left(\mu u^* + r(1-u^*) - \frac{1}{2}\sigma^2 u^{*2}\right) t, \tag{5.7}$$

$$\mathrm{Var}[X_t] = \sigma^2 u^{*2} t. \tag{5.8}$$

Substituting these expressions along with r into (2.6) yields

$$SR_t^{ea} = \frac{(\mu u^* + r(1-u^*) - \frac{1}{2}\sigma^2 u^{*2})t - r}{\sigma u^* \sqrt{t}}.$$

After one year, that is at time $t = 1$, we have that $SR_1^{ea} = -4.4060 \times 10^{-3}$, which is a negative Sharpe ratio. Does this mean that the expected return of the portfolio is less than the expected value of the risk free asset? No, not necessarily, because in this thesis we use log returns instead of arithmetic returns. If we consider the expected theoretical wealth of the portfolio at time t,

$$E[V_t] = V_0 \exp((\mu u^* + r(1-u^*))t), \tag{5.9}$$

we observe that the return of this quantity is $\exp((\mu u^* + r(1-u^*))t) - 1$, which also is the expected arithmetic return of the portfolio. The (expected) arithmetic return of an investment in the risk-free asset is $\exp(rt) - 1$. Thus, maybe the difference $(\mu u^* + r(1-u^*))t - rt$ would have been a more natural measure of the expected excess return of the portfolio investment versus the risk free investment. Using log returns we get an extra term $-.5\sigma^2 u^{*2} t$ in the expected excess return, which gives a negative ex ante Sharpe ratio. However, the main purpose of using the Sharpe ratio in this thesis is not to compare the portfolio performance against the risk free asset, but to compare the different rebalancing strategies relative to each other. In this context a negative Sharpe ratio is not a considerable problem.

Regarding the comparison of the different rebalancing strategies, we are interested in comparing the ex post Sharpe ratios at terminal time, that is after one year. In order to make the ex post Sharpe ratio comparable to the ex ante Sharpe ratio, we need to annualize the ex post Sharpe ratio. This is achieved by using the annualized estimators of chapter 4, that is, equation (4.1) and equation (4.2).

As mentioned earlier, log returns give a notational advantage over arithmetic returns. For instance, a series of log returns form a telescoping series. Assuming that $\tilde{v}_0 = 1$, the telescoping property of the log returns yields $\tilde{\mu}_x = \frac{1}{n\Delta t} \log \tilde{v}_n$. Further, the time increments are equidistant and assumed equal to $\delta = 1/n$. This means that $\tilde{\mu}_x = \log \tilde{v}_n$. As for the continuously compounding risk free return, it is for each time interval constant and approximately equal to r/n. As a consequence of this, the annualized risk free return is equal to r, as it should be. With the sample mean of the log returns equal to $\widetilde{\log}(v_n)/n$, we have that the annualized sample standard deviation is formulated as

$$\hat{\sigma}_x = \sqrt{\frac{n}{n-1} \sum_{k=0}^{n-1} \left(x_k - \frac{\log \tilde{v}_n}{n} \right)^2}.$$

Given a time discretization (5.3), a set of log returns $\{x_k\}_{k=1,\ldots,n}$ and a set of risk free rents $\{r_k\}_{k=1,\ldots,n}$, the annualized ex post Sharpe ratio at time $t = 1$ is defined as

$$SR_n^{\mathrm{a}} = \frac{\log \tilde{v}_n - r}{\sqrt{\frac{n}{n-1} \sum_{k=0}^{n-1} \left(x_k - \frac{\log \tilde{v}_n}{n} \right)^2}}. \tag{5.10}$$

To calculate the ex post Sharpe ratios we will use the same set of data of one million simulation runs for each rebalancing strategy, as was used in estimations of the losses of utility. For each and every simulation run the ex post Sharpe ratio is calculated by using equation (5.10). To compare the strategies we can for instance look at the sample mean of the ex post Sharpe ratios of each strategy.

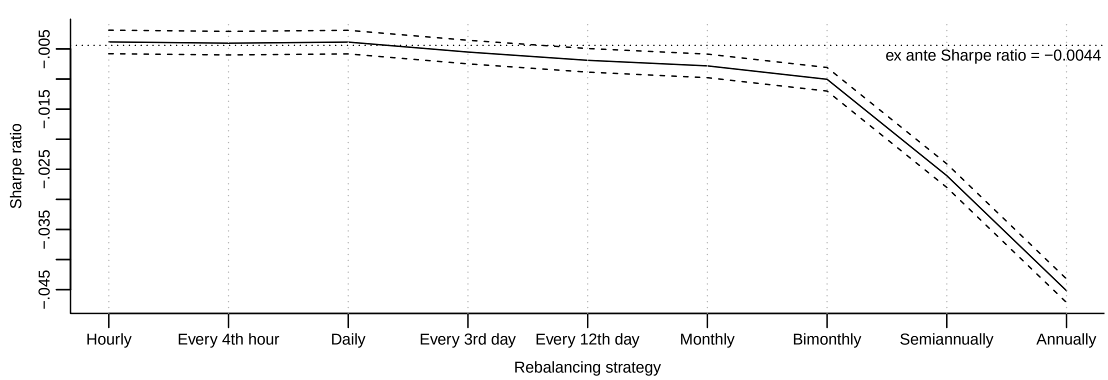


Figure 5.8: Rebalancing strategy versus ex post Sharpe ratio.

Table 5.3 summarizes the different rebalancing strategies' ex post Sharpe ratios. We observe that there is a clear positive correlation between Sharpe ratio and

	Simulation model		Sample means: Terminal log return	Sample means: Vol.	Sample means: Sharpe ratio	Vol. of vol.	Corr.	Rank
Rebalancing strategy	Hourly	Th	4.4230×10^{-2}	.1728	$-.3880\times10^{-2}$	$.1570\times10^{-2}$	0	1
		Sim	4.4230×10^{-2}	.1728	$-.3834\times10^{-2}$	$.1570\times10^{-2}$	-.0047	
	Every 4th hour	Th	4.4197×10^{-2}	.1728	$-.4062\times10^{-2}$	$.1569\times10^{-2}$	-.0006	3
		Sim	4.4197×10^{-2}	.1728	$-.4041\times10^{-2}$	$.1569\times10^{-2}$	-.0031	
	Daily	Th	4.4252×10^{-2}	.1728	$-.3744\times10^{-2}$	$.1569\times10^{-2}$	-.0009	2
		Sim	4.4251×10^{-2}	.1728	$-.3862\times10^{-2}$	$.1569\times10^{-2}$	.0113	
	Every 3rd day	Th	4.3994×10^{-2}	.1728	$-.5245\times10^{-2}$	$.1571\times10^{-2}$	.0004	4
		Sim	4.3994×10^{-2}	.1728	$-.5519\times10^{-2}$	$.1572\times10^{-2}$	.0302	
	Every 12th day	Th	4.4037×10^{-2}	.1728	$-.5012\times10^{-2}$	$.1573\times10^{-2}$	.0019	5
		Sim	4.4036×10^{-2}	.1728	$-.6891\times10^{-2}$	$.1618\times10^{-2}$	.2020	
	Monthly	Th	4.4117×10^{-2}	.1728	$-.4529\times10^{-2}$	$.1571\times10^{-2}$	-.0007	6
		Sim	4.4123×10^{-2}	.1727	$-.7816\times10^{-2}$	$.1705\times10^{-2}$	.3356	
	Bimonthly	Th	4.4320×10^{-2}	.1728	$-.3363\times10^{-2}$	$.1573\times10^{-2}$	.0003	7
		Sim	4.4320×10^{-2}	.1727	-1.0044×10^{-2}	$.2064\times10^{-2}$	.5595	
	Semi-annualy	Th	4.3881×10^{-2}	.1728	$-.5896\times10^{-2}$	$.1570\times10^{-2}$	-.0002	8
		Sim	4.3874×10^{-2}	.1726	-2.6059×10^{-2}	$.4307\times10^{-2}$	.8045	
	Annually	Th	4.4047×10^{-2}	.1728	$-.4920\times10^{-2}$	$.1572\times10^{-2}$	-.0018	9
		Sim	4.4043×10^{-2}	.1722	-4.5186×10^{-2}	$.8155\times10^{-2}$	.8451	

Table 5.3: The Sharpe ratios of the different rebalancing strategies along with other statistics.

rebalancing frequency. The rebalancing strategy which involves hourly rebalancings of the portfolio has the best Sharpe ratio by a slight margin, although both the strategies which involves rebalancings every fourth hour and rebalancings daily, perform very similarly. The annual strategy, which implies no rebalancings during one year (only allocation of the wealth according to u^* at the start of the year) has the worst performance. The plot of figure 5.8 shows the different rebalancing strategies versus Sharpe ratio. Also included are 95% confidence intervals, which show that the first four rebalancing strategies are not significantly different from the ex ante Sharpe ratio and that the strategies of semiannual and annual rebalancing strategies perform relatively much worse than the other strategies.

Now, why do the strategies that involve frequent rebalancings of the portfolio perform better according to the Sharpe ratio? Analysing the table we observe that the differences between the mean terminal log returns are small, which imply that the excess returns also are. Even the sample means of the estimated volatilities that the different strategies yield are very similar. The column named "Vol. of. vol", an abbreviation for the volatility of the volatilities, shows the sample standard deviations of the volatilities of each simulated portfolio for each strategy. Figure 5.9 shows the distributions of volatilities of the log returns of the two strategies that are furthest apart from each other, that is the hourly rebalancings-strategy (shaded) and the annual rebalancings-strategy. The volatilities of the

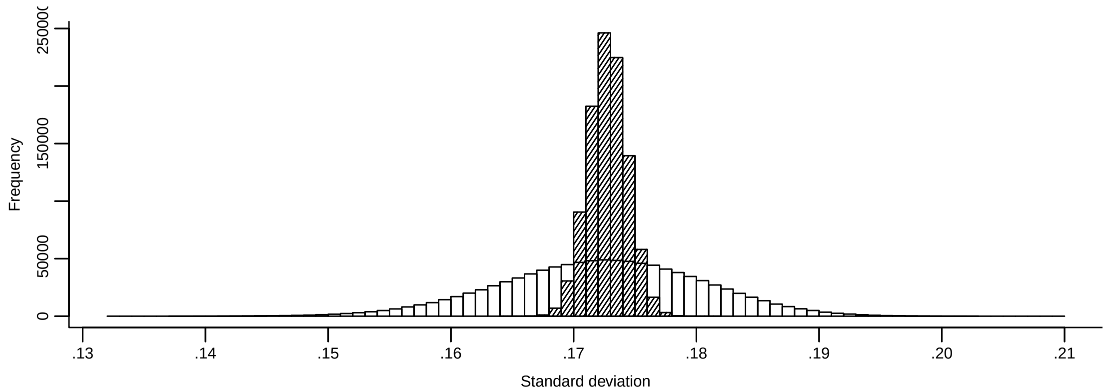


Figure 5.9: The distributions of the annualized sample standard deviations of the log returns of the "Hourly"-strategy (shaded) and the "Annually"-strategy.

annual-strategy are much more spread out compared to the volatilities of the hourly-strategy. We can conclude that the spread of the volatilities of the rebalancing strategies is negatively correlated with rebalancing frequency. This is not surprising considering that the purpose of the optimal rebalancing strategy is to limit risk. If the return on the risky asset is less than the return on the risk-free asset during the time interval between rebalancing time points, the investor puts the portfolio in a state of balance by reducing the amount invested in the risk-free asset and increasing the amount invested in the risky asset. If the risky asset performs worse than the risk-free asset over a time period, the investor can, by using this strategy, take advantage of a positive rebound of the risky asset. There is a chance however, that the value of the risky asset could decrease regularly over a long period of time. If this would be the case, the rebalancing strategy could actually increase the loss of wealth and utility, since the strategy implies that more and more wealth is reallocated into the risky asset. But the results of the simulations tell us that this is in fact not the case. By using the rebalancing strategy, the investor reduces the downside risk of the portfolio. In an opposite situation, where the risky asset performs better than the risk-free asset, the investor reduces the risky asset investment and increases the risk-free asset investment. This way, the investor is better off if the value of the risky asset goes down, compared to an investor who doesn't rebalance. If, however, the value of the risky asset increases strongly over a long period of time, frequent rebalancings of the portfolio will reduce the potential gain of wealth, compared to infrequent rebalancings or a non-rebalancing strategy. This explains the negative skew we observe in figure 5.5 of the distributions of losses of utility of the rebalancing strategies that involve infrequent rebalancings. We see that on rare occations, when the development of the risky asset is extremely strong, the rebalancing strategies that involve infrequent rebalancings beat the theoretical strategy of continuous rebalancings by a clear margin. We can conclude that the strategy of holding a constant fraction of the wealth in the risky asset reduces the potential upside gain as well as reduces the downside risk of the portfolio. But table 5.3 as well as 5.9 also tell us that the range of estimated volatilities of the annual-rebalancing strategy is much wider than the range of estimated volatilities of the hourly-rebalancing strategy. Some of the estimated volatilities of the annual-rebalancing strategy are indeed much lower. This points to the fact that even though the strategy of holding a constant fraction in the risky asset is the optimal strategy for a risk-averse investor, it does not mean that the investor wants to reduce risk at all costs. By reallocating wealth into the risky asset when the risky asset, over a time period, has performed worse than the risk-free asset, the investor is in fact increasing, in relative terms, the risk of the portfolio. The risk is increased in relative terms, because the risk or the potential change of the portfolio value, which in our model are governed by the risky asset drift, the risk-free rent and a Brownian motion, is scaled by the portfolio value itself. The fact that the optimal strategy implies both a relative increase in risk when the risky asset performs worse than risk-free asset and vice versa, makes the optimal strategy a risk-preserving strategy. One might say that, the goal of an investor using the optimal strategy, is to keep the level of risk as high as possible but at the same time below a certain threshold.

But why does this risk-preserving strategy give better Sharpe ratios? The numbers of the column named "Corr." in table 5.3 are measures of the correlations between the estimated log returns and volatilities of all the simulation runs within each rebalancing strategy. From these numbers it becomes clear that there is a negative correlation between rebalancing frequency and the correlation between log returns and volatilities. For the four strategies with the highest rebalancing frequencies, the correlations are close to zero. For the annual-rebalancing strategy the correlation is over 80%. As mentioned above, the risk or the potential change factor of the portfolio is scaled by the portfolio value itself. Higher portfolio values are associated with higher risk. By rebalancing the portfolio frequently, the association between risk and portfolio value is reduced. If the rebalancing frequency is high enough, this association is nearly completely zeroed out. For the rebalancing strategies that involve infrequent rebalancings, the correlation is stronger. This means, that for such rebalancing strategies, high log returns are associated with high volatilities. Remember that the Sharpe ratio is calculated as the ratio between the terminal excess return and the volatility of a portfolio value time series. If $a, b, c > 0$ and $a > b$, then we have that $\frac{c}{a} < \frac{c}{b}$. This just means, when calculating the Sharpe ratio, that high excess returns are more likely to be "penalized" by a high estimated volatility, if the correlation between log returns and volatilities are high.

5.3 Simulation with transaction costs

5.3.1 Introduction

In the portfolio simulations so far we have assumed transaction costs equal to zero, which is a rather unrealistic assumption. To remedy this and to add more realism into the simulations, we will in this section take transaction costs into consideration. Transaction costs can be modelled in various ways, but to keep matters simple we will assume proportional transaction costs. Proportional transaction costs mean that the transactions costs are proportional to the values or the sizes of the asset transactions by a constant factor. There are written several articles addressing Merton's portfolio problem with transaction costs. In 1990, Davis and Norman [4] studied and solved the special case of proportional transaction costs. Their solution means that the incorporation of proportional transaction costs into Merton's portfolio problem changes the optimal asset allocation strategy, which entails that the Merton ratio (3.10) no longer is the optimal strategy. However, the focus of this thesis is to study simulations of portfolios using the optimal strategy found in the original problem as it was formulated by Merton.

In what way should the payments of the transaction costs be implemented? As mentioned earlier, we want the portfolio simulations to be as realistic as possible, and seen from a realistic point of view it is natural to perceive the risk free asset as a bank account. All payments of transaction costs will therefore be deducted from the bank account. The transaction costs can be paid in mainly two ways: one is to make the payment after the portfolio has been rebalanced. The other way is to require the portfolio to be rebalanced after the payment of the transaction cost has been carried out. Among the two methods, the first method is the crude and straightforward way and is probably the method that a real life portfolio manager would use. The second method is a little bit more sophisticated and maybe less realistic. However, it can be argued that the second method reflects the idea of a constant rebalancing strategy more correctly. Thus, both methods are interesting in the context of this thesis and both methods will therefore be implemented.

5.3.2 Simulation model II

Assume now that transaction costs are paid after the portfolio has been rebalanced (we will hereafter refer to this method as subsequent transaction costs). For the portfolio to be rebalanced in this setting, we need to recalculate the transaction size Q_k. Let the transaction cost proportionality constant be denoted by λ. Remember that the set of rebalancing time points is given by $\mathcal{T}^{\text{reb}} = \{t_0, t_\epsilon, t_{2\epsilon}, \ldots, t_n\}$, and that the last rebalancing time point relative to the current time point in which we want to simulate the portfolio, is given by t_{k^*}. Assume that t_k is a rebalancing time point and let the value of the transaction at time t_k be denoted by Q_k. The proportionality of the transaction cost simply means that the transaction cost is equal to $\lambda|Q_k|$. The value of the transaction itself is the same as in the setting with no transaction costs (5.4). Compared to simulation model I of section 5.2, the inclusion of subsequent transaction costs gives the following slightly modified simulation scheme:

Simulation model II
Transaction costs: subsequent
Volatility: constant

$$\begin{aligned}
\tilde{V}_k'^S &= u^*\tilde{V}_{k^*}\prod_{j=k^*}^{k-1}(1+\mu\delta+\sigma\Delta B_j) \\
\tilde{V}_k'^R &= (1-u^*)\tilde{V}_{k^*}(1+r\delta)^{k-k^*} \\
Q_k &= (1-u^*)\tilde{V}_k'^S - u^*\tilde{V}_k'^R \\
\tilde{V}_k^S &= \begin{cases} \tilde{V}_k'^S - Q_k, & t_k \in \mathcal{T}^{\text{reb}} \\ \tilde{V}_k'^S, & \text{otherwise} \end{cases} \\
\tilde{V}_k^R &= \begin{cases} \tilde{V}_k'^R + Q_k - \lambda|Q_k|, & t_k \in \mathcal{T}^{\text{reb}} \\ \tilde{V}_k'^R, & \text{otherwise} \end{cases} \\
\tilde{V}_k &= \tilde{V}_k^S + \tilde{V}_k^R.
\end{aligned} \tag{5.11}$$

Similarly to the simulation model I of section 5.2, this simulation scheme can also be restated in a more compact way,

$$\tilde{V}_k = \begin{cases} \begin{cases} u^*\tilde{V}_{k^*}\prod_{j=k^*}^{k-1}(1+\mu\delta+\sigma\Delta B_j) + (1-u^*)\tilde{V}_{k^*}(1+r\delta)^{k-k^*} \\ -\lambda u^*(1-u^*)\tilde{V}_{k^*}\left|\prod_{j=k^*}^{k-1}(1+\mu\delta+\sigma\Delta B_j) - (1+r\delta)^{k-k^*}\right| \end{cases}, & t_k \in \mathcal{T}^{\text{reb}} \\ u^*\tilde{V}_{k^*}\prod_{j=k^*}^{k-1}(1+\mu\delta+\sigma\Delta B_j) + (1-u^*)\tilde{V}_{k^*}(1+r\delta)^{k-k^*}, & \text{otherwise.} \end{cases}$$

5.3.3 Simulation model III

Assume instead that transaction costs are paid before the portfolio is rebalanced (we will hereafter refer to this method as preceding transaction costs). Let the difference between the return on the risky asset and the return on the risk-free asset since the last rebalancing time point t_{k^*} at time t_k be denoted by D_k, that is

$$D_k = \left(\prod_{j=k^*}^{k-1} (1 + \mu\delta + \sigma\Delta B_j) - 1 \right) - \left((1 + r\delta)^{k-k^*} - 1 \right). \tag{5.12}$$

As we have seen earlier, it is clear that the direction of the transaction between the risky and the risk free asset investment to rebalance the portfolio, only depends on the sign of the difference in returns on the investments, that is the sign of D_k. Still assume that t_k is a rebalancing time point. Remember that $\tilde{V}_k'^S$ and $\tilde{V}_k'^R$ are the values of the risky asset and risk free asset investment, respectively, before the portfolio is rebalanced. For the portfolio to be rebalanced after transaction costs have been paid, the following relations have to be fulfilled:

$$u^* = \frac{\tilde{V}_k'^S - Q_k}{\tilde{V}_k},$$

$$1 - u^* = \begin{cases} \dfrac{\tilde{V}_k'^R + Q_k - \lambda Q_k}{\tilde{V}_k}, & D_k \geq 0 \\ \dfrac{\tilde{V}_k'^R + Q_k + \lambda Q_k}{\tilde{V}_k}, & D_k < 0 \end{cases}.$$

Solving these equations with respect to $\tilde{V}_k$ and then putting the solutions together yields,

$$\frac{\tilde{V}_k'^S - Q_k}{u^*} = \begin{cases} \dfrac{\tilde{V}_k'^R + Q_k - \lambda Q_k}{1 - u^*}, & D_k \geq 0 \\ \dfrac{\tilde{V}_k'^R + Q_k + \lambda Q_k}{1 - u^*}, & D_k < 0 \end{cases}.$$

Finally, solving this equation with respect to Q_k gives the following solution:

$$Q_k = \begin{cases} \dfrac{(1 - u^*)\tilde{V}_k'^S - u^*\tilde{V}_k'^R}{1 - \lambda u^*}, & D_k \geq 0 \\ \dfrac{(1 - u^*)\tilde{V}_k'^S - u^*\tilde{V}_k'^R}{1 + \lambda u^*}, & D_k < 0 \end{cases}.$$

The solution is almost equal to the solution (5.4) of section 5.2 except for the additional expressions in the denominators. We notice that if $Q_k \geq 0$, then $1 - \lambda u^* \leq 1$, which reflects the fact that the portfolio manager has to take into account the deduction of the transaction cost from the bank account before the portfolio is rebalanced. As a consequence she has to make a bigger transfer from the risky asset investment to ensure that the portfolio becomes rebalanced, compared to the setting with no transaction costs or subsequent transactions costs. If $Q_k < 0$ she needs to transfer less than before, since the deduction of the transaction cost itself contributes towards a rebalanced portfolio. The inclusion of preceding transaction costs gives the following, slightly modified, simulation scheme:

Simulation model III
Transaction costs: preceding
Volatility: constant

$$
\begin{aligned}
\tilde{V}_k'^S &= u^* \tilde{V}_{k^*} \prod_{j=k^*}^{k-1} (1 + \mu\delta + \sigma \Delta B_j) \\
\tilde{V}_k'^R &= (1 - u^*) \tilde{V}_{k^*} (1 + r\delta)^{k-k^*} \\
D_k &= \left(\prod_{j=k^*}^{k-1} (1 + \mu\delta + \sigma \Delta B_j) - 1 \right) - \big((1 + r\delta)^{k-k^*} - 1 \big) \\
Q_k &= \begin{cases} \dfrac{(1 - u^*)\tilde{V}_k'^S - u^* \tilde{V}_k'^R}{1 - \lambda u^*}, & D_k \geq 0 \\ \dfrac{(1 - u^*)\tilde{V}_k'^S - u^* \tilde{V}_k'^R}{1 + \lambda u^*}, & D_k < 0 \end{cases} \\
\tilde{V}_k^S &= \begin{cases} \tilde{V}_k'^S - Q_k, & t_k \in \mathcal{T}^{\text{reb}} \\ \tilde{V}_k'^S, & \text{otherwise} \end{cases} \\
\tilde{V}_k^R &= \begin{cases} \tilde{V}_k'^R + Q_k - \lambda |Q_k|, & t_k \in \mathcal{T}^{\text{reb}} \\ \tilde{V}_k'^R, & \text{otherwise} \end{cases} \\
\tilde{V}_k &= \tilde{V}_k^S + \tilde{V}_k^R.
\end{aligned}
\tag{5.13}
$$

A shorter representation of this simulation scheme is stated as follows,

$$\tilde{V}_k = \begin{cases} \begin{cases} u^*\tilde{V}_{k^*} \prod_{j=k^*}^{k-1} (1+\mu\delta+\sigma\Delta B_j) + (1-u^*)\tilde{V}_{k^*}(1+r\delta)^{k-k^*} \\ -\dfrac{\lambda u^*(1-u^*)\tilde{V}_{k^*} \left| \prod_{j=k^*}^{k-1}(1+\mu\delta+\sigma\Delta B_j) - (1+r\delta)^{k-k^*} \right|}{1-\lambda u^* \mathrm{sgn}\left(\prod_{j=k^*}^{k-1}(1+\mu\delta+\sigma\Delta B_j) - (1+r\delta)^{k-k^*} \right)} \end{cases}, & t_k \in \mathcal{T}^{\mathrm{reb}} \\ u^*\tilde{V}_{k^*} \prod_{j=k^*}^{k-1} (1+\mu\delta+\sigma\Delta B_j) + (1-u^*)\tilde{V}_{k^*}(1+r\delta)^{k-k^*}, & \text{otherwise} \end{cases}$$

where the function $\mathrm{sgn}(x)$ is defined as

$$\mathrm{sgn}(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases}.$$

The question now is, how will the two slightly different simulation schemes perform and how will they compare against each other? Which strategy is the most profitable? The only difference between the strategies are the transaction costs $\{\lambda|Q_k|\}_{k\in\mathcal{T}^{\mathrm{reb}}}$. Assume that $\lambda|Q_k^{\mathrm{pre}}|$ and $\lambda|Q_k^{\mathrm{sub}}|$ denote the transaction costs of the simulation scheme with preceding transaction costs and with subsequent transaction costs, respectively. We have that

$$\begin{aligned} \lambda|Q_k^{\mathrm{pre}}| - \lambda|Q_k^{\mathrm{sub}}| &= \begin{cases} \lambda Q_k^{\mathrm{pre}} - \lambda Q_k^{\mathrm{sub}}, & D_k \geq 0 \\ \lambda Q_k^{\mathrm{sub}} - \lambda Q_k^{\mathrm{pre}} = -(\lambda Q_k^{\mathrm{pre}} - \lambda Q_k^{\mathrm{sub}}), & D_k < 0 \end{cases} \\ &= \begin{cases} \dfrac{\lambda\left((1-u^*)\tilde{V}_k'^S - u^*\tilde{V}_k'^R\right)}{1-\lambda u^*} - \lambda\left((1-u^*)\tilde{V}_k'^S - u^*\tilde{V}_k'^R\right), & D_k \geq 0 \\ -\left(\dfrac{\lambda\left((1-u^*)\tilde{V}_k'^S - u^*\tilde{V}_k'^R\right)}{1+\lambda u^*} - \lambda\left((1-u^*)\tilde{V}_k'^S - u^*\tilde{V}_k'^R\right) \right), & D_k < 0 \end{cases} \\ &= \begin{cases} \dfrac{\lambda^2 u^*}{1-\lambda u^*}\left((1-u^*)\tilde{V}_k'^S - u^*\tilde{V}_k'^R\right), & D_k \geq 0 \\ \dfrac{\lambda^2 u^*}{1+\lambda u^*}\left((1-u^*)\tilde{V}_k'^S - u^*\tilde{V}_k'^R\right), & D_k < 0 \end{cases} \\ &= \begin{cases} \dfrac{\lambda^2 u^{*2}(1-u^*)}{1-\lambda u^*}\tilde{V}_{k^*}D_k, & D_k \geq 0 \\ \dfrac{\lambda^2 u^{*2}(1-u^*)}{1+\lambda u^*}\tilde{V}_{k^*}D_k, & D_k < 0 \end{cases} \end{aligned} \tag{5.14}$$

Given a portfolio value $\tilde{V}_{k^*}$ at the previous rebalancing time point t_{k^*}, we see that the difference in transaction cost at time t_k is simply a function of the difference in returns on the risky and the risk free asset investments times $\tilde{V}_{k^*}$ times a constant. We also see that the difference between preceding and subsequent transaction cost depends on the direction of the transaction, which in turn depends on the difference in return on the risky asset and the risk-free asset, which is given by D_k. $D_k > 0$ favours the subsequent transaction cost strategy, whereas $D_k < 0$ favours the preceding transaction cost strategy. The plots of figure 5.10 shows

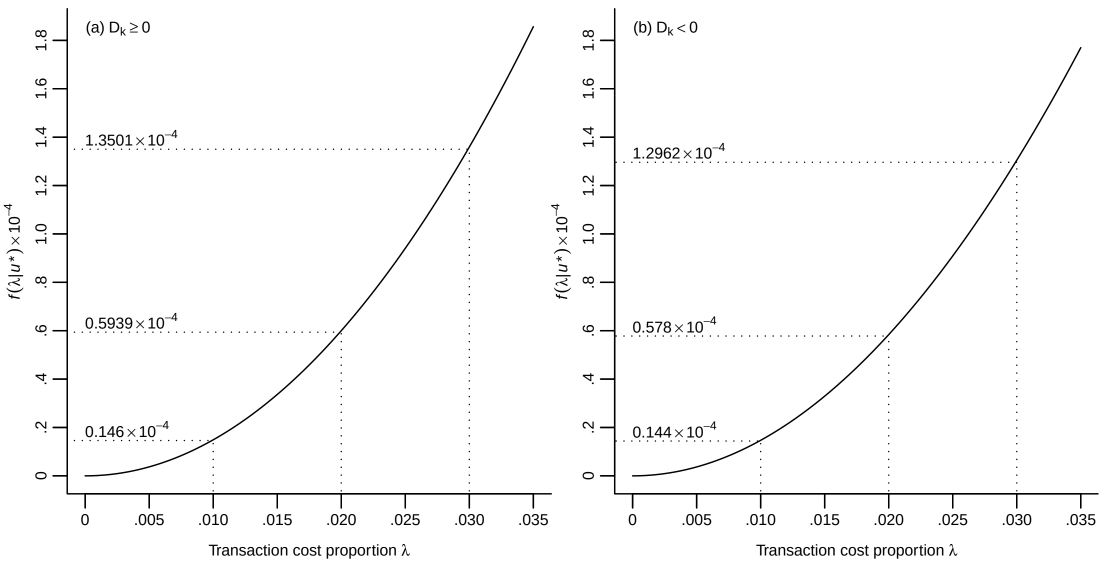


Figure 5.10: (a) $f(\lambda|u^*) = (\lambda^2 u^{*2}(1-u^*))/(1-\lambda u^*)$, (b) $f(\lambda|u^*) = (\lambda^2 u^{*2}(1-u^*))/(1+\lambda u^*)$

how the constants $(\lambda^2 u^{*2}(1-u^*))/(1-\lambda u^*)$ and $(\lambda^2 u^{*2}(1-u^*))/(1+\lambda u^*)$ increases exponentially as a function of the proportionality constant λ.

What values of λ are reasonable seen from a realistic point of view? That could depend on various factors such as the size of the transaction, the size and power of the company involved in the transaction, the relation between the company and the broker and probably many other factors. According to the thesis supervisor $.02 - .03$ could be reasonable values for a small player in the market. A large enough player could perhaps achieve less than .01. To be on the safe side we will consider different values, .01, .02 and .03, for λ in the calculations of the transaction costs. In figure 5.10, these particular values on the horizontal axis and the corresponding values as a function of λ on the vertical axis are indicated by the dotted lines. The exponential relationship means for instance that a tripling of the transaction cost proportion from $\lambda = \lambda_1 = .01$ to $\lambda = \lambda_3 = .03$, will imply

$$
\begin{aligned}
&\begin{cases}
\dfrac{\lambda_3^2 u^{*2}(1-u^*)}{1-\lambda_3 u^*} \Big/ \dfrac{\lambda_1^2 u^{*2}(1-u^*)}{1-\lambda_1 u^*} = \dfrac{(3\lambda_1)^2 u^{*2}(1-u^*)}{1-3\lambda_1 u^*} \Big/ \dfrac{\lambda_1^2 u^{*2}(1-u^*)}{1-\lambda_1 u^*} \\
\dfrac{\lambda_3^2 u^{*2}(1-u^*)}{1+\lambda_3 u^*} \Big/ \dfrac{\lambda_1^2 u^{*2}(1-u^*)}{1+\lambda_1 u^*} = \dfrac{(3\lambda_1)^2 u^{*2}(1-u^*)}{1+3\lambda_1 u^*} \Big/ \dfrac{\lambda_1^2 u^{*2}(1-u^*)}{1+\lambda_1 u^*}
\end{cases} \\
&= \begin{cases}
\dfrac{9(1-\lambda_1 u^*)}{1-3\lambda_1 u^*} = 9.1251 \\
\dfrac{9(1+\lambda_1 u^*)}{1+3\lambda_1 u^*} = 8.8799
\end{cases}
\end{aligned}
\tag{5.15}
$$

approximately, a nine-time increase in the transaction cost difference between the two transaction cost strategies, assuming equal values for $\tilde{V}_{k^*}$ and D_k. In reality this difference will be slightly lower considering that the returns on the portfolio will be reduced due to the increased transaction costs. The equations (5.14) also tell us that the strategy of subsequent transaction costs is slightly better if the return on the risky asset investment is greater than the return on the risk free asset investment since the previous rebalancing time point k^*. If the return on the risk free asset is greater, then the strategy of preceding transaction costs is better. This might suggest that we ought to choose the strategy of subsequent transaction costs if we expect the risky asset to beat the risk free asset in the market, and vice versa. In the next section we will through two simulation test runs investigate this further.

5.3.4 Simulation test runs

How does the incorporation of transaction costs into the portfolio model affect the development of the value and the utility of the portfolio over time? In this section we will try to answer this question through the analysis of a complete (one year) time series of simulated portfolio values, including both preceding and subsequent transaction costs. As in section 5.2, the simulation algorithm first simulates a Brownian motion time series. The Brownian motion is updated hourly over one year, that is 252 trading days, which result in a Brownian motion time series consisting of 6048 points. The same Brownian motion time series is then used to calculate a theoretical portfolio time series and to simulate a portfolio without transaction costs that will serve as an alternative baseline for comparison, a portfolio with preceding transaction costs (5.13) and a portfolio with subsequent transaction costs (5.11). By using the same Brownian motion in the calculation of the theoretical portfolio and in the, up till now, three different portfolio simulation schemes, it will be easier to make comparisons.

The plots of figure 5.11 continues on the discussion of preceding versus subsequent transaction costs of the previous subsection. The plot of subfigure (a) shows a

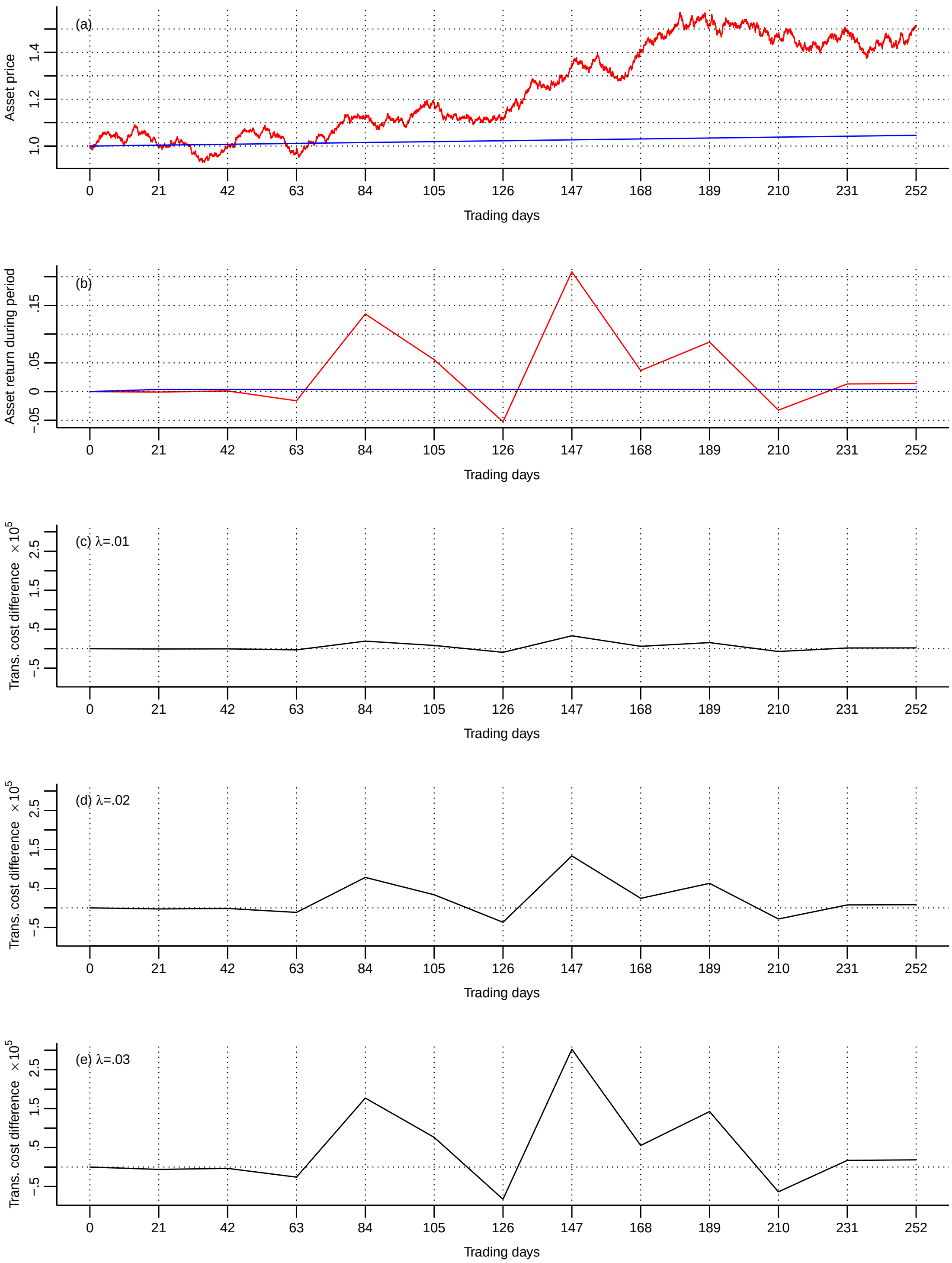


Figure 5.11: (a) asset prices, (b) asset returns, (c) transaction cost differences $\lambda = .01$, (d) $\lambda = .02$ and (e) $\lambda = .03$.

random development of the two possible investment objects, namely the risky asset (red) and the risk free asset (blue). In this particular simulation run the risky asset beats the risk free asset by a clear margin. This is reflected in the plot of subfigure (b) which shows the returns on each asset during the time periods between the rebalancing time points. The plots of subfigures (c), (d) and (e) show the transaction cost differences $\{\lambda|Q_k^{\text{pre}}| - \lambda|Q_k^{\text{sub}}|\}_{k\in\mathcal{T}^{\text{pre}}}$ at each rebalancing time point for different values of the transaction cost proportionality constant λ. We observe that large changes in the risky asset value require large (in relative terms) transactions to rebalance the portfolio. These plots also confirm that the transaction cost differences are a simple function of the risky and the risk-free asset returns and that the differences increase exponentially as a function of λ.

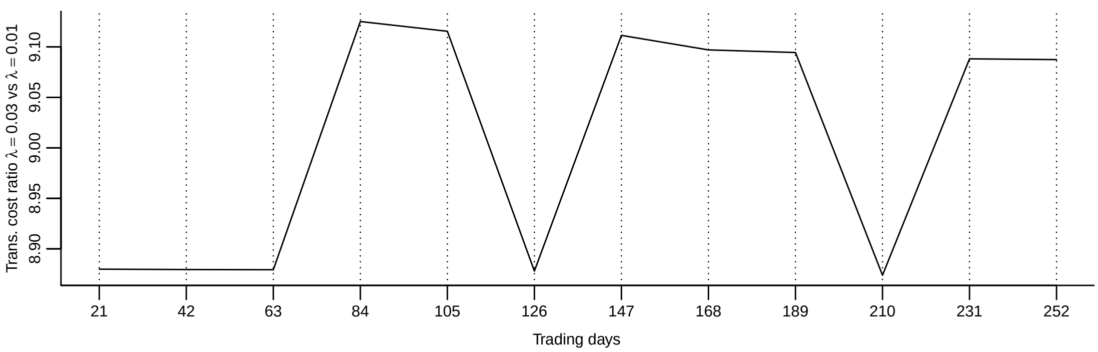


Figure 5.12: Transaction cost difference ratio, $\lambda = .03$ versus $\lambda = .01$.

The plot of figure 5.12 displays transaction costs difference ratios at each rebalancing time point, comparing $\lambda = .03$ versus $\lambda = .01$ using the same Brownian motion. They are in agreement with the previous calculations (5.15). Table 5.4

	Simulation model		Terminal wealth	Loss of wealth	Terminal utility	Loss of utility	Total transaction costs
	Theoretical		1.3537	0	1.1725	0	0
Transaction costs	None		1.3535	$.1758\times10^{-3}$	1.1724	$.0800\times10^{-3}$	0
	Preceding	$\lambda = .01$	1.3517	1.9476×10^{-3}	1.1716	$.8868\times10^{-3}$	1.5722×10^{-3}
		$\lambda = .02$	1.3500	3.7323×10^{-3}	1.1708	1.6999×10^{-3}	3.1561×10^{-3}
		$\lambda = .03$	1.3482	5.5303×10^{-3}	1.1700	2.5196×10^{-3}	4.7521×10^{-3}
	Subsequent	$\lambda = .01$	1.3517	1.9404×10^{-3}	1.1716	$.8835\times10^{-3}$	1.5656×10^{-3}
		$\lambda = .02$	1.3500	3.7032×10^{-3}	1.1708	1.6867×10^{-3}	3.1294×10^{-3}
		$\lambda = .03$	1.3482	5.4641×10^{-3}	1.1700	2.4894×10^{-3}	4.6914×10^{-3}

Table 5.4: Summary of the first simulation run with transaction costs incorporated.

summarizes the first simulation run with transaction costs incorporated. It is of course clear that the incorporation of transaction costs into the portfolio simulation model entail a loss of both wealth and utility, compared to the theoretical model or the model with no transaction costs. The losses are however quite small. Even for the models with the highest transaction costs ($\lambda = .3$), the loss of wealth is only about .4% and the loss of utility is only about .2%. Of course, for a portfolio of great value this loss could still be quite significant. As for the transaction cost totals, they make up about 80-90% of the total losses of wealth for their respective portfolio models. The remainder of the total losses is made up of the lower returns on the portfolio. Which transaction cost strategy gives the smallest loss? The differences between the strategies are very small, but the losses of the portfolios that use the strategy of subsequent transaction costs are slightly smaller, which is as expected based on the conclusions of the previous section. For the sake of comparison we will include a second simulation run.

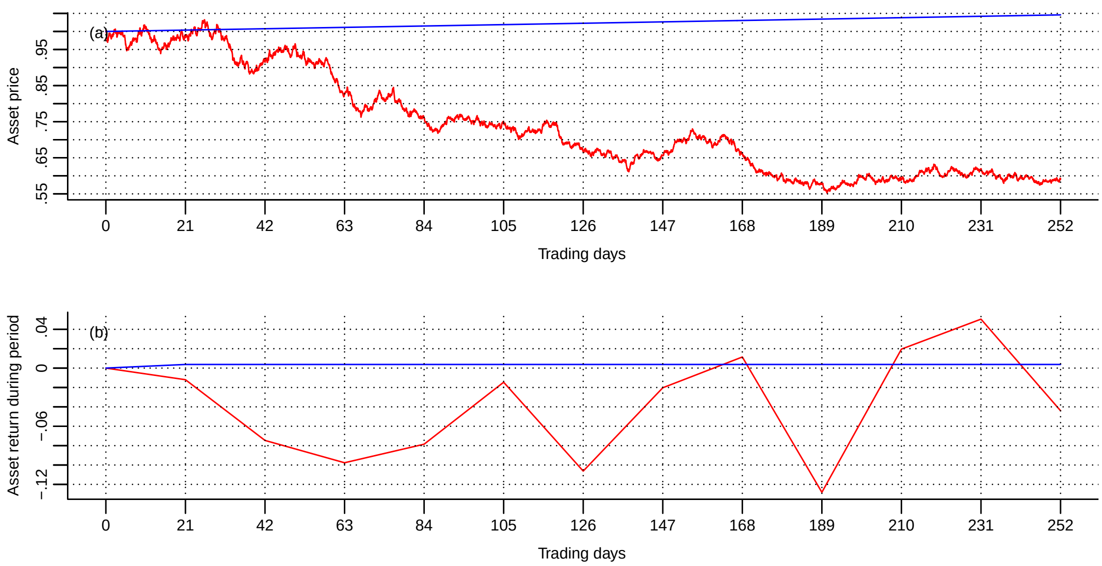


Figure 5.13: (a) asset prices, (b) asset returns

The plots of figure 5.13 show the price developments (a) and the returns during the time periods between rebalancing time points (b) of the risky asset (red) and the risk free asset (blue) of the second simulation run. This time the performance of the risky asset is quite bad: the risky asset price decrease nearly 50% over the simulated time period of one year. By observing the quantities of table 5.5 we can conclude that for the second simulation run, the strategy of preceding transaction costs gives slightly smaller losses of wealth and utility, compared to the strategy of subsequent transaction costs. This is the opposite conclusion of the first simulation run and in accordance with the conclusions of the previous section. If we take a look at the total transaction costs we see that they actually are larger than the total losses of wealth for their respective strategies and parameter configurations.

Simulation scheme			Terminal wealth	Loss of wealth	Terminal utility	Loss of utility	Total transaction costs
Theoretical			0.7153	0	0.8386	0	0
Transaction costs	None		0.7151	0.2083×10^{-3}	0.8384	0.1283×10^{-3}	0
	Preceding	$\lambda = .01$	0.7140	1.3149×10^{-3}	0.8378	0.8104×10^{-3}	1.2377×10^{-3}
		$\lambda = .02$	0.7129	2.4081×10^{-3}	0.8371	1.4847×10^{-3}	2.4602×10^{-3}
		$\lambda = .03$	0.7118	3.4883×10^{-3}	0.8364	2.1514×10^{-3}	3.6680×10^{-3}
	Subsequent	$\lambda = .01$	0.7140	1.3210×10^{-3}	0.8378	0.8141×10^{-3}	1.2446×10^{-3}
		$\lambda = .02$	0.7129	2.4322×10^{-3}	0.8371	1.4995×10^{-3}	2.4877×10^{-3}
		$\lambda = .03$	0.7118	3.5419×10^{-3}	0.8364	2.1845×10^{-3}	3.7292×10^{-3}

Table 5.5: Summary of the second simulation run with transaction costs incorporated.

Let $\tilde{V}_k$ denote the simulated portfolio value at time t_k without transaction costs and let $\tilde{V}_k^{\text{tc}}$ denote the simulated portfolio value at time t_k with transaction costs. Assume that $\tilde{V}_0 = \tilde{V}_0^{\text{tc}}$. Remember that the set of rebalancing time points $\mathcal{T}^{\text{reb}}$ can be defined by the distance ϵ between the rebalancing time points indices, that is $\mathcal{T}^{\text{reb}} = \{t_\epsilon, t_{2\epsilon}, \ldots, t_n\}$ ($\frac{n}{\epsilon} \in \mathcal{N}$). Hence, the first rebalancing time point is t_ϵ. At the first rebalancing time point, we have that

$$\begin{aligned}
\tilde{V}_\epsilon - \tilde{V}_\epsilon^{\text{tc}} &= u^*\tilde{V}_0 \prod_{j=0}^{\epsilon-1}(1+\mu\delta+\sigma\Delta B_j) + (1-u^*)\tilde{V}_0(1+r\delta)^\epsilon \\
&\quad - \left(u^*\tilde{V}_0^{\text{tc}} \prod_{j=0}^{\epsilon-1}(1+\mu\delta+\sigma\Delta B_j) + (1-u^*)\tilde{V}_0^{\text{tc}}(1+r\delta)^\epsilon - \lambda|Q_\epsilon|\right) \\
&= \lambda|Q_\epsilon|.
\end{aligned}$$

At the second rebalancing time point, we have that

$$\begin{aligned}
\tilde{V}_{2\epsilon} - \tilde{V}_{2\epsilon}^{\text{tc}} &= u^*\tilde{V}_\epsilon \prod_{j=\epsilon}^{2\epsilon-1}(1+\mu\delta+\sigma\Delta B_j) + (1-u^*)\tilde{V}_\epsilon(1+r\delta)^\epsilon \\
&\quad - \left(u^*\tilde{V}_\epsilon^{\text{tc}} \prod_{j=\epsilon}^{2\epsilon-1}(1+\mu\delta+\sigma\Delta B_j) + (1-u^*)\tilde{V}_\epsilon^{\text{tc}}(1+r\delta)^\epsilon - \lambda|Q_{2\epsilon}|\right) \\
&= (\tilde{V}_\epsilon - \tilde{V}_\epsilon^{\text{tc}})\left(u^* \prod_{j=\epsilon}^{2\epsilon-1}(1+\mu\delta+\sigma\Delta B_j) + (1-u^*)(1+r\delta)^\epsilon\right) + \lambda|Q_{2\epsilon}| \\
&= \lambda|Q_\epsilon|\left(u^* \prod_{j=\epsilon}^{2\epsilon-1}(1+\mu\delta+\sigma\Delta B_j) + (1-u^*)(1+r\delta)^\epsilon\right) + \lambda|Q_{2\epsilon}|,
\end{aligned}$$

and so on. This means that the difference between a simulated portfolio value without transaction costs incorporated and one with, is not simply the sum of the transaction costs at each rebalancing time point (except at the first rebalancing time point). If the expression in the parentheses is less that one, the loss of wealth will be smaller than the sum of the transaction costs, and vice versa. This is also what we basically observed in the comparisons of the two simulation runs.

5.3.5 Mean loss of utility

In this subsection we will look at the results of 100,000 simulation runs[1] of each combination of the input parameter triplet, rebalancing strategy, transaction cost strategy and transaction cost proportion. Regarding the rebalancing strategies, we will include the same strategies as we have done so far in our simulations. As for the transaction cost strategies, we will include both preceding and subsequent transaction costs. We will also include transaction costs proportions equal to .01, .02 and .03 in our simulations as well as no transaction costs for the sake of comparison. More specifically, for each pairwise combination of transaction cost proportion and rebalancing strategy we will do 100,000 simulation runs. For each simulation run, we will generate one Brownian motion consisting of 6048 points that will be used to simulate a time series of theoretical portfolio values, a time series of simulated portfolio values with no transaction costs, a time series of simulated portfolio values using preceding transaction costs and a time series of simulated portfolio values using subsequent transaction costs. This will result in a total of 10.8 million portfolio value time series, from a total of 2.7 million Brownian motions, each consisting of 6048 time points, giving the possibility to calculate 36 mean losses of utility.

The first batch of simulation runs assumes transaction cost proportion $\lambda = .01$. The plots of figure 5.14 show the mean losses of utility versus number of simulations for a selection of four rebalancing strategies, namely hourly, daily, monthly and annually. The choice of transaction cost method is also included in the plots. According to the plots, the mean losses of utility seem to converge toward a constant value, similar to what we saw with simulation model I in section 5.2.

Table 5.6 summarizes the results of the first batch of simulation runs with transaction cost proportion equal to .01. The abbreviations of the third column of the table need an explanation. "Th" is an abbreviation for "theoretical". The "Th"-rows show the results of the simulations of the theoretical portfolio values, calculated according to equation (5.2), that is the exact solution of the continuous

[1] In the previous section, where the focus was on simulation model I, we performed 1,000,000 simulation runs for each rebalancing strategy. Because of the added complexity of simulation model II and III, which entails both an increased number of parameter combinations that need to be simulated, and more complex, slower running simulation algorithms, we need to reduce the number of simulation runs for each parameter combination, in order to attain acceptable simulation running times.

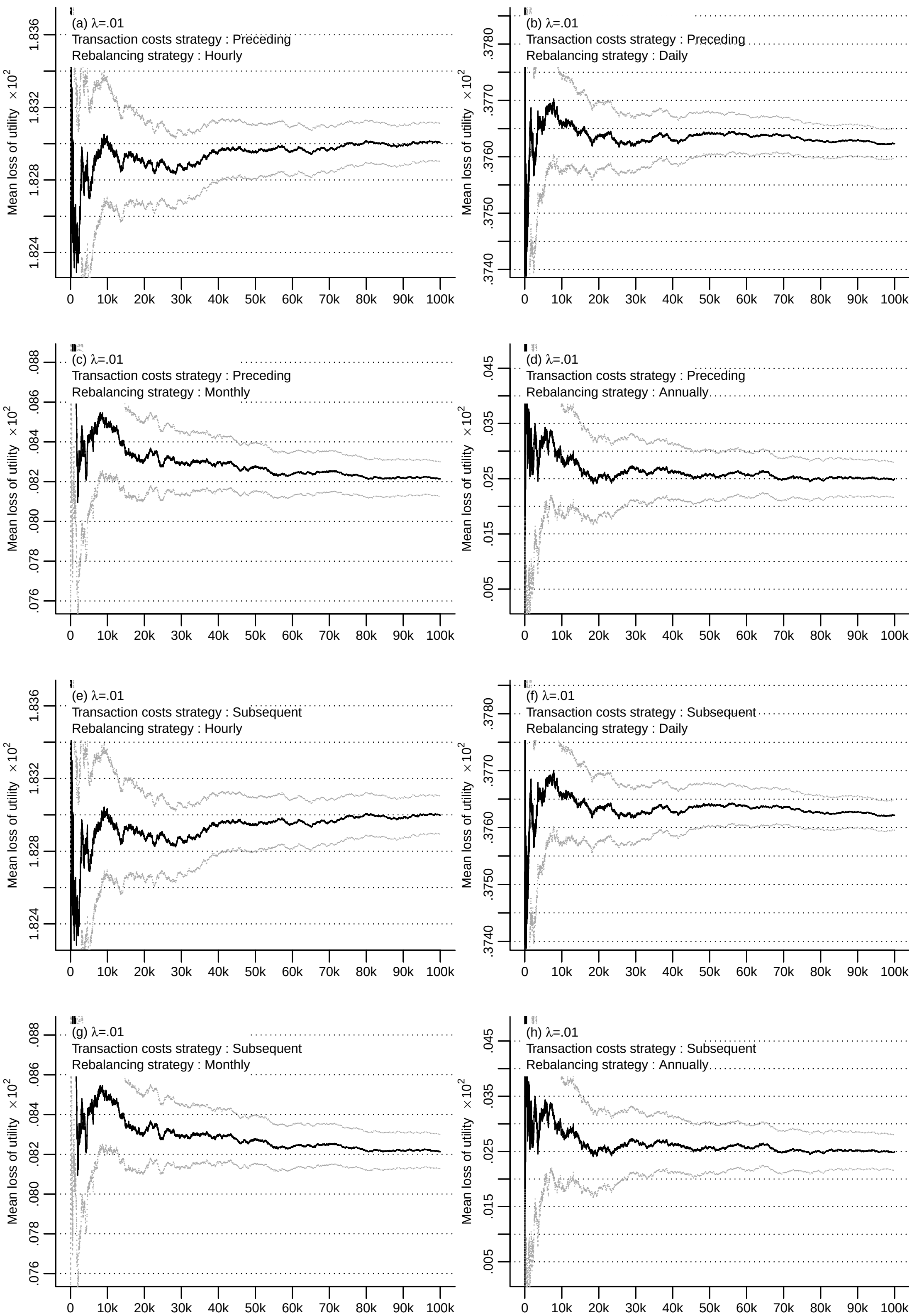


Figure 5.14: The mean losses of utility with transaction cost proportion $\lambda = .01$. (a)-(d) preceding transaction costs and (e)-(h) subsequent transaction costs.

$\lambda = .01$			Sample means				StDev
Simulation model			Term. wealth	Total cost	Term. utility	Loss of utility	Loss of utility
Rebalancing strategy: Hourly		Th	1.0607	-	1.0276	-	-
		No	1.0607	-	1.0276	$-.0001\times10^{-2}$	$.1464\times10^{-3}$
		Pre	1.0250	3.4621×10^{-2}	1.0093	1.8301×10^{-2}	1.6891×10^{-3}
		Sub	1.0250	3.4619×10^{-2}	1.0093	1.8300×10^{-2}	1.6870×10^{-3}
Every 4th hour		Th	1.0607	-	1.0276	-	-
		No	1.0607	-	1.0276	0	$.1893\times10^{-3}$
		Pre	1.0428	1.7458×10^{-2}	1.0185	$.9191\times10^{-2}$	$.8713\times10^{-3}$
		Sub	1.0428	1.7457×10^{-2}	1.0185	$.9191\times10^{-2}$	$.8693\times10^{-3}$
Daily		Th	1.0608	-	1.0277	-	-
		No	1.0608	-	1.0277	0	$.3619\times10^{-3}$
		Pre	1.0534	$.7168\times10^{-2}$	1.0239	$.3762\times10^{-2}$	$.4267\times10^{-3}$
		Sub	1.0534	$.7167\times10^{-2}$	1.0239	$.3762\times10^{-2}$	$.4251\times10^{-3}$
Every 3rd day		Th	1.0611	-	1.0278	-	-
		No	1.0611	-	1.0278	$-.0001\times10^{-2}$	$.4954\times10^{-3}$
		Pre	1.0559	$.5075\times10^{-2}$	1.0252	$.2662\times10^{-2}$	$.4315\times10^{-3}$
		Sub	1.0559	$.5075\times10^{-2}$	1.0252	$.2662\times10^{-2}$	$.4304\times10^{-3}$
Every 12th day		Th	1.0603	-	1.0274	-	-
		No	1.0603	-	1.0274	$.0006\times10^{-2}$	1.1737×10^{-3}
		Pre	1.0582	$.2074\times10^{-2}$	1.0263	$.1092\times10^{-2}$	1.0142×10^{-3}
		Sub	1.0582	$.2074\times10^{-2}$	1.0263	$.1092\times10^{-2}$	1.0140×10^{-3}
Monthly		Th	1.0617	-	1.0281	-	-
		No	1.0617	-	1.0281	$-.0002\times10^{-2}$	1.5543×10^{-3}
		Pre	1.0601	$.1574\times10^{-2}$	1.0273	$.0821\times10^{-2}$	1.3880×10^{-3}
		Sub	1.0601	$.1574\times10^{-2}$	1.0273	$.0821\times10^{-2}$	1.3880×10^{-3}
Bimonthly		Th	1.0604	-	1.0275	-	-
		No	1.0604	-	1.0275	$-.0004\times10^{-2}$	2.2016×10^{-3}
		Pre	1.0593	$.1115\times10^{-2}$	1.0269	$.0577\times10^{-2}$	2.0325×10^{-3}
		Sub	1.0593	$.1114\times10^{-2}$	1.0269	$.0577\times10^{-2}$	2.0325×10^{-3}
Seminannualy		Th	1.0610	-	1.0277	-	-
		No	1.0610	-	1.0277	0	3.7754×10^{-3}
		Pre	1.0603	$.0651\times10^{-2}$	1.0274	$.0335\times10^{-2}$	3.6069×10^{-3}
		Sub	1.0603	$.0650\times10^{-2}$	1.0274	$.0335\times10^{-2}$	3.6070×10^{-3}
Annually		Th	1.0609	-	1.0277	-	-
		No	1.0610	-	1.0277	$.0014\times10^{-2}$	5.3259×10^{-3}
		Pre	1.0605	$.0465\times10^{-2}$	1.0275	$.0250\times10^{-2}$	5.1586×10^{-3}
		Sub	1.0605	$.0464\times10^{-2}$	1.0275	$.0250\times10^{-2}$	5.1587×10^{-3}

Table 5.6: Mean losses of utility and other statistics, $\lambda = .01$.

time SDE (3.2), which implies no transaction costs and continuous rebalancing of the portfolio. This is the same baseline as was used in section 5.2. This baseline will also be used in calculations of loss of utility. Also included are discrete time portfolio simulations without transaction costs, i.e. simulations based on simulation model I of section 5.2. The results of these simulations are given in the rows named "No", which are an abbreviation for "no transaction costs". These results function as an alternative baseline. The "Pre"-rows show the results of the simulated discrete time portfolio values assuming preceding transaction costs, i.e. simulations based on simulation model III. Finally, the rows of the "Sub"-category are the results of the simulated discrete time portfolio values assuming subsequent transaction costs, i.e. simulations based on simulation model II.

So what do the summary of results of table 5.6 tell us about the mean loss of utility? Clearly, the introduction of transaction costs into the simulation model has an significant negative effect on both terminal wealth and terminal utility. The effect is most noticeable for the rebalancing strategies that involve frequent rebalancings of the portfolio. For example for the hourly-rebalancing strategy the mean loss of utility is approximately 1.8300×10^{-2} for both transaction cost methods, which is a loss of about 1.8% compared to both the mean utility of the terminal theoretical wealth and the mean utility of the discrete time terminal wealth of the portfolio with no transaction costs. We also observe that the mean loss of utility for the hourly-rebalancing strategy is about twice as large as the 'every 4th hour'-rebalancing strategy, which in turn is about 2.4 times as large as the daily-rebalancing strategy. In fact, it seems like the mean loss of utility approximately is multiplied by a factor $\sqrt{n}$, if the rebalancing frequency is multiplied by a factor n.

The increased mean losses of utility are of course a direct consequence of the added transaction costs. But why is the mean total transaction costs so much higher for the high-frequency rebalancing strategies? Well, the reason is that for a high-frequency rebalancing strategy, frequent but small transactions are needed to rebalance the portfolio frequently, whereas a less frequent rebalancing strategy would imply fewer, but potentially larger transactions. But there is always a chance for a scenario in which the returns on both the risky and the risk-free assets could be nearly equal after a long period of time. Figure 5.15 exhibit such a scenario over a duration of one year. We observe that there is a lot of variation in the risky asset price during the year, but after one year we observe that the risky asset price almost becomes equal to the risk-free asset price. This means that in this particular scenario, the total transaction cost for an investor that rebalances her portfolio at an hourly basis, would amount to a total of 9.3348×10^{-2}, assuming preceding transaction costs and $\lambda = .03$. The total transaction cost for an investor that uses an annual-rebalancing strategy, would amount to a total of 2.9800×10^{-5}, since the transaction amount needed to rebalance the portfolio after one year would be very small. Figure 5.16 shows

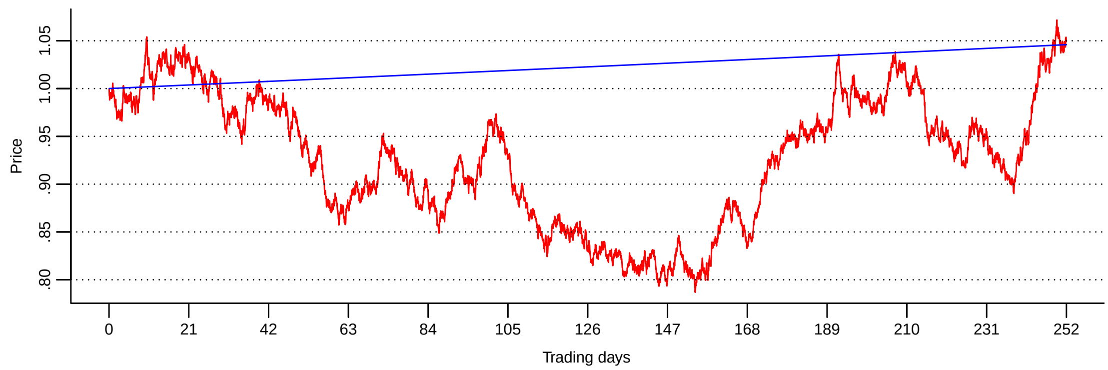


Figure 5.15: Example of price developments of the risky asset (red) and the risk-free asset (blue) that give approximately equal returns after one year.

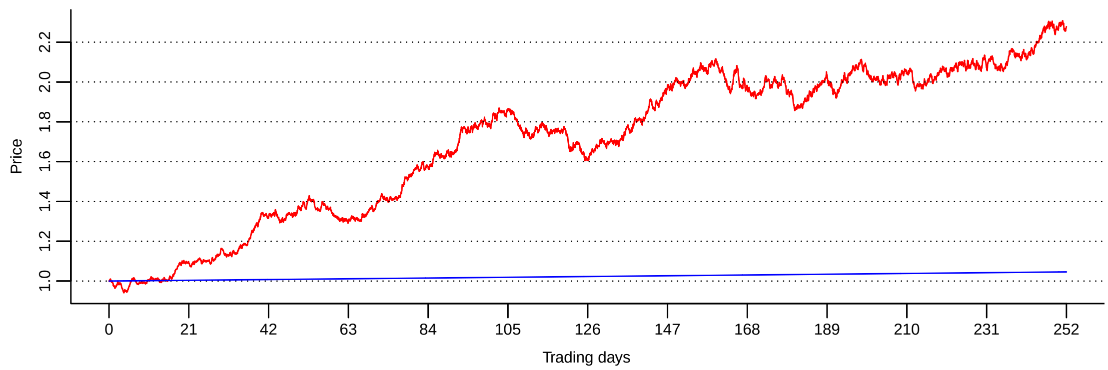


Figure 5.16: Example of price developments of the risky asset (red) and the risk-free asset (blue) that give extremely different returns after one year.

a completely different scenario in which the development of the risky asset is very strong compared to the development of the risk-free asset. A scenario like this would lead to a high total transaction cost even for the annual-rebalancing strategy, because of the in the end huge difference between the return on risky and the risk-free asset. In this specific scenario, the total transaction costs for the hourly-rebalancing strategy are .1419. For the annual-rebalancing strategy the total costs are 8.1830×10^{-3}. For the monthly-rebalancing strategy the total costs are 1.0053×10^{-2}. We see that the difference in total transaction costs between the annual-strategy and the monthly-strategy are relatively small. The reason for this is the relative steady increase of the risky asset price. We can conclude that for a high-frequency rebalancing strategy, frequent but small transactions is the norm, whereas a less frequent rebalancing strategy would imply fewer but potentially larger transaction.

The above reasoning explains the shapes of the distributions of the total transaction costs of each rebalancing strategy, given by the histograms of figure 5.17. An interesting comparison in the context of this example can be found by compar-

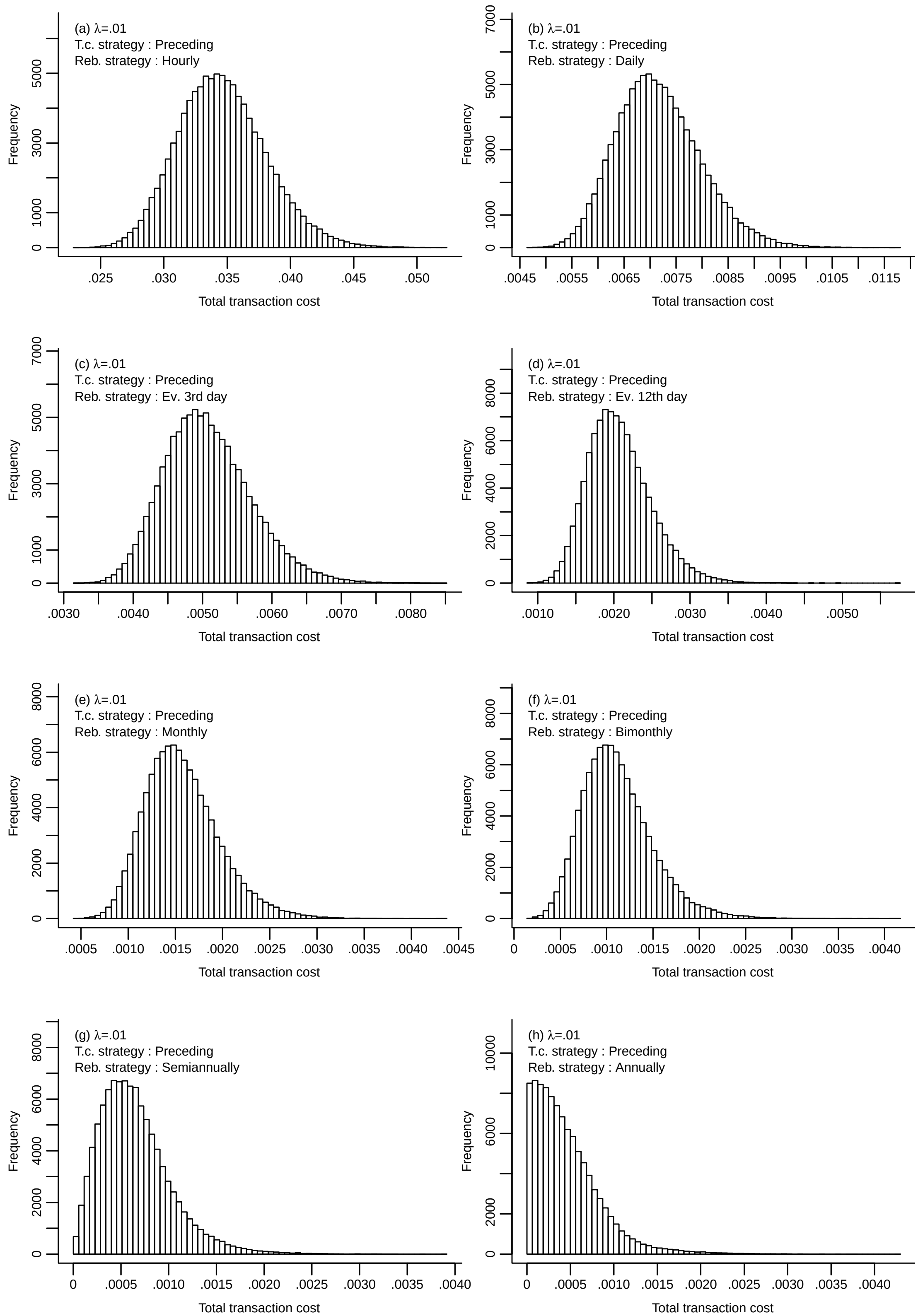


Figure 5.17: Distributions of total transaction costs.

ing the histograms of (e) the monthly-rebalancing strategy and (h) the annual-rebalancing strategy. The range of the total transaction costs of the monthly-strategy is [4.1961×10^{-4},4.3752×10^{-3}], whereas the range of the annual-rebalancing strategy is [4.3000×10^{-8},4.2977×10^{-3}], i.e. quite similar ranges. The distributions are however very different. We observe that the distribution of the total transaction costs of the hourly-strategy is similar to a normal distribution, but as the rebalancing frequency is reduced, the bell-shape is gradually "transformed" into a right skewed distribution.

Table 5.7 show the results of the simulations with transaction cost proportion $\lambda = .02$. It is clear that with $\lambda = .02$ the mean total transaction costs of each rebalancing strategy are approximately doubled compared to the previous simulation scenario with $\lambda = .01$. This doubling of mean total transaction costs also results in an approximate doubling of the mean loss of wealth. The mean losses of utility are also approximately doubled. There are in other words no surprises in table 5.7.

Table 5.8 show the results of the simulations with transaction cost proportion $\lambda = .03$. Compared to the simulation scenario with $\lambda = .01$, we observe an approximate tripling of the mean losses of wealth, the mean total transaction costs and the mean losses of utility.

The plots of figure 5.18 summarizes the findings so far. Regarding mean loss of utility, it is clear that the introduction of transaction costs into the simulation model has the biggest influence on the high-frequency rebalancing strategies. We can conclude that there is a positive relationship between rebalancing frequency and mean loss of utility. As for the size of the transaction cost proportion λ, it only serves to proportionally scale the mean losses of utility.

5.3.6 Portfolio return and Sharpe ratio

Table 5.9 show the Sharpe ratio of each rebalancing strategy along with other related statistics. As we saw with the mean losses of utility in tables 5.6, 5.7 and 5.8 of the previous subsection, the introduction of transaction costs has the most negative effect on the high-frequency rebalancing strategies. We observe for example that the mean terminal log returns are reduced quite a bit for such rebalancing strategies, compared to the simulations without transaction costs. Using equation (5.10) to calculate the Sharpe ratio, we see that this reduction in log returns has a direct impact on the Sharpe ratio. As a result, the performances of portfolios using high-frequency rebalancing strategies are quite poor according to the Sharpe ratio. We observe for example that the hourly-strategy and the 'every 4th hour'-strategy have got the worst Sharpe ratio scores and are ranked last and second-last according to the ratio. This is a completely different picture

$\lambda = .02$			Sample means				StDev
	Simulation model		Term. wealth	Total cost	Term. utility	Loss of utility	Loss of utility
Rebalancing strategy	Hourly	Th	1.0611	-	1.0278	-	-
		No	1.0611	-	1.0278	0	0.1472×10^{-3}
		Pre	0.9909	6.8106×10^{-2}	0.9915	3.6288×10^{-2}	3.3461×10^{-3}
		Sub	0.9909	6.8093×10^{-2}	0.9915	3.6281×10^{-2}	3.3377×10^{-3}
	Every 4th hour	Th	1.0597	-	1.0271	-	-
		No	1.0597	-	1.0271	$-.0001\times10^{-2}$	$.1892\times10^{-3}$
		Pre	1.0241	3.4611×10^{-2}	1.0088	1.8294×10^{-2}	1.7144×10^{-3}
		Sub	1.0241	3.4604×10^{-2}	1.0088	1.8291×10^{-2}	1.7063×10^{-3}
	Daily	Th	1.0609	-	1.0278	-	-
		No	1.0609	-	1.0278	$-.0002\times10^{-2}$	$.3619\times10^{-3}$
		Pre	1.0462	1.4291×10^{-2}	1.0202	$.7514\times10^{-2}$	$.7334\times10^{-3}$
		Sub	1.0462	1.4287×10^{-2}	1.0202	$.7513\times10^{-2}$	$.7257\times10^{-3}$
	Every 3rd day	Th	1.0606	-	1.0275	-	-
		No	1.0606	-	1.0275	$-.0001\times10^{-2}$	$.4980\times10^{-3}$
		Pre	1.0501	1.0126×10^{-2}	1.0222	$.5317\times10^{-2}$	$.5605\times10^{-3}$
		Sub	1.0501	1.0123×10^{-2}	1.0222	$.5316\times10^{-2}$	$.5534\times10^{-3}$
	Every 12th day	Th	1.0618	-	1.0282	-	-
		No	1.0618	-	1.0282	$.0007\times10^{-2}$	1.1806×10^{-3}
		Pre	1.0575	$.4148\times10^{-2}$	1.0260	$.2180\times10^{-2}$	$.8785\times10^{-3}$
		Sub	1.0576	$.4146\times10^{-2}$	1.0260	$.2179\times10^{-2}$	$.8768\times10^{-3}$
	Monthly	Th	1.0607	-	1.0276	-	-
		No	1.0607	-	1.0276	$.0006\times10^{-2}$	1.5531×10^{-3}
		Pre	1.0575	$.3139\times10^{-2}$	1.0260	$.1648\times10^{-2}$	1.2317×10^{-3}
		Sub	1.0575	$.3138\times10^{-2}$	1.0260	$.1648\times10^{-2}$	1.2309×10^{-3}
	Bimonthly	Th	1.0605	-	1.0275	-	-
		No	1.0605	-	1.0275	$.0008\times10^{-2}$	2.1770×10^{-3}
		Pre	1.0582	$.2226\times10^{-2}$	1.0264	$.1170\times10^{-2}$	1.8471×10^{-3}
		Sub	1.0582	$.2225\times10^{-2}$	1.0264	$.1170\times10^{-2}$	1.8468×10^{-3}
	Semiannually	Th	1.0619	-	1.0282	-	-
		No	1.0619	-	1.0282	$.0004\times10^{-2}$	3.7852×10^{-3}
		Pre	1.0606	$.1301\times10^{-2}$	1.0275	$.0674\times10^{-2}$	3.4497×10^{-3}
		Sub	1.0606	$.1299\times10^{-2}$	1.0275	$.0674\times10^{-2}$	3.4501×10^{-3}
	Annually	Th	1.0614	-	1.028	-	-
		No	1.0614	-	1.0279	$.0029\times10^{-2}$	5.3368×10^{-3}
		Pre	1.0605	$.0930\times10^{-2}$	1.0275	$.0500\times10^{-2}$	5.0048×10^{-3}
		Sub	1.0605	$.0928\times10^{-2}$	1.0275	$.0500\times10^{-2}$	5.0054×10^{-3}

Table 5.7: Mean losses of utility and other statistics, $\lambda = .02$.

$\lambda = .03$	Simulation model		Sample means: Term. wealth	Total cost	Term. utility	Loss of utility	StDev Loss of utility
Rebalancing strategy	Hourly	Th	1.0614	-	1.0280	-	-
		No	1.0614	-	1.0280	0	$.1474\times10^{-3}$
		Pre	.9578	10.0463×10^{-2}	.9740	5.3970×10^{-2}	4.9752×10^{-3}
		Sub	.9579	10.0420×10^{-2}	.9740	5.3948×10^{-2}	4.9559×10^{-3}
	Every 4th hour	Th	1.0611	-	1.0278	-	-
		No	1.0611	-	1.0278	0	$.1896\times10^{-3}$
		Pre	1.0080	5.1514×10^{-2}	1.0005	2.7345×10^{-2}	2.5674×10^{-3}
		Sub	1.0080	5.1491×10^{-2}	1.0005	2.7334×10^{-2}	2.5489×10^{-3}
	Daily	Th	1.0602	-	1.0274	-	-
		No	1.0602	-	1.0274	0	$.3616\times10^{-3}$
		Pre	1.0382	2.1355×10^{-2}	1.0161	1.1250×10^{-2}	1.0941×10^{-3}
		Sub	1.0382	2.1343×10^{-2}	1.0162	1.1245×10^{-2}	1.0766×10^{-3}
	Every 3rd day	Th	1.0602	-	1.0274	-	-
		No	1.0602	-	1.0274	$-.0001\times10^{-2}$	$.4937\times10^{-3}$
		Pre	1.0446	1.5144×10^{-2}	1.0194	$.7965\times10^{-2}$	$.7967\times10^{-3}$
		Sub	1.0446	1.5135×10^{-2}	1.0194	$.7962\times10^{-2}$	$.7794\times10^{-3}$
	Every 12th day	Th	1.0610	-	1.0277	-	-
		No	1.0610	-	1.0277	$-.0005\times10^{-2}$	1.1791×10^{-3}
		Pre	1.0546	$.6228\times10^{-2}$	1.0245	$.3258\times10^{-2}$	$.7686\times10^{-3}$
		Sub	1.0546	$.6223\times10^{-2}$	1.0245	$.3257\times10^{-2}$	$.7619\times10^{-3}$
	Monthly	Th	1.0609	-	1.0277	-	-
		No	1.0609	-	1.0277	$-.0005\times10^{-2}$	1.5459×10^{-3}
		Pre	1.0561	$.4718\times10^{-2}$	1.0253	$.2463\times10^{-2}$	1.0857×10^{-3}
		Sub	1.0561	$.4713\times10^{-2}$	1.0253	$.2462\times10^{-2}$	1.0823×10^{-3}
	Bimonthly	Th	1.0613	-	1.0279	-	-
		No	1.0613	-	1.0279	0	2.1959×10^{-3}
		Pre	1.0579	$.3345\times10^{-2}$	1.0262	$.1744\times10^{-2}$	1.7048×10^{-3}
		Sub	1.0579	$.3340\times10^{-2}$	1.0262	$.1744\times10^{-2}$	1.7039×10^{-3}
	Seminannualy	Th	1.0613	-	1.0279	-	-
		No	1.0614	-	1.0279	$-.0021\times10^{-2}$	3.8273×10^{-3}
		Pre	1.0594	$.1957\times10^{-2}$	1.0269	$.0987\times10^{-2}$	3.3234×10^{-3}
		Sub	1.0594	$.1953\times10^{-2}$	1.0269	$.0987\times10^{-2}$	3.3240×10^{-3}
	Annually	Th	1.0606	-	1.0276	0×10^{-2}	0×10^{-3}
		No	1.0607	-	1.0276	$-.0005\times10^{-2}$	5.3980×10^{-3}
		Pre	1.0593	$.1400\times10^{-2}$	1.0269	$.0705\times10^{-2}$	4.8986×10^{-3}
		Sub	1.0593	$.1397\times10^{-2}$	1.0269	$.0705\times10^{-2}$	4.8998×10^{-3}

Table 5.8: Mean losses of utility and other statistics, $\lambda = .03$.

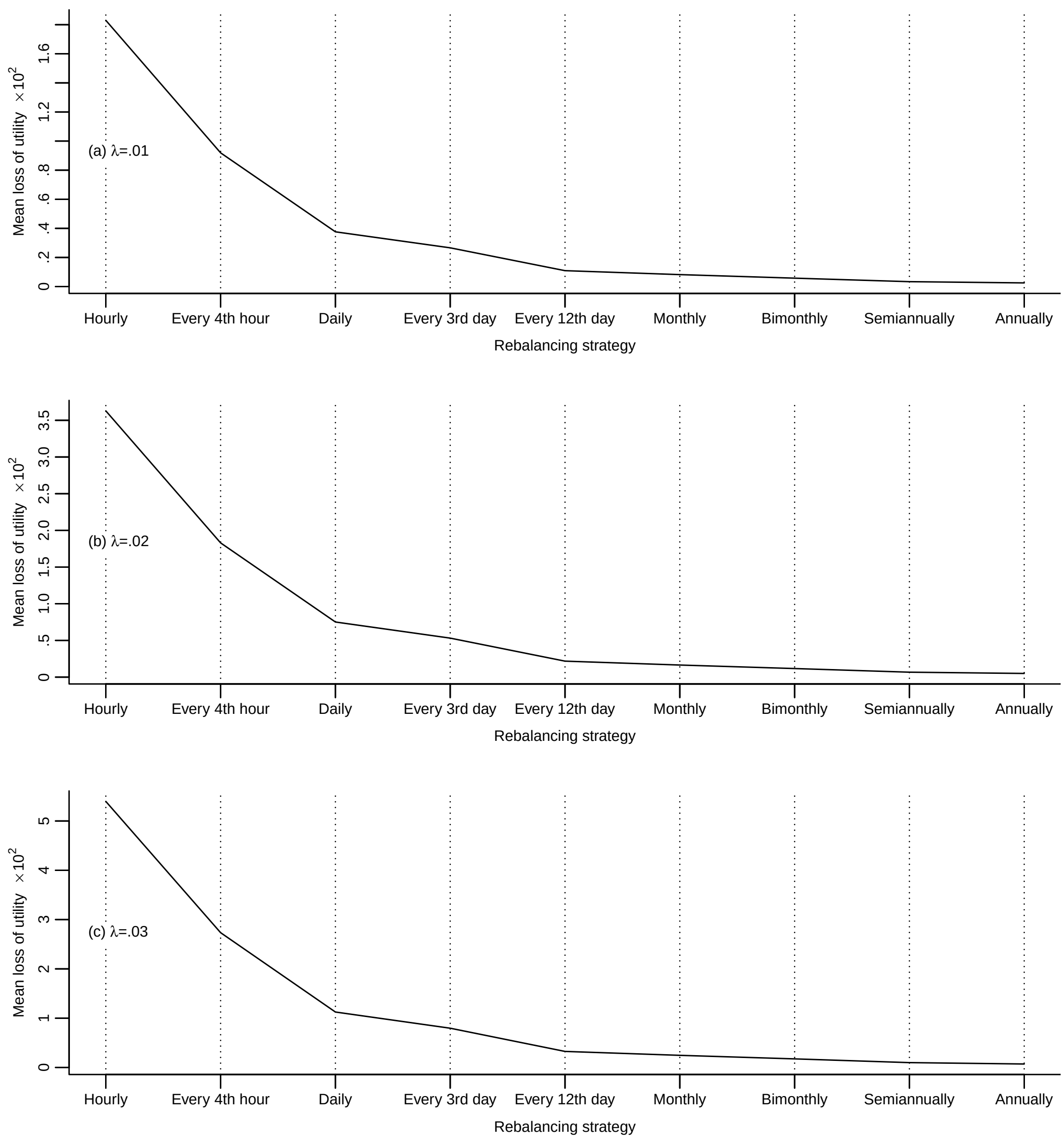


Figure 5.18: Mean losses of utility: (a) $\lambda = .01$, (b) $\lambda = .02$ and (c) $\lambda = .03$.

$\lambda = .01$			Sample means			Vol.		
	Simulation model		Terminal log return	Vol.	Sharpe ratio	of vol.	Corr.	Rank
Rebalancing strategy	Hourly	Th	4.4128×10^{-2}	.1728	-.0045	1.5629×10^{-3}	-.0004	9
		No	4.4129×10^{-2}	.1728	-.0044	1.5630×10^{-3}	-.0051	
		Pre	$.9934\times10^{-2}$	.1728	-.2023	1.5631×10^{-3}	-.0105	
		Sub	$.9935\times10^{-2}$	.1728	-.2023	1.5631×10^{-3}	-.0105	
	Every 4th hour	Th	4.4074×10^{-2}	.1728	-.0048	1.5696×10^{-3}	.0004	8
		No	4.4074×10^{-2}	.1728	-.0048	1.5696×10^{-3}	-.0021	
		Pre	2.6977×10^{-2}	.1728	-.1037	1.5697×10^{-3}	-.0048	
		Sub	2.6978×10^{-2}	.1728	-.1037	1.5697×10^{-3}	-.0048	
	Daily	Th	4.4093×10^{-2}	.1728	-.0047	1.5713×10^{-3}	.0006	6
		No	4.4093×10^{-2}	.1728	-.0048	1.5716×10^{-3}	.0128	
		Pre	3.7113×10^{-2}	.1728	-.0452	1.5716×10^{-3}	.0117	
		Sub	3.7113×10^{-2}	.1728	-.0452	1.5716×10^{-3}	.0117	
	Every 3rd day	Th	4.4313×10^{-2}	.1728	-.0034	1.5697×10^{-3}	.0004	5
		No	4.4315×10^{-2}	.1728	-.0037	1.5709×10^{-3}	.0304	
		Pre	3.9379×10^{-2}	.1728	-.0322	1.5710×10^{-3}	.0297	
		Sub	3.9379×10^{-2}	.1728	-.0322	1.5710×10^{-3}	.0297	
	Every 12th day	Th	4.3583×10^{-2}	.1728	-.0077	1.5706×10^{-3}	.0034	3
		No	4.3576×10^{-2}	.1728	-.0096	1.6162×10^{-3}	.2041	
		Pre	4.1562×10^{-2}	.1728	-.0212	1.6162×10^{-3}	.2038	
		Sub	4.1562×10^{-2}	.1728	-.0212	1.6162×10^{-3}	.2038	
	Monthly	Th	4.4947×10^{-2}	.1728	.0002	1.5784×10^{-3}	.0060	1
		No	4.4952×10^{-2}	.1727	-.0031	1.7153×10^{-3}	.3406	
		Pre	4.3428×10^{-2}	.1727	-.0119	1.7153×10^{-3}	.3404	
		Sub	4.3428×10^{-2}	.1727	-.0119	1.7153×10^{-3}	.3404	
	Bimonthly	Th	4.3660×10^{-2}	.1728	-.0072	1.5709×10^{-3}	-.0003	2
		No	4.3671×10^{-2}	.1727	-.0138	2.0635×10^{-3}	.5602	
		Pre	4.2594×10^{-2}	.1727	-.0201	2.0634×10^{-3}	.5601	
		Sub	4.2593×10^{-2}	.1727	-.0201	2.0634×10^{-3}	.5601	
	Seminannualy	Th	4.4203×10^{-2}	.1728	-.0040	1.5656×10^{-3}	.0009	4
		No	4.4217×10^{-2}	.1726	-.0241	4.3086×10^{-3}	.8052	
		Pre	4.3594×10^{-2}	.1726	-.0277	4.3085×10^{-3}	.8052	
		Sub	4.3594×10^{-2}	.1726	-.0277	4.3085×10^{-3}	.8052	
	Annually	Th	4.4210×10^{-2}	.1728	-.0040	1.5678×10^{-3}	.0036	7
		No	4.4218×10^{-2}	.1723	-.0443	8.1598×10^{-3}	.8474	
		Pre	4.3780×10^{-2}	.1723	-.0469	8.1595×10^{-3}	.8476	
		Sub	4.3779×10^{-2}	.1723	-.0469	8.1595×10^{-3}	.8476	

Table 5.9: The Sharpe ratio versus rebalancing strategy and other statistics, $\lambda = .01$.

compared to what we saw in connection with the simulations without transaction costs, where the hourly-strategy was ranked first. But similarly to what we saw in connection with the simulations without transaction costs, the Sharpe ratio is also affected by the spread of the volatilities of the log returns, that is the volatility of the volatilities, and the correlation between log returns and volatilities. And as before, the annual-rebalancing strategy has the highest volatility of volatilities and correlation, which means that this strategy, according to the Sharpe ratio, ranks seventh, even though the mean total transaction costs for this strategy were lowest. In connection with the simulations assuming no transaction costs, this strategy ranked last. The best rebalancing strategy, assuming $\lambda = .01$, is the monthly-strategy. Even though this strategy has higher mean total transaction costs than the bimonthly, semiannual and annual strategy, it has, according to the Sharpe ratio, lower risk. The monthly-strategy has in other words the best trade-off between return and risk.

Table 5.10 show that increasing the transaction cost proportion λ from .01 to .02 approximately doubles the differences between the theoretical log returns and the log returns of the portfolio models with transaction costs. This obviously causes the Sharpe ratios to decrease compared to the Sharpe ratios of the simulated portfolios with $\lambda = .01$. If we use the Sharpe ratios of the simulated portfolios with no transaction costs (the 'No' category) as baseline, increasing λ from .01 to .02 causes a doubling of the Sharpe ratios in the negative direction. This time, according to the Sharpe ratio, the 'every 12th day'-strategy performs best, but if we use the Sharpe ratio of the 'Th'-category as reference point we find that the bimonthly-strategy performs best. An increase in transaction cost proportion should disfavour high-frequency rebalancing strategies relatively more than rebalancing strategies with lower rebalancing frequencies, because of the bigger increase in total transaction costs of the high-frequency rebalancing strategies. This is just what we see, comparing the Sharpe ratios of the first batch of simulations with $\lambda = .01$ with the second batch of simulations with $\lambda = .02$. The best performing rebalancing strategy moves towards a rebalancing strategy with fewer rebalancings when the transaction cost proportion increases so to speak.

Table 5.11 show the mean Sharpe ratios of the simulated portfolios when we assume $\lambda = .03$. The discussion in this situation is basically the same as for the previous discussion, where we assumed $\lambda = .02$, except that all references to "double" and "doubling" must be changed with "triple" and "tripling". With $\lambda = .03$, according to the Sharpe ratio, the bimonthly-strategy performs best even when we use the mean Sharpe ratios of the 'Th'-category as the point of reference.

The plots of figure 5.19 show each rebalancing strategy plotted against its corresponding Sharpe ratio for (a) $\lambda = .01$, (b) $\lambda = .02$ and (c) $\lambda = .03$. The plots show what we already have discussed, that the Sharpe ratios of high-frequency

$\lambda = .02$	Simulation model		Sample means: Terminal log return	Vol.	Sharpe ratio	Vol. of vol.	Corr.	Rank
Rebalancing strategy	Hourly	Th	4.4315×10^{-2}	.1728	-.0034	1.5704×10^{-3}	-.0034	9
		No	4.4315×10^{-2}	.1728	-.0033	1.5705×10^{-3}	-.0081	
		Pre	-2.4085×10^{-2}	.1728	-.3992	1.5706×10^{-3}	-.0190	
		Sub	-2.4074×10^{-2}	.1728	-.3991	1.5707×10^{-3}	-.0190	
	Every 4th hour	Th	4.3043×10^{-2}	.1728	-.0107	1.5719×10^{-3}	-.0041	8
		No	4.3044×10^{-2}	.1728	-.0107	1.5720×10^{-3}	-.0066	
		Pre	$.8844\times10^{-2}$	.1728	-.2086	1.5721×10^{-3}	-.0121	
		Sub	$.8849\times10^{-2}$	.1728	-.2086	1.5722×10^{-3}	-.0121	
	Daily	Th	4.4297×10^{-2}	.1728	-.0035	1.5729×10^{-3}	.0007	7
		No	4.4301×10^{-2}	.1728	-.0036	1.5732×10^{-3}	.0128	
		Pre	3.0335×10^{-2}	.1728	-.0844	1.5733×10^{-3}	.0106	
		Sub	3.0336×10^{-2}	.1728	-.0844	1.5733×10^{-3}	.0106	
	Every 3rd day	Th	4.3775×10^{-2}	.1728	-.0065	1.5743×10^{-3}	-.0023	6
		No	4.3777×10^{-2}	.1728	-.0068	1.5753×10^{-3}	.0277	
		Pre	3.3903×10^{-2}	.1728	-.0639	1.5753×10^{-3}	.0261	
		Sub	3.3903×10^{-2}	.1728	-.0639	1.5753×10^{-3}	.0261	
	Every 12th day	Th	4.5095×10^{-2}	.1728	.0011	1.5722×10^{-3}	-.0021	1
		No	4.5084×10^{-2}	.1728	-.0008	1.6168×10^{-3}	.1987	
		Pre	4.1059×10^{-2}	.1728	-.0241	1.6169×10^{-3}	.1981	
		Sub	4.1058×10^{-2}	.1728	-.0241	1.6169×10^{-3}	.1981	
	Monthly	Th	4.4022×10^{-2}	.1728	-.0051	1.5743×10^{-3}	-.0011	3
		No	4.4015×10^{-2}	.1727	-.0084	1.7072×10^{-3}	.3338	
		Pre	4.0971×10^{-2}	.1727	-.0260	1.7072×10^{-3}	.3334	
		Sub	4.0970×10^{-2}	.1727	-.0260	1.7072×10^{-3}	.3333	
	Bimonthly	Th	4.3852×10^{-2}	.1728	-.0061	1.5716×10^{-3}	.0057	2
		No	4.3844×10^{-2}	.1727	-.0128	2.0619×10^{-3}	.5604	
		Pre	4.1691×10^{-2}	.1727	-.0253	2.0620×10^{-3}	.5602	
		Sub	4.1690×10^{-2}	.1727	-.0253	2.0619×10^{-3}	.5601	
	Seminannualy	Th	4.5077×10^{-2}	.1728	.0011	1.5696×10^{-3}	-.0045	4
		No	4.5081×10^{-2}	.1726	-.0190	4.2968×10^{-3}	.8031	
		Pre	4.3838×10^{-2}	.1726	-.0262	4.2963×10^{-3}	.8032	
		Sub	4.3837×10^{-2}	.1726	-.0262	4.2962×10^{-3}	.8032	
	Annually	Th	4.4697×10^{-2}	.1728	-.0012	1.5723×10^{-3}	.0012	5
		No	4.4675×10^{-2}	.1723	-.0414	8.1477×10^{-3}	.8450	
		Pre	4.3800×10^{-2}	.1723	-.0465	8.1464×10^{-3}	.8453	
		Sub	4.3798×10^{-2}	.1723	-.0466	8.1463×10^{-3}	.8453	

Table 5.10: The Sharpe ratio versus rebalancing strategy and other statistics, $\lambda = .02$.

λ = .03	Simulation model		Sample means: Terminal log return	Sample means: Vol.	Sample means: Sharpe ratio	Vol. of vol.	Corr.	Rank
Rebalancing strategy	Hourly	Th	4.4601×10^{-2}	.1728	-.0017	1.5722×10^{-3}	.0008	9
		No	4.4601×10^{-2}	.1728	-.0017	1.5723×10^{-3}	-.0039	
		Pre	-5.8024×10^{-2}	.1728	-.5957	1.5725×10^{-3}	-.0201	
		Sub	-5.7984×10^{-2}	.1728	-.5953	1.5728×10^{-3}	-.0201	
	Every 4th hour	Th	4.4315×10^{-2}	.1728	-.0034	1.5702×10^{-3}	.0038	8
		No	4.4314×10^{-2}	.1728	-.0034	1.5702×10^{-3}	.0013	
		Pre	$-.6996\times10^{-2}$	.1728	-.3003	1.5703×10^{-3}	-.0068	
		Sub	$-.6978\times10^{-2}$	.1728	-.3002	1.5704×10^{-3}	-.0068	
	Daily	Th	4.3672×10^{-2}	.1728	-.0071	1.5694×10^{-3}	.0033	7
		No	4.3672×10^{-2}	.1728	-.0072	1.5696×10^{-3}	.0154	
		Pre	2.2724×10^{-2}	.1728	-.1285	1.5698×10^{-3}	.0122	
		Sub	2.2729×10^{-2}	.1728	-.1284	1.5698×10^{-3}	.0122	
	Every 3rd day	Th	4.3517×10^{-2}	.1728	-.0080	1.5726×10^{-3}	-.0009	6
		No	4.3518×10^{-2}	.1728	-.0083	1.5738×10^{-3}	.0289	
		Pre	2.8708×10^{-2}	.1728	-.0940	1.5740×10^{-3}	.0265	
		Sub	2.8711×10^{-2}	.1728	-.0939	1.5740×10^{-3}	.0265	
	Every 12th day	Th	4.4182×10^{-2}	.1728	-.0041	1.5728×10^{-3}	-.0025	4
		No	4.4192×10^{-2}	.1728	-.0060	1.6176×10^{-3}	.1991	
		Pre	3.8142×10^{-2}	.1728	-.0410	1.6178×10^{-3}	.1982	
		Sub	3.8142×10^{-2}	.1728	-.0410	1.6178×10^{-3}	.1982	
	Monthly	Th	4.4146×10^{-2}	.1728	-.0044	1.5696×10^{-3}	.0001	3
		No	4.4159×10^{-2}	.1727	-.0076	1.7081×10^{-3}	.3372	
		Pre	3.9585×10^{-2}	.1728	-.0341	1.7084×10^{-3}	.3366	
		Sub	3.9584×10^{-2}	.1728	-.0341	1.7084×10^{-3}	.3365	
	Bimonthly	Th	4.4515×10^{-2}	.1728	-.0022	1.5757×10^{-3}	-.0066	1
		No	4.4518×10^{-2}	.1727	-.0088	2.0600×10^{-3}	.5552	
		Pre	4.1287×10^{-2}	.1727	-.0275	2.0600×10^{-3}	.5549	
		Sub	4.1286×10^{-2}	.1727	-.0276	2.0599×10^{-3}	.5548	
	Seminannualy	Th	4.4488×10^{-2}	.1728	-.0024	1.5700×10^{-3}	.0012	2
		No	4.4546×10^{-2}	.1726	-.0222	4.3092×10^{-3}	.8052	
		Pre	4.2677×10^{-2}	.1726	-.0331	4.3080×10^{-3}	.8053	
		Sub	4.2675×10^{-2}	.1726	-.0331	4.3077×10^{-3}	.8052	
	Annually	Th	4.3852×10^{-2}	.1728	-.0061	1.5683×10^{-3}	.0036	5
		No	4.3892×10^{-2}	.1722	-.0465	8.2000×10^{-3}	.8455	
		Pre	4.2574×10^{-2}	.1722	-.0542	8.1970×10^{-3}	.8459	
		Sub	4.2572×10^{-2}	.1722	-.0542	8.1965×10^{-3}	.8459	

Table 5.11: The Sharpe ratio versus rebalancing strategy and other statistics, $\lambda = .03$.

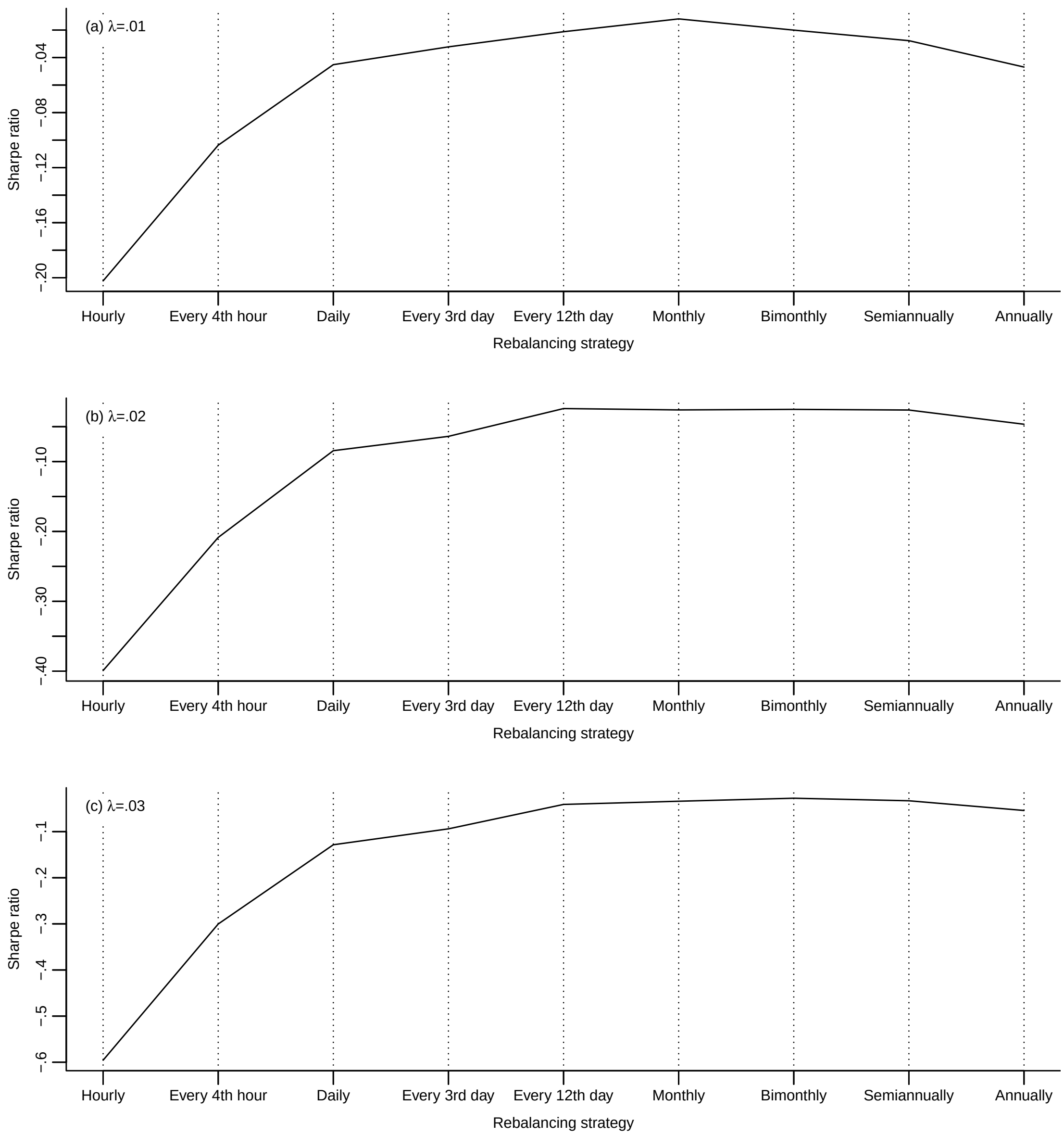


Figure 5.19: Sharpe ratios versus rebalancing strategies: (a) $\lambda = .01$, (b) $\lambda = .02$ and (c) $\lambda = .03$.

rebalancing strategies are punished by the high transaction costs of such strategies. The annual rebalancing strategy are punished by higher risk. The 'every 12th day', the monthly, the bimonthly and the semiannual rebalancing strategy are the best strategies according to the Sharpe ratio.

5.4 Simulation with stochastic volatility

5.4.1 Introduction

The first simulation model, that is simulation model I, was a rather unrealistic model. In section 5.3 we increased the complexity and the realism of the model by introducing transaction costs. One shortcoming of simulation models I, II and III is the assumption of constant volatility. As the plots of short-term volatilities of figure 4.4 of section 4.3 show, this assumption is rather unrealistic. In this section we will further increase the complexity and realism of the simulation model by assuming stochastic volatility. The new simulation model will, perhaps not so surprisingly, be dubbed simulation model IV.

5.4.2 Stochastic volatility

There exists many different models for modelling stochastic volatility. One class of models are driven by Brownian motion(s), such as the CEV model, the SABR volatility model, the GARCH model, the 3/2 model, the Chen model and other models. Another class of stochastic volatility models are the Levy driven models. We will in this thesis use a Brownian motion driven stochastic volatility model, namely the Heston model. The definition of this model is stated in section 2.1. The SDE (2.4), which describes the dynamics of the volatility, is, as stated in that section, a so-called CIR-process. One important property of the CIR-process is mean reversion, which means that in the long run, the process tends to drift towards its long-term mean. This mean reversion tendency is in accordance with evidence from equity markets [6].

A standard method of simulating a Heston stochastic volatility process is through its Euler approximation. Assuming equidistant time increments, the Euler approximation of the SDE (2.4) is straightforwardly

$$\nu_{k+1} = \nu_k + \kappa(\theta - \nu_k)\delta + \xi\sqrt{\nu_k}\Delta B_k^{\nu}.$$

With the introduction of stochastic volatility we need to reconsider the Merton ratio. Remember that the optimal allocation strategy given by the Merton ratio

(3.10) has so far been constant. We now have to take into consideration that the volatility will vary with time when we calculate the Merton ratio, so the optimal allocation strategy has to be redefined as

$$u_t^* = \frac{\mu - r}{\nu_t(1-\gamma)}. \tag{5.16}$$

We see that the optimal allocation strategy now indirectly has become a stochastic quantity. We also see that if the volatility increases, the investor will invest less in the risky asset and vice versa, as it should be reflecting the risk-aversion of the investor.

Compared to simulation model II and III, the transaction quantity Q_k at time t_k, will in simulation model IV, as a consequence of the introduction of stochastic volatility, be slightly altered. We will in the new simulation model only consider preceding transaction costs, not subsequent transaction costs. As argued for before, preceding transaction costs reflect the idea of a rebalanced portfolio at each rebalancing time point more accurately and, as the simulations of section 5.3 did show, the difference in total transaction costs between the two transaction cost methods is minute. Another consequence of the introduction of stochastic volatility is that the direction of the transaction between the risky asset investment and risk-free asset investment at rebalancing time points, is no longer only determined by the difference in returns (5.12) between the risky asset and the risk-free asset since the previous rebalancing time point. The optimal allocation strategy at rebalancing time points must now also be taken into consideration. Assume that t_k is a rebalancing time point. One possible scenario is for example that the return on the risky asset investment since the previous rebalancing time point t_k^* is higher than the return on the risk-free asset investment. Such a scenario would in simulation model I, II and III imply a reduction of the risky asset investment and a corresponding increase of the risk-free asset investment (before the deduction of transaction costs) at time t_k. With stochastic volatility, a high value of u_k could require a reverse transaction even though the return on the risky asset investment is higher. To determine the direction of the transaction in simulation model IV, we need to replace (5.12) with

$$D_k = (1-u_k^*)u_{k^*}^* \prod_{j=k^*}^{k-1}(1+\mu\delta+\sigma_{j+1}\Delta B_j^S) - u_k^*(1-u_{k^*}^*)(1+r\delta)^{k-k^*}$$

Similar to the previous calculations of the transaction quantity, for the portfolio to become rebalanced, we require that

$$u_k^* = \frac{\tilde{V}_k'^S - Q_k}{\tilde{V}_k},$$

$$1 - u_k^* = \begin{cases} \dfrac{\tilde{V}_k'^R + Q_k - \lambda Q_k}{\tilde{V}_k}, & D_k \geq 0 \\ \dfrac{\tilde{V}_k'^R + Q_k + \lambda Q_k}{\tilde{V}_k}, & D_k < 0. \end{cases}$$

The solution with respect to Q_k is

$$Q_k = \begin{cases} \dfrac{(1 - u_k^*)\tilde{V}_k'^S - u_k^* \tilde{V}_k'^R}{1 - \lambda u_k^*}, & D_k \geq 0 \\ \dfrac{(1 - u_k^*)\tilde{V}_k'^S - u_k^* \tilde{V}_k'^R}{1 + \lambda u_k^*}, & D_k < 0. \end{cases}$$

It is clear that the stochastic volatility induces extra variability into the simulation model. The question is, how will this added variability effect the outcomes of the simulations?

5.4.3 Simulation model IV

Simulation model IV
Transaction costs: Preceding
Volatility: Stochastic

$$
\begin{aligned}
\sigma_k &= \sqrt{\nu_k} = \sqrt{\nu_{k-1} + \kappa(\theta - \nu_{k-1})\delta + \xi\sqrt{\nu_{k-1}}\Delta B_{k-1}^{\nu}} \\
u_k^* &= \frac{\mu - r}{\sigma_k^2(1-\gamma)} \\
\tilde{V}_k^{\prime S} &= u_{k^*}^* \tilde{V}_{k^*} \prod_{j=k^*}^{k-1} (1 + \mu\delta + \sigma_{j+1}\Delta B_j^S) \\
\tilde{V}_k^{\prime R} &= (1 - u_{k^*}^*)\tilde{V}_{k^*}(1 + r\delta)^{k-k^*} \\
D_k &= (1 - u_k^*)u_{k^*}^* \prod_{j=k^*}^{k-1} (1 + \mu\delta + \sigma_{j+1}\Delta B_j^S) - u_k^*(1 - u_{k^*}^*)(1 + r\delta)^{k-k^*} \\
Q_k &= \begin{cases} \dfrac{(1-u_k^*)\tilde{V}_k^{\prime S} - u_k^*\tilde{V}_k^{\prime R}}{1 - \lambda u_k^*}, & D_k \geq 0 \\ \dfrac{(1-u_k^*)\tilde{V}_k^{\prime S} - u_k^*\tilde{V}_k^{\prime R}}{1 + \lambda u_k^*}, & D_k < 0 \end{cases} \\
\tilde{V}_k^S &= \begin{cases} \tilde{V}_k^{\prime S} - Q_k, & t_k \in \mathcal{T}^{\text{reb}} \\ \tilde{V}_k^{\prime S}, & \text{otherwise} \end{cases} \\
\tilde{V}_k^R &= \begin{cases} \tilde{V}_k^{\prime R} + Q_k - \lambda|Q_k|, & t_k \in \mathcal{T}^{\text{reb}} \\ \tilde{V}_k^{\prime R}, & \text{otherwise} \end{cases} \\
\tilde{V}_k &= \tilde{V}_k^S + \tilde{V}_k^R.
\end{aligned}
$$

The above framed equations show the required equations of the stochastic volatility simulation model, namely simulation model IV.

5.4.4 Implementation

For the actual simulations of the volatility we will use the estimates found through the linear regression estimation of section 4.3. These estimates are summarized in table 5.12. As for the estimates of μ, r and γ, we use the same parameter estimates as we used in the thesis so far, that is the parameter estimates of table 4.1. Due to the increased complexity and, hence, slower run time of the simulation model

Parameter	Estimate
ν_0	6.6105×10^{-2}
κ	320.1192
θ	6.7456×10^{-2}
ξ	.0590
ρ	2.6706×10^{-2}

Table 5.12: The parameter estimations of the Heston model.

IV, a total 50,000 simulations of each combination of transaction cost proportion and rebalancing strategy, that is a total of 36 combinations (including $\lambda = 0$, that is no transaction costs) were run.

So far in this thesis we have used the theoretical portfolio value (5.2) as the point of reference when measuring the loss of wealth and the loss of utility. The introduction of stochastic volatility gives a sligthly modified expression for the SDE (3.2) of the portfolio value process. The new SDE of the portfolio value process is

$$dV_t = (\mu u_t + r(1 - u_t))V_t dt + \sqrt{\nu_t} u_t V_t dB_t. \tag{5.17}$$

Since a closed form solution of (5.17) similar to (5.2) doesn't exist, it is natural to use a different point of reference. One such point of reference is the simulated discrete time portfolio with constant volatility, that is simulation model III or I, depending on whether we assume preceding transaction costs or not. A natural choice of the constant volatility of the new reference portfolio is the square root of the long-term mean θ. The new constant volatility also implies a modified value of the optimal allocation strategy u^*. Using the estimate of θ, that is 6.7456×10^{-2} along with the estimates of μ, r and γ of table 4.1 yields $u^* = .6498$.

5.4.5 Simulation test run

Figure 5.20 (a) show an example of a stochastic volatility time series, simulated over one year (252 trading days) with hourly updates. The horizontal dotted line indicates the square root of the long-term mean θ of the Heston stochastic volatility process. Figure 5.20 (b) show the corresponding stochastic optimal allocation strategy (5.16). Here, the horizontal dotted line indicates the constant optimal allocation strategy u^* with volatility $\sqrt{\theta}$. These plots just confirm the fact that $u^*_k = constant \cdot \sigma_k^{-2}$.

As mentioned above, one method of comparing stochastic volatility to constant volatility is to use simulation model I or III with constant volatility equal to the square root of the long-term mean θ as a reference. The histogram of figure 5.21

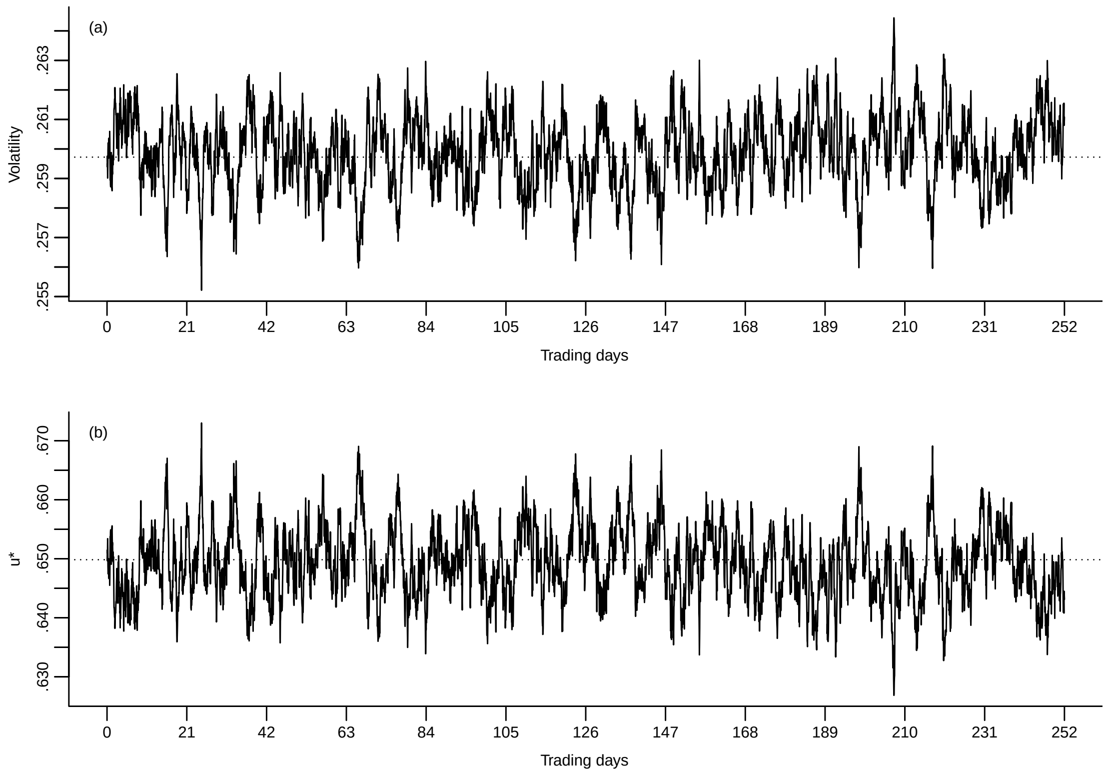


Figure 5.20: Stochastic volatility versus u^*.

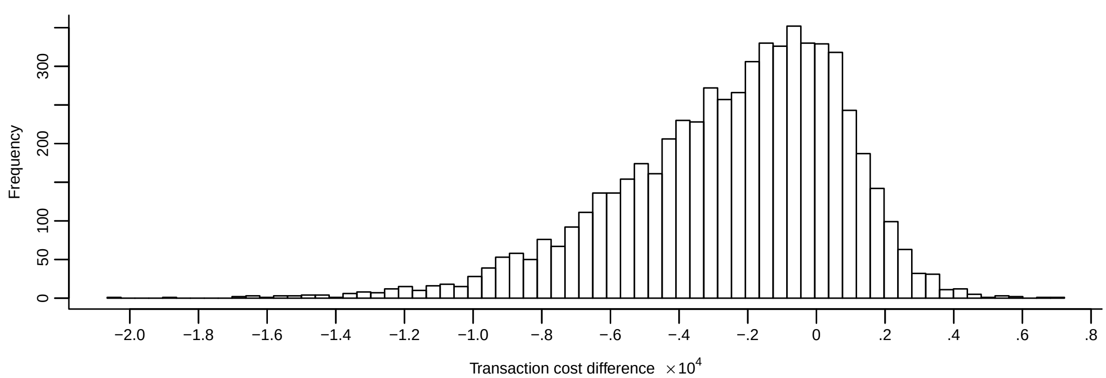


Figure 5.21: Distribution of transaction cost differences between constant volatility model and stochastic volatility model using daily-rebalancing strategy.

show the distribution of the transaction cost differences between such a constant volatility reference portfolio and a portfolio with stochastic volatility, using the same underlying Brownian motion time series for the simulation of the risky asset and assuming hourly rebalancings and transaction cost proportion $\lambda = .03$. An obvious question is: how will the added variability of the stochastic volatility and thereby the added variability of the non-constant optimal allocation strategy affect the transaction costs? Will the added variability increase the total variability and thereby increase the transaction costs, will the added variability in the long run zero out and thereby not change the transaction costs in total or will the added variability counteract the already existing variability caused by the Brownian motion of the risky asset and thereby reduce the transaction costs? The histogram of figure 5.21 give us a mixed picture, but in general we see that the introduction of stochastic volatility and non-constant optimal allocation strategy increases the overall variability of the portfolio and thereby increases the total transaction costs. The total transaction costs of the simulated portfolio assuming constant volatility is .1005. The total transaction costs of the simulated portfolio assuming constant volatility is .2509, which is considerably more. And compared to an initial portfolio value of 1 it is extremely high. Considering the fact that one portfolio simulation run consists of 6048 time points and equally many transaction cost differences (when we assume portfolio rebalancings at an hourly basis) means that the distribution of figure 5.21 as well as the transaction cost totals, should be quite indicative about the general relation between the transaction costs of simulated portfolios assuming constant volatility and simulated portfolios assuming stochastic volatility.

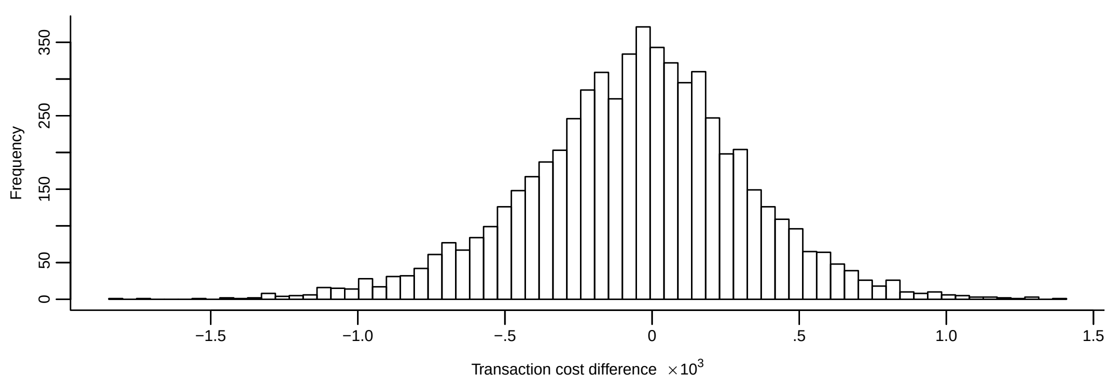


Figure 5.22: Distribution of transaction cost differences between constant volatility model and stochastic volatility model using monthly-rebalancing strategy.

When we use the monthly-rebalancing strategy we get a different picture. The histogram of figure 5.22 shows the distribution of 6000 transaction cost differences[2]. Now, the differences in transaction costs between portfolios assuming

[2]Since one portfolio simulation run using the monthly-strategy only gives twelve transaction cost differences per run, the 6000 differences was obtained from 500 portfolio simulation runs.

constant volatility and portfolios assuming stochastic volatility are fairly evenly distributed around a mean approximately equal to zero.

5.4.6 Mean loss of utility

$\lambda = 0$			Sample means			
	Simulation model		Term. wealth	Total cost	Term. utility	Loss of utility
Rebalancing strategy	Hourly	Const	1.0601	-	1.0275	-
		Stoch	1.0613	-	1.0281	-5.7842×10^{-4}
	Every 4th hour	Const	1.0605	-	1.0277	-
		Stoch	1.0610	-	1.0279	-2.1399×10^{-4}
	Daily	Const	1.0598	-	1.0273	-
		Stoch	1.0600	-	1.0275	-1.4043×10^{-4}
	Every 3rd day	Const	1.0605	-	1.0277	-
		Stoch	1.0614	-	1.0282	-4.7085×10^{-4}
	Every 12th day	Const	1.0601	-	1.0275	-
		Stoch	1.0616	-	1.0282	-7.3880×10^{-4}
	Monthly	Const	1.0596	-	1.0273	-
		Stoch	1.0601	-	1.0275	-2.4584×10^{-4}
	Bimonthly	Const	1.0601	-	1.0275	-
		Stoch	1.0613	-	1.0281	-6.2473×10^{-4}
	Seminannualy	Const	1.0606	-	1.0277	-
		Stoch	1.0601	-	1.0275	2.6877×10^{-4}
	Annually	Const	1.0607	-	1.0278	-
		Stoch	1.0602	-	1.0275	3.0116×10^{-4}

Table 5.13: The mean losses of utility of each rebalancing strategy and other related statistics, $\lambda = 0$.

Table 5.13 and figure 5.23 (a) show the mean losses of utility of each rebalancing strategy whan assuming no transaction costs. Included in table 5.13 are also related statistics. The category "Const" in the table refers to the constant volatility (assumed equal to $\sqrt{\theta}$) portfolio simulations using simulation model I. The results of this category serve as a reference point for assessing the impact of assuming stochastic volatility instead of constant volatility. The category "Stoch" refers to the stochastic volatility portfolio simulations using simulation model IV. All of the statistics of table 5.13 were calculated on a basis of 150,000 portfolio simulation runs for each rebalancing strategy for both constant volatility portfolios and stochastic volatility portfolios, but this time, the simulations were not done in parallel.

The estimates of table 5.13 as well as the plot of figure 5.23 (a) might suggest that the mean terminal utilities of the constant volatility portfolios are not significantly different from the mean terminal utilities of the stochastic volatility portfolios when we assume no transaction costs.

$\lambda = .01$	Simulation model		Sample means			
			Term. wealth	Total cost	Term. utility	Loss of utility
Rebalancing strategy	Hourly	Const	1.0219	3.7062×10^{-2}	1.0079	-
		Stoch	.9607	9.8763×10^{-2}	.9756	3.222×10^{-2}
	Every 4th hour	Const	1.0411	1.8711×10^{-2}	1.0177	-
		Stoch	1.0105	4.8909×10^{-2}	1.0019	1.5796×10^{-2}
	Daily	Const	1.0515	$.7676\times10^{-2}$	1.0231	-
		Stoch	1.0431	1.6667×10^{-2}	1.0189	$.4264\times10^{-2}$
	Every 3rd day	Const	1.0563	$.5442\times10^{-2}$	1.0255	-
		Stoch	1.0515	$.9980\times10^{-2}$	1.0231	$.2421\times10^{-2}$
	Every 12th day	Const	1.0580	$.2227\times10^{-2}$	1.0265	-
		Stoch	1.0598	$.2657\times10^{-2}$	1.0273	$-.0830\times10^{-2}$
	Monthly	Const	1.0576	$.1685\times10^{-2}$	1.0262	-
		Stoch	1.0581	$.1870\times10^{-2}$	1.0265	$-.0264\times10^{-2}$
	Bimonthly	Const	1.0582	$.1194\times10^{-2}$	1.0265	-
		Stoch	1.0620	$.1266\times10^{-2}$	1.0284	$-.1936\times10^{-2}$
	Seminannualy	Const	1.0606	$.0701\times10^{-2}$	1.0277	-
		Stoch	1.0604	$.0708\times10^{-2}$	1.0276	$.0023\times10^{-2}$
	Annually	Const	1.0598	$.0495\times10^{-2}$	1.0274	-
		Stoch	1.0597	$.0503\times10^{-2}$	1.0272	$.0110\times10^{-2}$

Table 5.14: Mean losses of utility of each rebalancing strategy and other statistics, $\lambda = .01$.

As discussed earlier, table 5.14, 5.15 and 5.16 as well as figure 5.23 (b), (c) and (d) show that the introduction of transaction costs has a significant effect on the transaction cost totals of the high-frequency rebalancing strategies, compared to the transaction cost totals of the constant volatility portfolios. This relation is visualized in the histograms of figure 5.24 as well as in the histograms of figure A.5 and figure A.6 in the appendix. Here the distributions of the constant volatility portfolio transaction costs are given as the shaded histograms. The stochastic volatility portfolio transaction costs are in white. According to the confidence intervals of figure 5.23 (b), (c) and (d), there are significant differences from zero for the hourly-, the 'every 4th hour'-, the daily- and the 'every 3rd day'-rebalancing strategies. The value of the transaction cost proportion λ only serve to scale the transaction cost totals.

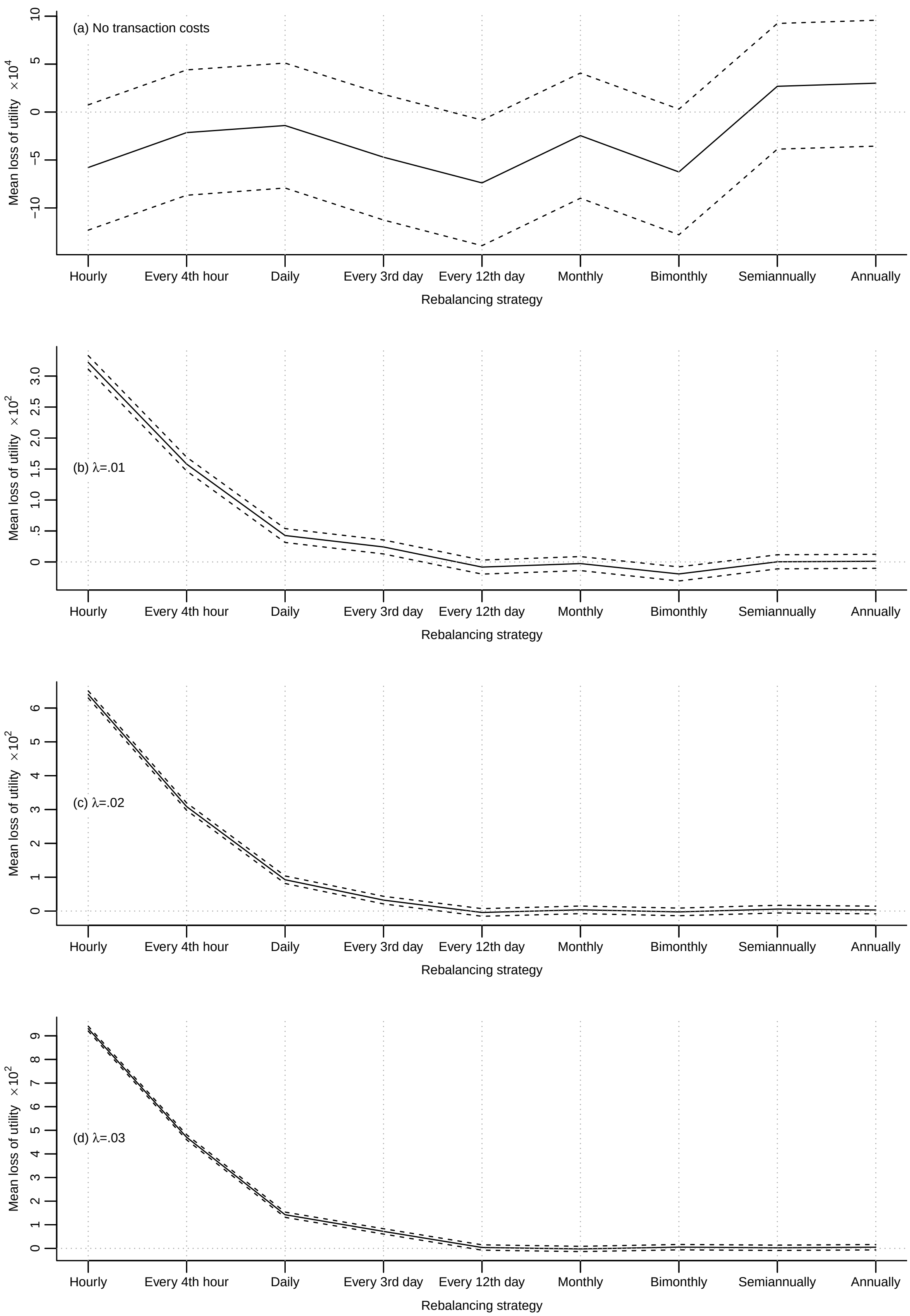


Figure 5.23: Mean losses of utility of each rebalancing strategy with 95% confidence intervals, (a) $\lambda = 0$, (b) $\lambda = .01$, (c) $\lambda = .02$ and (d) $\lambda = .03$.

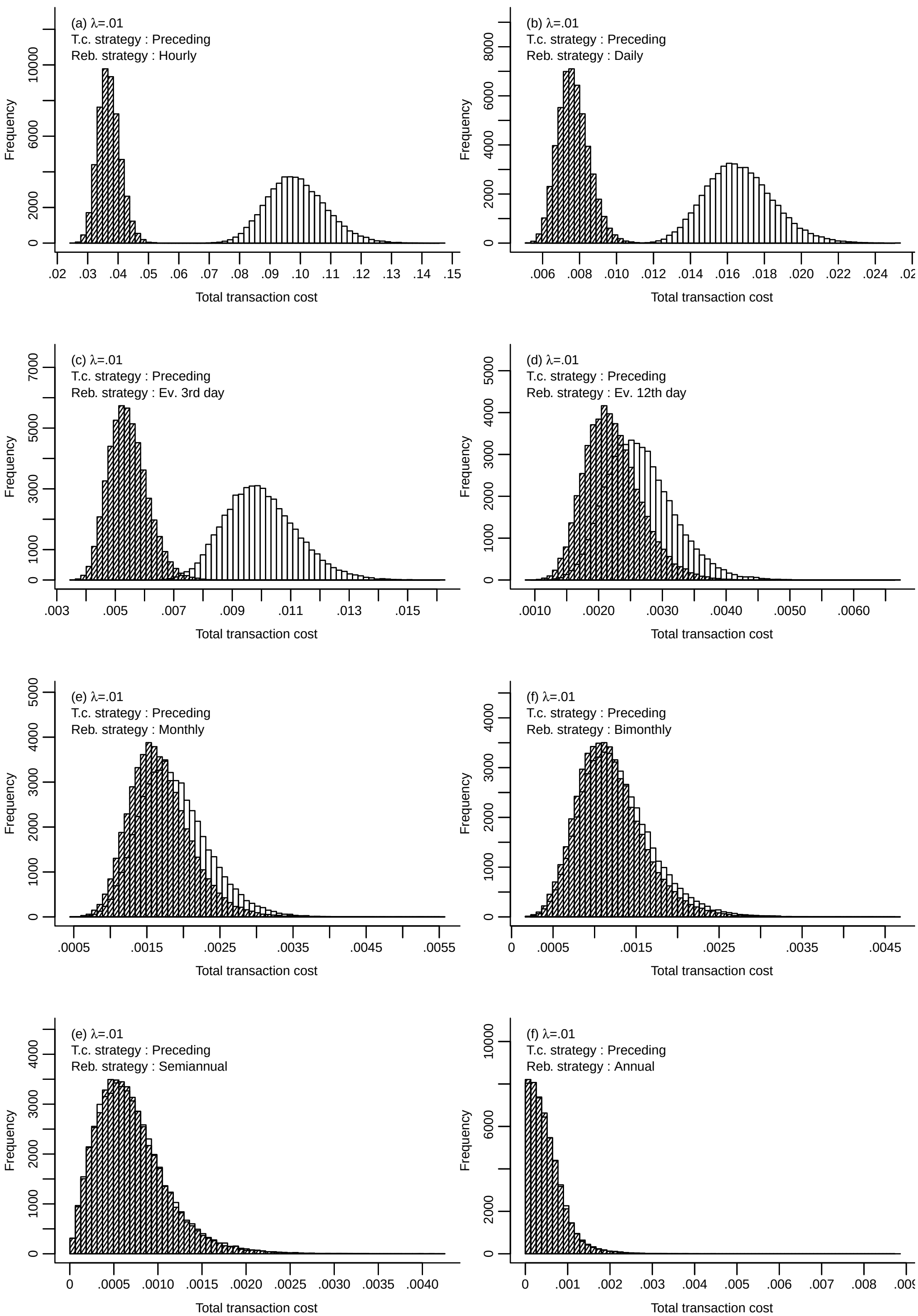


Figure 5.24: Distributions of total transaction costs of stochastic volatility portfolios and constant volatility portfolios (shaded), $\lambda = .01$.

$\lambda = .02$	Simulation model		Sample means: Term. wealth	Total cost	Term. utility	Loss of utility
Rebalancing strategy	Hourly	Const	.9849	7.2801×10^{-2}	.9885	-
		Stoch	.8671	18.7700×10^{-2}	.9245	6.4033×10^{-2}
	Every 4th hour	Const	1.0217	3.7072×10^{-2}	1.0078	-
		Stoch	.9629	9.5487×10^{-2}	.9769	3.0871×10^{-2}
	Daily	Const	1.0435	1.5303×10^{-2}	1.0190	-
		Stoch	1.0255	3.3041×10^{-2}	1.0098	$.9258\times10^{-2}$
	Every 3rd day	Const	1.0477	1.0840×10^{-2}	1.0212	-
		Stoch	1.0415	1.9869×10^{-2}	1.0179	$.3230\times10^{-2}$
	Every 12th day	Const	1.0558	$.4446\times10^{-2}$	1.0253	-
		Stoch	1.0567	$.5303\times10^{-2}$	1.0257	$-.0419\times10^{-2}$
	Monthly	Const	1.0564	$.3368\times10^{-2}$	1.0256	-
		Stoch	1.0557	$.3745\times10^{-2}$	1.0253	$.0357\times10^{-2}$
	Bimonthly	Const	1.0572	$.2386\times10^{-2}$	1.0260	-
		Stoch	1.0578	$.2525\times10^{-2}$	1.0263	$-.0253\times10^{-2}$
	Seminannualy	Const	1.0590	$.1390\times10^{-2}$	1.0270	-
		Stoch	1.0580	$.1412\times10^{-2}$	1.0264	$.0572\times10^{-2}$
	Annually	Const	1.0595	$.1001\times10^{-2}$	1.0271	-
		Stoch	1.0588	$.1000\times10^{-2}$	1.0268	$.0313\times10^{-2}$

Table 5.15: Mean losses of utility of each rebalancing strategy and other statistics, $\lambda = .02$.

$\lambda = .03$	Simulation model		Sample means: Term. wealth	Total cost	Term. utility	Loss of utility
Rebalancing strategy	Hourly	Const	.9499	10.7296×10^{-2}	.9699	-
		Stoch	.7840	26.8417×10^{-2}	.8769	9.3053×10^{-2}
	Every 4th hour	Const	1.0046	5.5147×10^{-2}	.9989	-
		Stoch	.9165	13.9884×10^{-2}	.9518	4.7043×10^{-2}
	Daily	Const	1.0370	2.2875×10^{-2}	1.0157	-
		Stoch	1.0096	4.9213×10^{-2}	1.0015	1.4247×10^{-2}
	Every 3rd day	Const	1.0441	1.6238×10^{-2}	1.0193	-
		Stoch	1.0301	2.9631×10^{-2}	1.0121	$.7190\times10^{-2}$
	Every 12th day	Const	1.0528	$.6667\times10^{-2}$	1.0238	-
		Stoch	1.0522	$.7932\times10^{-2}$	1.0234	$.0373\times10^{-2}$
	Monthly	Const	1.0545	$.5043\times10^{-2}$	1.0246	-
		Stoch	1.0550	$.5608\times10^{-2}$	1.0249	$-.0240\times10^{-2}$
	Bimonthly	Const	1.0575	$.3589\times10^{-2}$	1.0261	-
		Stoch	1.0564	$.3764\times10^{-2}$	1.0256	$.0516\times10^{-2}$
	Semiannualy	Const	1.0580	$.2084\times10^{-2}$	1.0265	-
		Stoch	1.0576	$.2122\times10^{-2}$	1.0262	$.0247\times10^{-2}$
	Annually	Const	1.0599	$.1502\times10^{-2}$	1.0273	-
		Stoch	1.0589	$.1509\times10^{-2}$	1.0268	$.0489\times10^{-2}$

Table 5.16: Mean losses of utility of each rebalancing strategy and other statistics, $\lambda = .03$.

5.4.7 Portfolio return and Sharpe ratio

$\lambda = 0$	Simulation model		Sample means: Terminal log return	Sample means: Vol.	Sample means: Sharpe ratio	Vol. of vol.	Corr.	Rank
Rebalancing strategy	Hourly	Const	4.4192×10^{-2}	.1688	$-.4145\times10^{-2}$	1.5343×10^{-3}	-.0056	2
		Stock	4.5225×10^{-2}	.1688	$.1968\times10^{-2}$	1.5360×10^{-3}	-.0049	
	Every 4th hour	Const	4.4526×10^{-2}	.1688	$-.2207\times10^{-2}$	1.5386×10^{-3}	-.0007	4
		Stoch	4.4925×10^{-2}	.1688	$.0176\times10^{-2}$	1.5302×10^{-3}	-.0031	
	Daily	Const	4.3856×10^{-2}	.1688	$-.6348\times10^{-2}$	1.5332×10^{-3}	.0177	5
		Stoch	4.4137×10^{-2}	.1688	$-.4662\times10^{-2}$	1.5382×10^{-3}	.0156	
	Every 3rd day	Const	4.4417×10^{-2}	.1688	$-.3171\times10^{-2}$	1.5381×10^{-3}	.0336	1
		Stoch	4.5351×10^{-2}	.1688	$.2358\times10^{-2}$	1.5372×10^{-3}	.0346	
	Every 12th day	Const	4.4167×10^{-2}	.1688	$-.6466\times10^{-2}$	1.5870×10^{-3}	.2258	3
		Stoch	4.5440×10^{-2}	.1688	$.1108\times10^{-2}$	1.6178×10^{-3}	.2173	
	Monthly	Const	4.3719×10^{-2}	.1687	-1.0707×10^{-2}	1.6989×10^{-3}	.3691	7
		Stoch	4.4163×10^{-2}	.1688	$-.8058\times10^{-2}$	1.7483×10^{-3}	.3571	
	Bimonthly	Const	4.4080×10^{-2}	.1687	-1.2349×10^{-2}	2.1204×10^{-3}	.5955	6
		Stoch	4.5230×10^{-2}	.1687	$-.5567\times10^{-2}$	2.2039×10^{-3}	.5755	
	Semi-annualy	Const	4.4613×10^{-2}	.1686	-2.4237×10^{-2}	4.6609×10^{-3}	.8151	8
		Stoch	4.4103×10^{-2}	.1686	-2.7296×10^{-2}	4.7218×10^{-3}	.8049	
	Annually	Const	4.4725×10^{-2}	.1682	-4.6118×10^{-2}	8.9094×10^{-3}	.8488	9
		Stoch	4.4119×10^{-2}	.1682	-4.9781×10^{-2}	8.8958×10^{-3}	.8480	

Table 5.17: Sharpe ratios of each rebalancing strategy and other related statistics, $\lambda = 0$.

Table 5.17 displays, among other statistics, the Sharpe ratios of each rebalancing strategy when assuming no transaction costs. The same Sharpe ratios with 95% confidence intervals are plotted in figure 5.25 (a). The picture is basically similar to what we get when assuming constant volatility: The best performing rebalancing strategies according to the Sharpe ratio are the high-frequency rebalancing strategies. We see that the 'every 3rd day'-strategy is ranked as number one, but with more simulations and consequently more precise estimates, the hourly-strategy would probably be ranked first, similar to the rankings of section 5.2.6.

Tables 5.18, 5.19 and 5.20 assumes transaction costs. Similar to what we saw in section 5.3.6, the introduction of transaction costs has the most negative effect on high-frequency rebalancing strategies. By comparing the 'Stoch'- with the 'Const'-category, we see that this effect is much stronger when we in addition assume stochastic volatility. We also see that the best performing rebalancing strategy this time is the bimonthly-strategy. This is the case for all three transaction cost proportions, although by small margin.

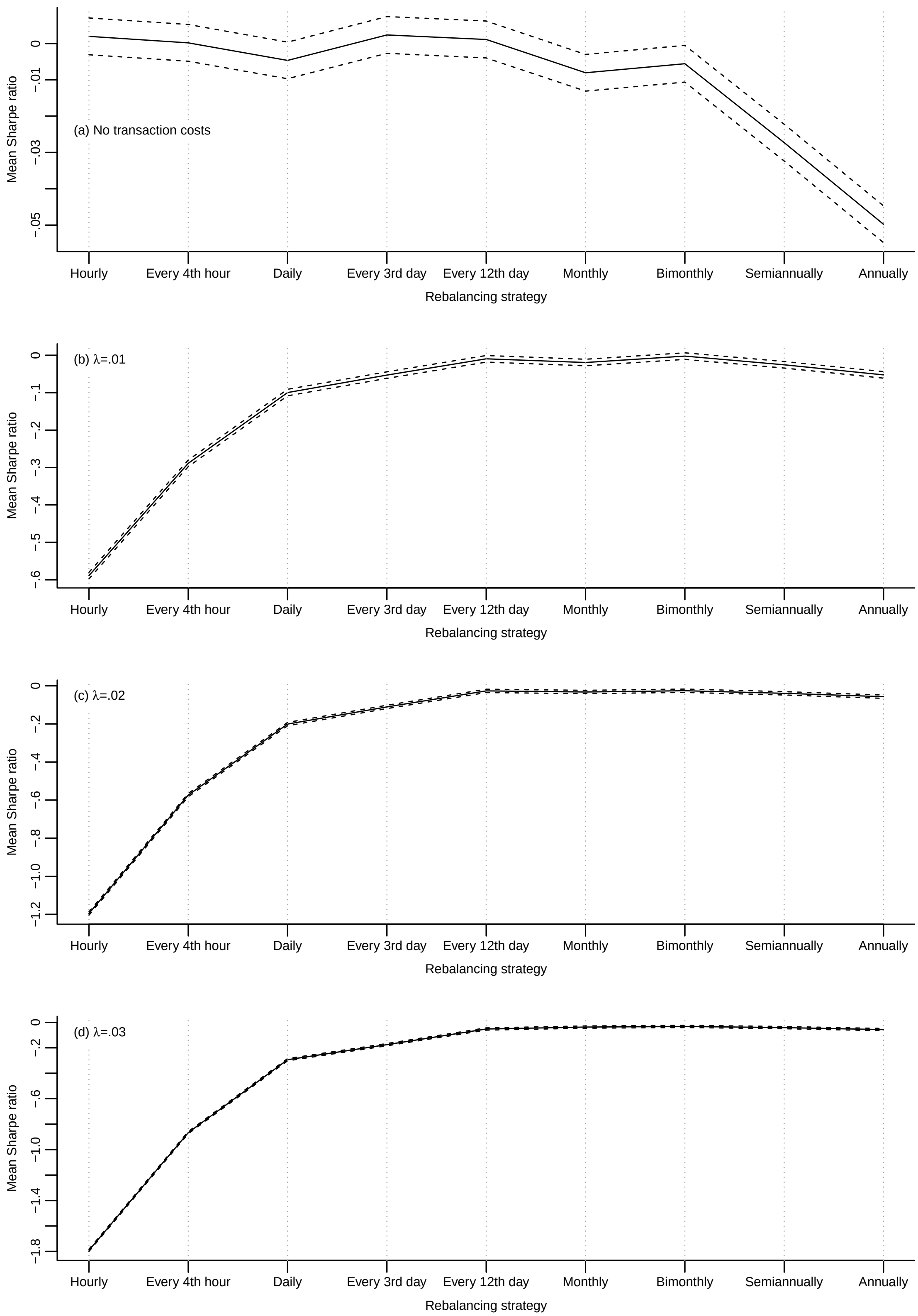


Figure 5.25: Sharpe ratios with 95% confidence intervals, (a) $\lambda = 0$, (b) $\lambda = .01$, (c) $\lambda = .02$ and (d) $\lambda = .03$.

$\lambda = .01$	Simulation model		Sample means: Terminal log return	Sample means: Vol.	Sample means: Sharpe ratio	Vol. of vol.	Corr.	Rank
Rebalancing strategy	Hourly	Const	$.7422\times10^{-2}$	.1688	-.2220	1.5315×10^{-3}	-.0086	9
		Stoch	-5.4509×10^{-2}	.1688	-.5889	1.5385×10^{-3}	-.0115	
	Every 4th hour	Const	2.5900×10^{-2}	.1688	-.1126	1.5385×10^{-3}	-.0030	8
		Stoch	$-.3836\times10^{-2}$	.1688	-.2888	1.5280×10^{-3}	-.0006	
	Daily	Const	3.6028×10^{-2}	.1688	-.0527	1.5378×10^{-3}	.0134	7
		Stoch	2.8118×10^{-2}	.1688	-.0996	1.5311×10^{-3}	.0184	
	Every 3rd day	Const	4.0425×10^{-2}	.1688	-.0269	1.5446×10^{-3}	.0377	6
		Stoch	3.6018×10^{-2}	.1688	-.0529	1.5399×10^{-3}	.0299	
	Every 12th day	Const	4.2247×10^{-2}	.1688	-.0179	1.5918×10^{-3}	.2279	2
		Stoch	4.3668×10^{-2}	.1688	-.0094	1.6110×10^{-3}	.2230	
	Monthly	Const	4.1755×10^{-2}	.1687	-.0223	1.6951×10^{-3}	.3672	3
		Stoch	4.2273×10^{-2}	.1688	-.0193	1.7447×10^{-3}	.3611	
	Bimonthly	Const	4.2232×10^{-2}	.1687	-.0233	2.1240×10^{-3}	.5958	1
		Stoch	4.5814×10^{-2}	.1687	-.0022	2.2065×10^{-3}	.5790	
	Semi-annualy	Const	4.4347×10^{-2}	.1686	-.0262	4.6881×10^{-3}	.8179	4
		Stoch	4.4413×10^{-2}	.1686	-.0254	4.7186×10^{-3}	.8044	
	Annually	Const	4.4053×10^{-2}	.1682	-.0496	8.8753×10^{-3}	.8472	5
		Stoch	4.3685×10^{-2}	.1682	-.0524	8.8856×10^{-3}	.8480	

Table 5.18: Sharpe ratios of each rebalancing strategy and other related statistics, $\lambda = .01$.

$\lambda = .02$	Simulation model		Sample means: Terminal log return	Sample means: Vol.	Sample means: Sharpe ratio	Vol. of vol.	Corr.	Rank
Rebalancing strategy	Hourly	Const	-2.9378×10^{-2}	.1688	-.4400	1.5331×10^{-3}	-.0194	9
		Stoch	-15.6862×10^{-2}	.1688	-1.1955	1.5325×10^{-3}	-.0097	
	Every 4th hour	Const	$.7239\times10^{-2}$	.1688	-.2231	1.5413×10^{-3}	-.0112	8
		Stoch	-5.1978×10^{-2}	.1688	-.5739	1.5315×10^{-3}	-.0092	
	Daily	Const	2.8353×10^{-2}	.1688	-.0982	1.5250×10^{-3}	.0166	7
		Stoch	1.1048×10^{-2}	.1688	-.2007	1.5479×10^{-3}	.0150	
	Every 3rd day	Const	3.2349×10^{-2}	.1688	-.0746	1.5296×10^{-3}	.0257	6
		Stoch	2.6277×10^{-2}	.1688	-.1106	1.5332×10^{-3}	.0319	
	Every 12th day	Const	4.0068×10^{-2}	.1688	-.0307	1.5786×10^{-3}	.2196	2
		Stoch	4.0788×10^{-2}	.1688	-.0264	1.6269×10^{-3}	.2148	
	Monthly	Const	4.0712×10^{-2}	.1687	-.0285	1.6968×10^{-3}	.3684	3
		Stoch	4.0073×10^{-2}	.1688	-.0322	1.7472×10^{-3}	.3530	
	Bimonthly	Const	4.1463×10^{-2}	.1687	-.0278	2.1187×10^{-3}	.5917	1
		Stoch	4.1845×10^{-2}	.1687	-.0257	2.2022×10^{-3}	.5780	
	Semi-annualy	Const	4.3199×10^{-2}	.1686	-.0325	4.6486×10^{-3}	.8136	4
		Stoch	4.2087×10^{-2}	.1685	-.0392	4.7045×10^{-3}	.8055	
	Annually	Const	4.3483×10^{-2}	.1682	-.0538	8.9180×10^{-3}	.8498	5
		Stoch	4.2930×10^{-2}	.1682	-.0569	8.9166×10^{-3}	.8478	

Table 5.19: Sharpe ratios of each rebalancing strategy and other related statistics, $\lambda = .02$.

$\lambda = .03$	Simulation model		Sample means: Terminal log return	Sample means: Vol.	Sample means: Sharpe ratio	Vol. of vol.	Corr.	Rank
Rebalancing strategy	Hourly	Const	-6.5555×10^{-2}	.1687	-.6544	1.5387×10^{-3}	-.0243	9
		Stoch	-25.7451×10^{-2}	.1688	-1.7915	1.5371×10^{-3}	-.0077	
	Every 4th hour	Const	$-.9602\times10^{-2}$	.1688	-.3229	1.5363×10^{-3}	-.0057	8
		Stoch	-10.1419×10^{-2}	.1688	-.8667	1.5319×10^{-3}	-.0070	
	Daily	Const	2.2271×10^{-2}	.1688	-.1342	1.5374×10^{-3}	.0159	7
		Stoch	$-.4647\times10^{-2}$	.1688	-.2936	1.5366×10^{-3}	.0099	
	Every 3rd day	Const	2.8717×10^{-2}	.1688	-.0962	1.5402×10^{-3}	.0323	6
		Stoch	1.5383×10^{-2}	.1688	-.1752	1.5396×10^{-3}	.0389	
	Every 12th day	Const	3.7227×10^{-2}	.1688	-.0476	1.5912×10^{-3}	.2280	4
		Stoch	3.6426×10^{-2}	.1688	-.0523	1.6164×10^{-3}	.2123	
	Monthly	Const	3.8898×10^{-2}	.1688	-.0393	1.7053×10^{-3}	.3703	2
		Stoch	3.9259×10^{-2}	.1688	-.0371	1.7535×10^{-3}	.3558	
	Bimonthly	Const	4.1618×10^{-2}	.1687	-.0270	2.1186×10^{-3}	.5984	1
		Stoch	4.0734×10^{-2}	.1687	-.0321	2.2030×10^{-3}	.5687	
	Semi-annualy	Const	4.2305×10^{-2}	.1685	-.0377	4.6438×10^{-3}	.8141	3
		Stoch	4.1753×10^{-2}	.1686	-.0414	4.7405×10^{-3}	.8050	
	Annually	Const	4.3823×10^{-2}	.1683	-.0518	8.9301×10^{-3}	.8503	5
		Stoch	4.2909×10^{-2}	.1682	-.0570	8.8796×10^{-3}	.8491	

Table 5.20: Sharpe ratios of each rebalancing strategy and other related statistics, $\lambda = .03$.

Chapter 6

Conclusion

To recapitulate, in the most basic version of Merton's portfolio problem we assume that an investor has two investment choices, a risky asset, where the price dynamics is described by an SDE known as a geometric Brownian motion, and a risk-free asset, where the price dynamics is described by deterministic differential equation. The solution to Merton's portfolio problem, that is the optimal allocation strategy or trading strategy, is to keep a constant fraction u^* of the wealth in the risky asset and consequently a constant fraction $1-u^*$ of the wealth in the risk-free asset. This is a frequently used strategy among different investors such as banks, investment funds etc.

To answer the question about how the constant allocation strategy performs in a more realistic discrete time scenario, we introduced a time discretization and transferred the continuous SDE of the portfolio value into a discrete time counterpart by an Euler approximation. This gave us a simple, iterative method of simulating portfolios using the constant allocation strategy. Each portfolio simulation run, simulates the portfolio value over a period of one year, that is 252 trading days.

The constant allocation strategy requires that the investor rebalances the portfolio. In Merton's portfolio problem the investor is allowed to rebalance the portfolio continuously in time. In our discrete time simulation scenario the investor is only allowed to rebalance the portfolio at discrete time points, which is a more realistic assumption. By only allowing the investor to rebalance the portfolio at certain subsets of the complete set of time points, we were able to simulate and compare different rebalancing strategies.

To measure the impact of discretization, different rebalancing strategies and later transaction costs and stochastic volatility, we calculated the mean losses of utility and the Sharpe ratios of the different outcomes of the different simulation model configurations.

In simulation model I, we made the rather naive assumptions of no transaction costs and that the volatility of the risky asset is constant. Under these assumptions we found that to rebalance the portfolio as frequently as possible gave the best results, both in terms of mean loss of utility and mean Sharpe ratio, although the mean losses of utility of the semiannual-strategy and the annual-strategy weren't very far from zero.

In simulation model II and III we introduced transaction costs. We assumed proportional transaction costs, which means that the transaction costs were calculated as a proportionality constant times the amount transacted. The introduction of transaction costs had the biggest impact on the high-frequency rebalancing strategies. We concluded that for such strategies, small but frequent transactions were the norm. Rebalancing strategies with longer time intervals between each portfolio rebalancing entailed fewer, but potentially larger transactions at each rebalancing time point. "Potentially" is the keyword here, because even though high-frequency strategies meant small transactions, the sum of many small such transactions and consequently the sum of many small transactions costs, turned out in sum to generally be much more costly than to rebalance the portfolio less frequently. As a consequence we found that in terms of mean loss of utility, the best strategy is to rebalance the portfolio as seldom as possible. A simulation time interval of one year meant that the annual-rebalancing strategy was the best choice in terms of mean loss of utility and that the hourly-strategy was the worst. In terms of mean Sharpe ratio the picture was a little bit more complicated. In the simulation model without transaction costs we saw that for the low-frequency rebalancing strategies, such as the semiannual or the annual-strategy, the Sharpe ratio indicated a lower reward-to-risk ratio for such strategies, because of higher correlation between return and risk. This specific picture was more or less the same after the introduction of transaction costs, but the transaction costs meant that also high-frequency rebalancing strategies got low Sharpe ratios, not because of increased risk or correlation between return and volatility, but because of high transaction cost totals and consequently lower returns. The combined effect of the correlation between return and volatility and high transaction cost totals for the high-frequency rebalancing strategies, meant that the medium-frequency rebalancing strategies such as the monthly or the bimonthly strategy got the best Sharpe ratios in this scenario. We also looked at two different approaches with regard to the calculation of the transaction costs themselves. At each rebalancing time point, one approach was to rebalance the portfolio first and then deduct the transaction cost from the bank account (the risk-free asset). We referred to this approach as subsequent transaction costs. The second approach was to require the portfolio to be rebalanced after the transaction had been deducted. This approach, we referred to as preceding transaction costs. We found that the differences between these two approaches were minimal, and in practice perhaps not very relevant. We also looked at three , different transaction cost proportions,

$\lambda = .01$, $\lambda = .02$ and $\lambda = .03$. We found that the λ scaled both the mean losses of utility and the mean Sharpe ratios proportionally.

In the last simulation model, simulation model IV, we did the more realistic assumption of stochastic volatility as opposed to the more unrealistic assumption of constant volatility. For modelling the stochastic volatility we used the well-known Heston model. The new stochastic volatility also implied a non-constant optimal allocation strategy. We found that this additional variability had a very negative impact on the mean losses of utility and the mean Sharpe ratios of portfolios with transaction costs using high-frequency rebalancing strategies. Compared to the mean losses of utility and the mean Sharpe ratios of constant volatility portfolio simulations, only portfolios using the hourly-, the 'every 4th hour'-, the daily- and the 'every 3rd day'-rebalancing strategy performed significantly worse. In terms of mean Sharpe ratio we basically saw the same picture as we did in the constant volatility scenario, but with even worse ratios for the four most frequent strategies due to the increased transaction cost totals. The best performing rebalancing strategy in the stochastic volatility scenario was the bimonthly-strategy.

A main focus of this thesis has been to build more or less realistic simulation models for assessing the performance of the constant allocation strategy, predicted as the optimal allocation strategy by Merton, in a discrete time scenario. Even though the simulation models of this thesis surely are more realistic than the continuous-time-, no transaction costs-, constant volatility-model that is assumed in Merton's portfolio problem, it has to be acknowledged that there are a lot of shortcomings in this thesis' simulation models as well. Firstly, we assume that the risky asset dynamics follow a geometric Brownian motion which implies normally distributed log returns. Research show that this is an unrealistic assumption, at least for the distribution of short-term log returns. Secondly, one might question the ability of the Heston model to simulate daily volatilities realistically. It seems that a better stochastic volatility model could have increased the realism of the simulations quite a bit. There are also many other ways of increasing the realism, for example by introducing stochastic drift, stochastic risk-free rate of return, better risky asset models and so forth. Of course, the disadvantage of making a model extremely complex is that it might lose generality and even become too complex to analyse and interpret.

Appendix A

Additional plots

A.1 Simulation model II and III

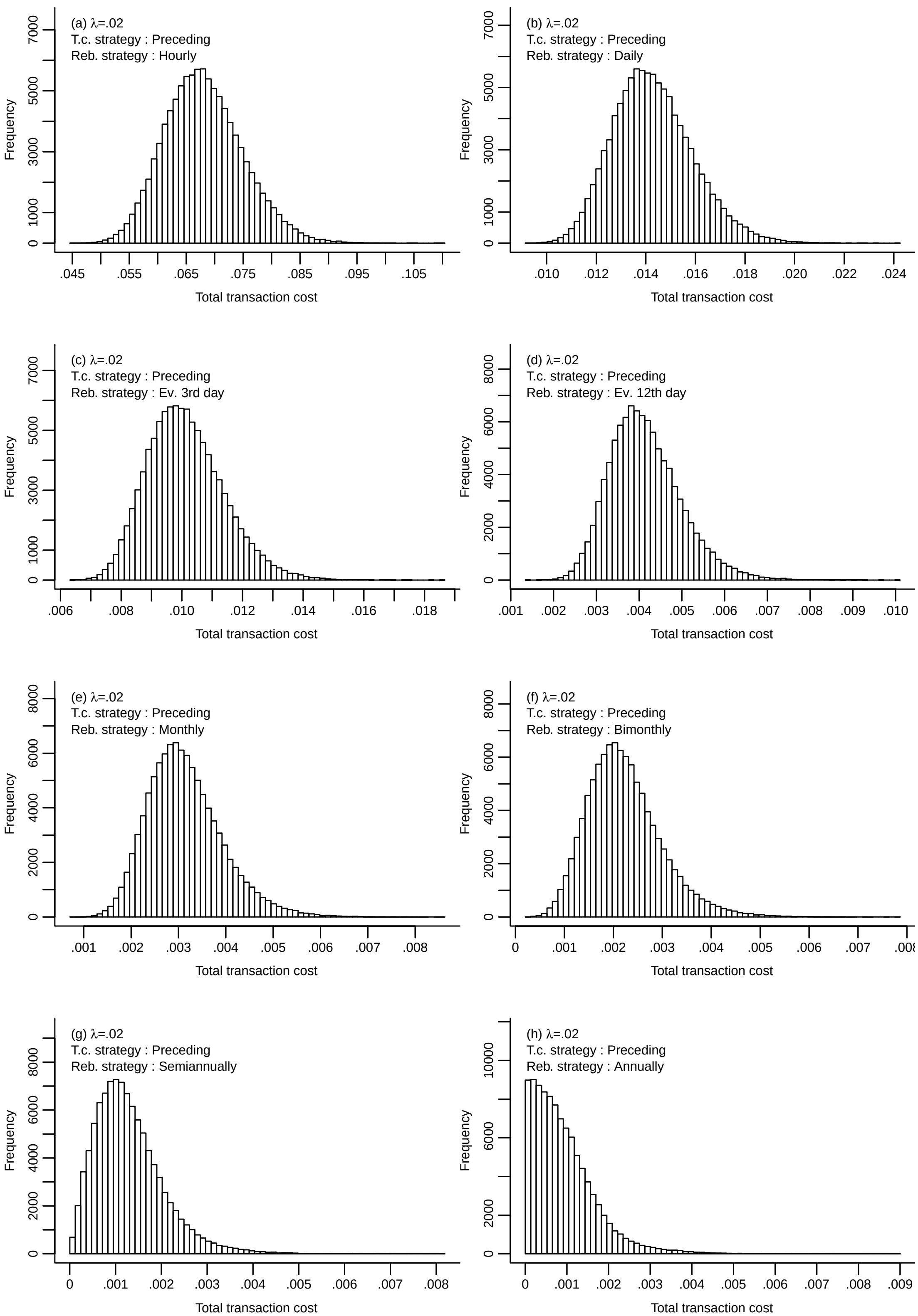


Figure A.1: Distributions of total transaction costs, $\lambda = .02$.

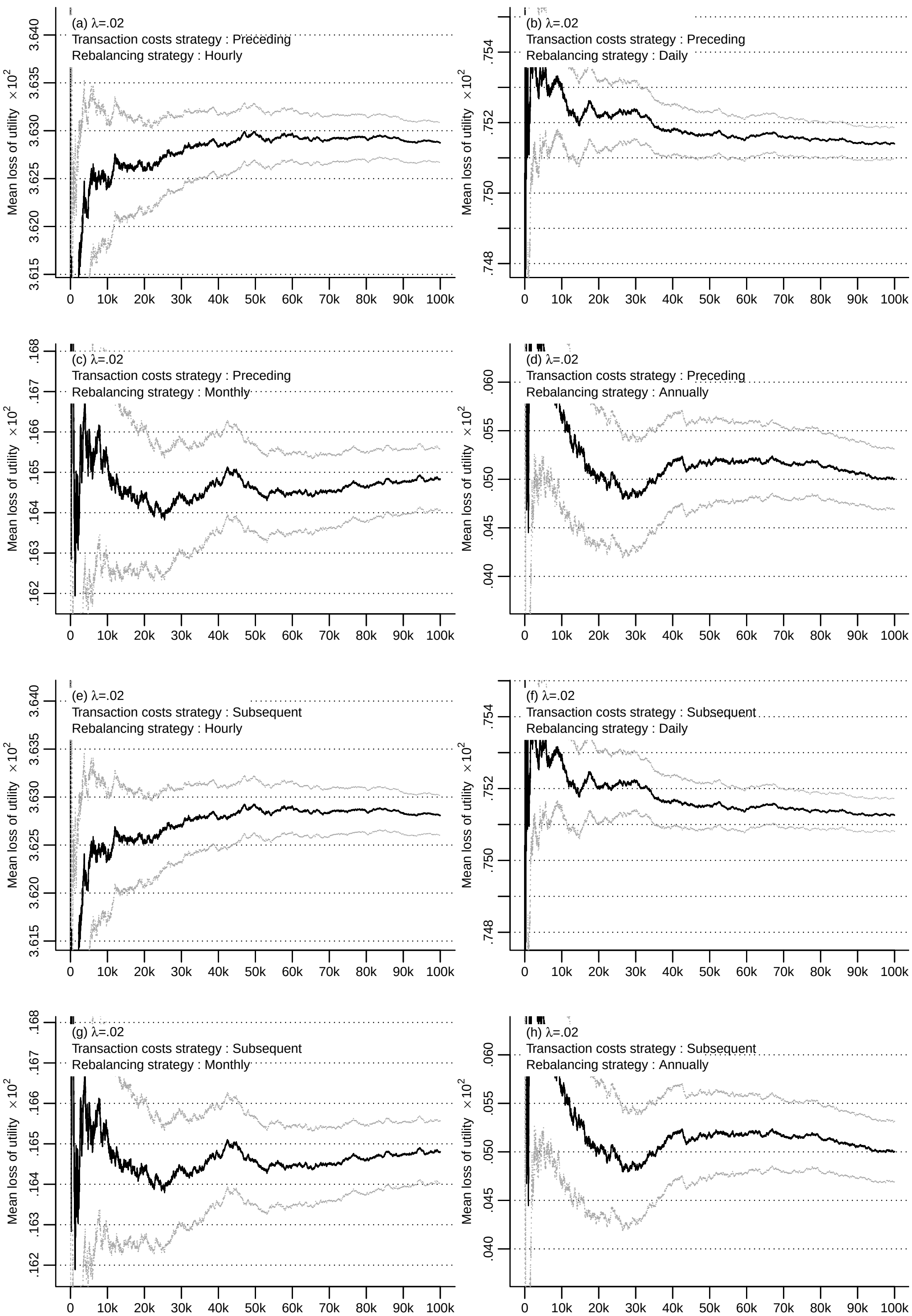


Figure A.2: The mean losses of utility with transaction cost proportion $\lambda = .02$. (a)-(d) preceding transaction costs and (e)-(h) subsequent transaction costs.

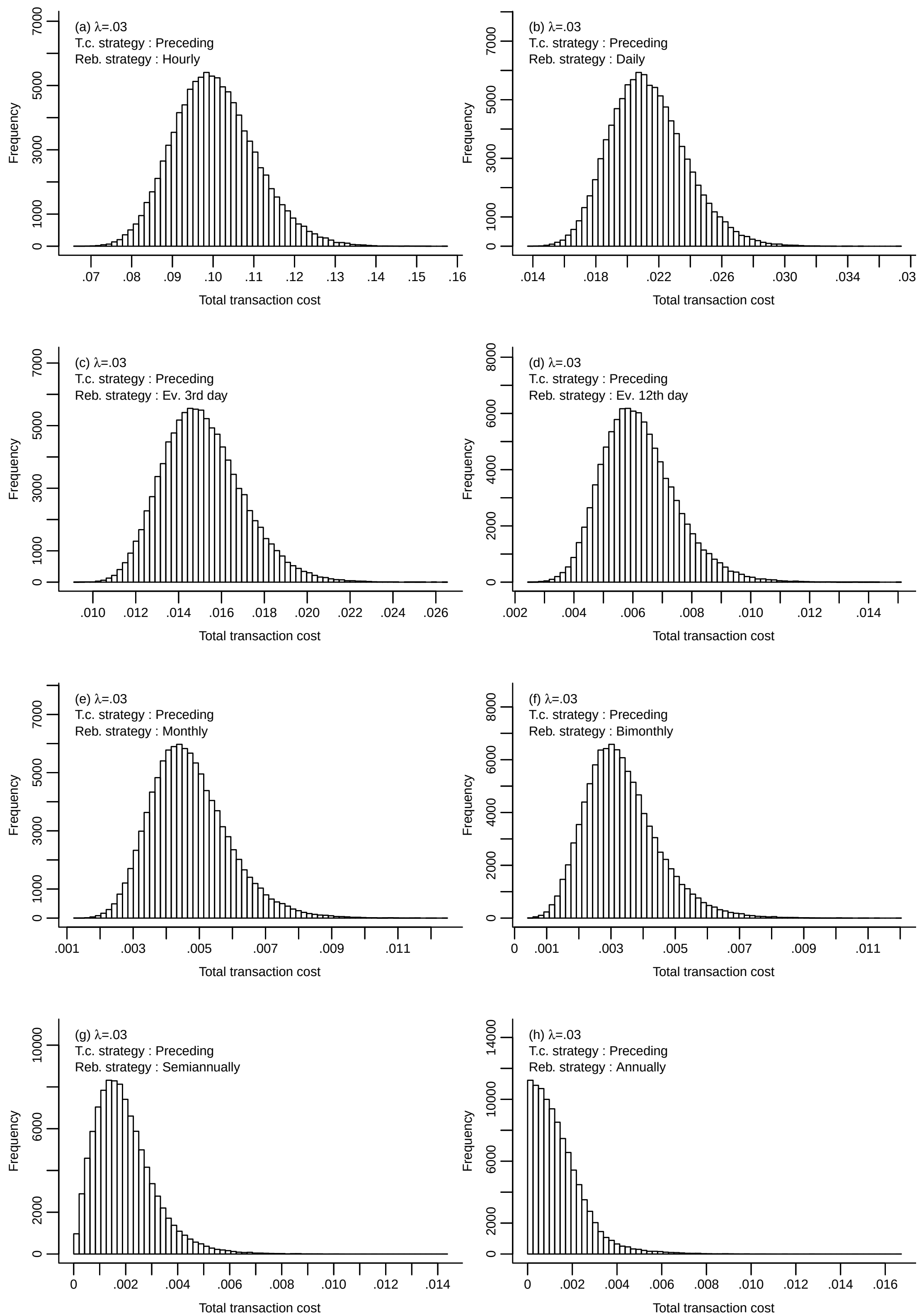


Figure A.3: Distributions of total transaction costs, $\lambda = .03$.

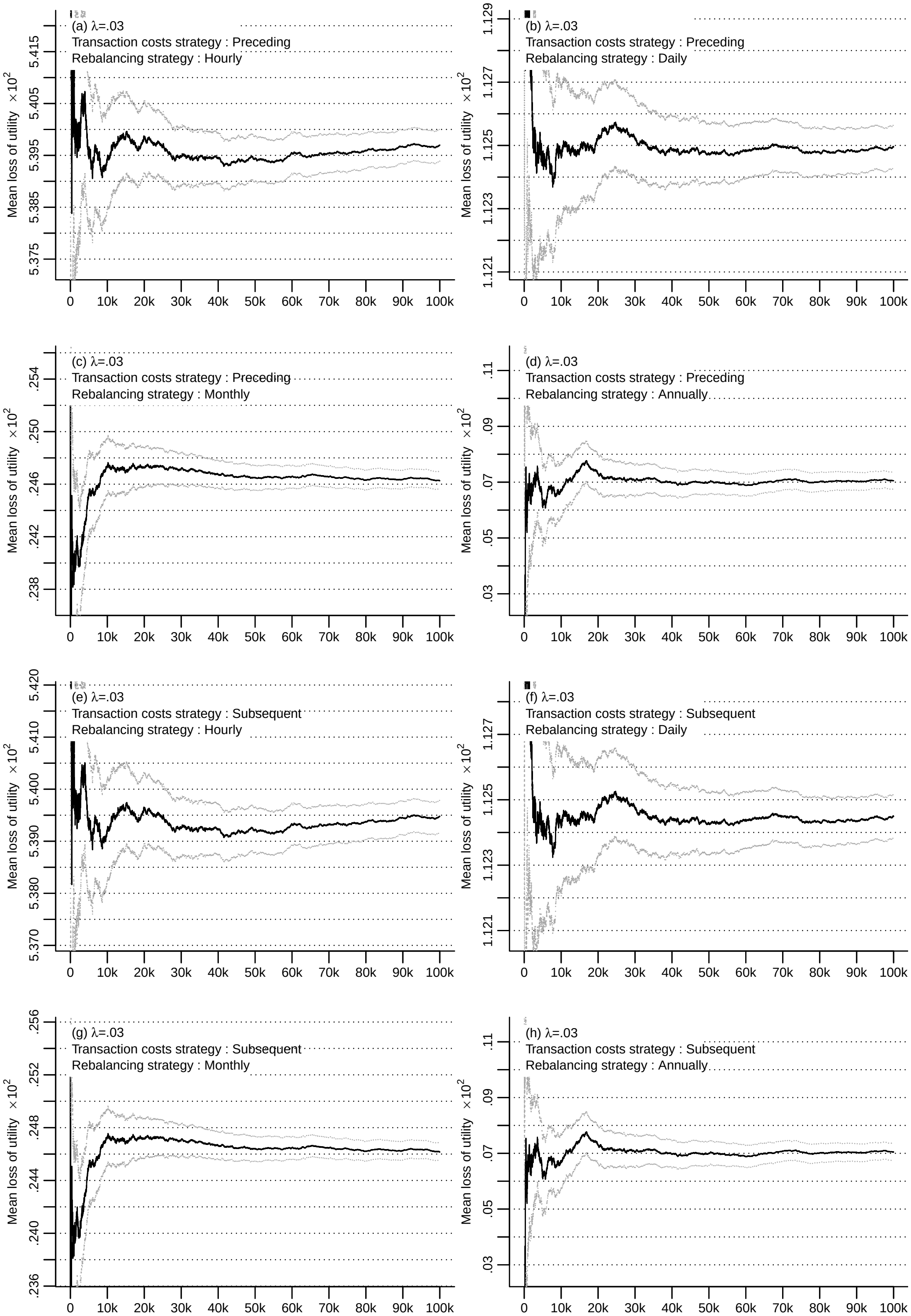


Figure A.4: The mean losses of utility with transaction cost proportion $\lambda = .03$. (a)-(d) preceding transaction costs and (e)-(h) subsequent transaction costs.

A.2 Simulation model IV

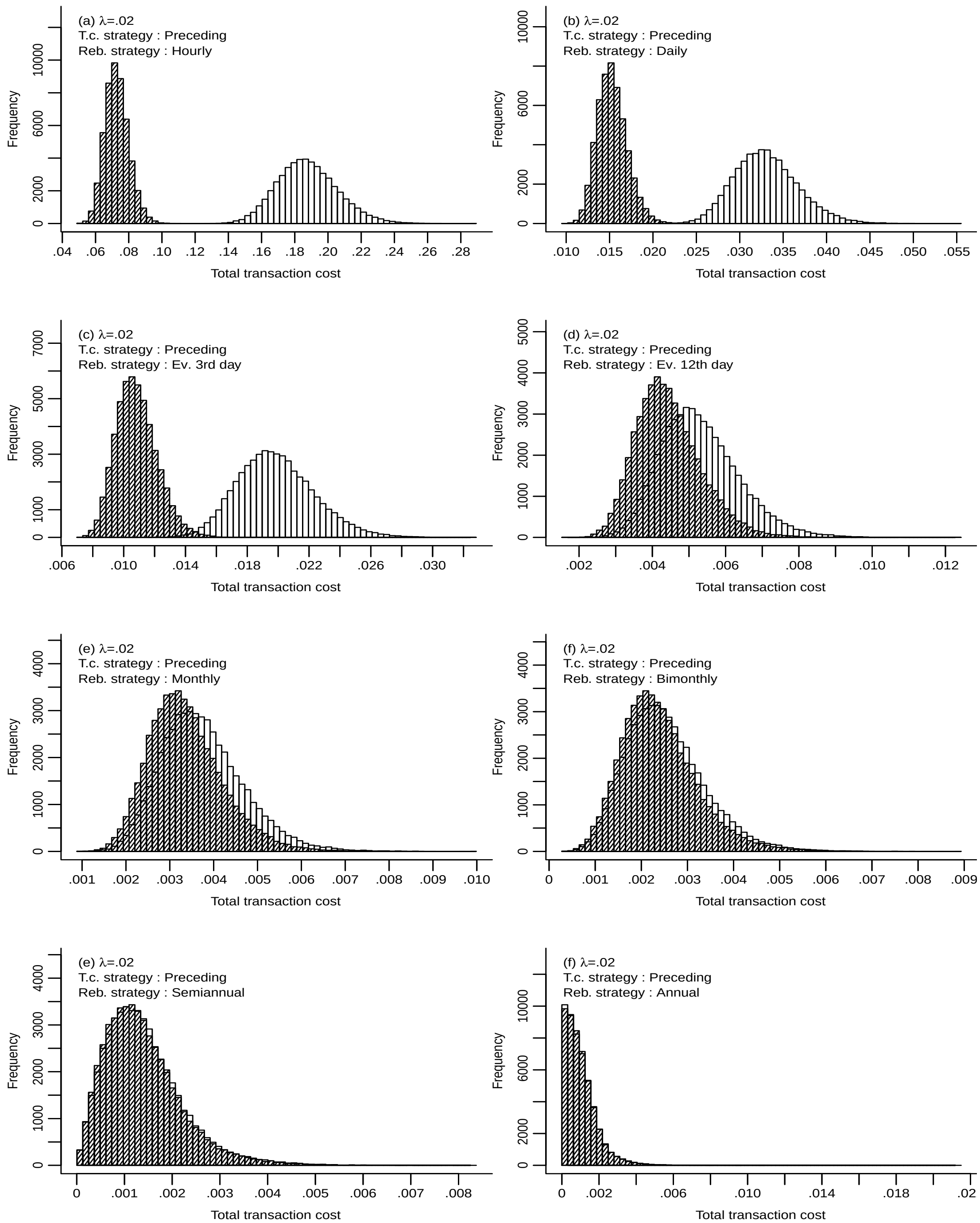


Figure A.5: Distributions of total transaction costs of stochastic volatility portfolios and constant volatility portfolios (shaded), $\lambda = .02$.

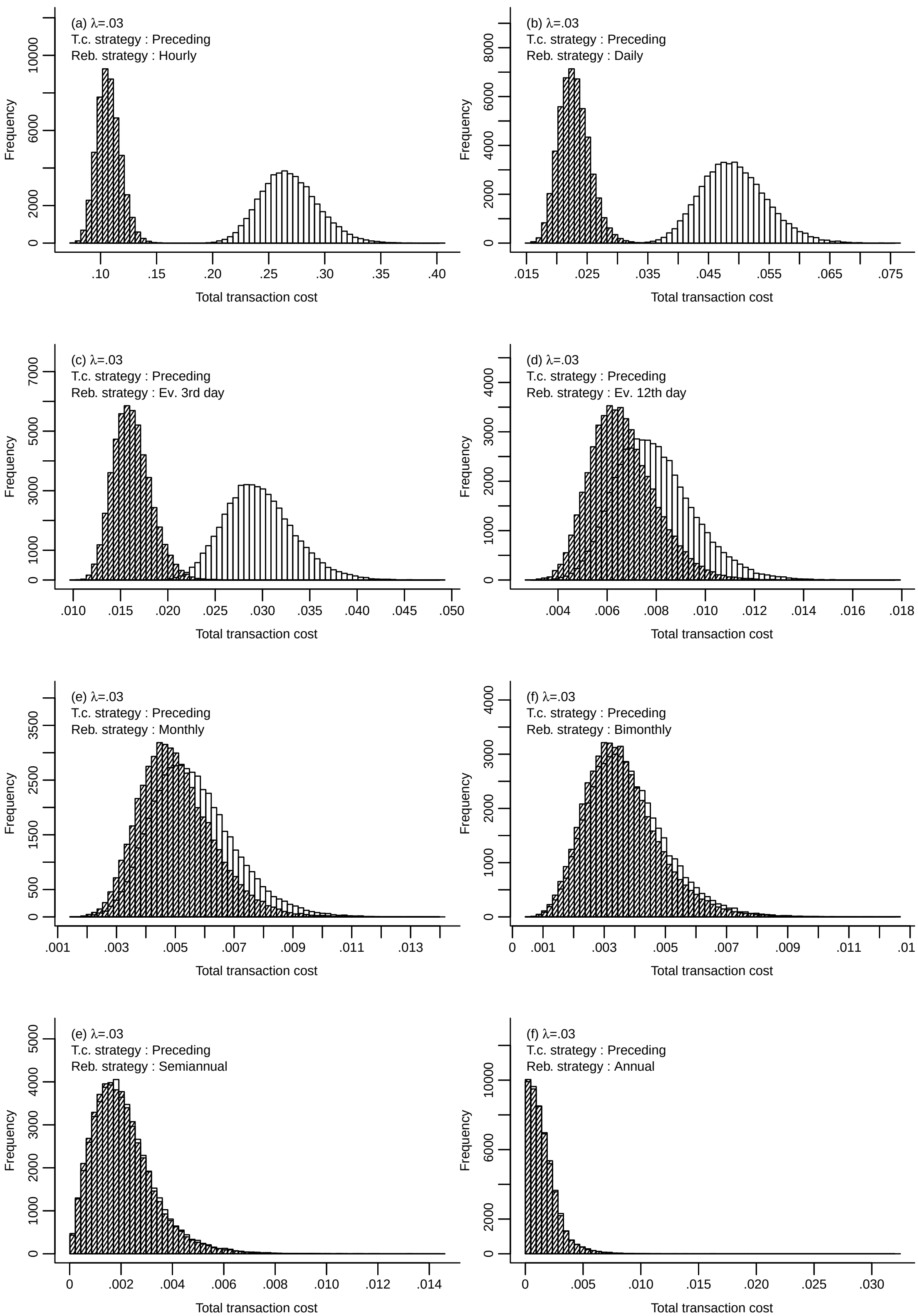


Figure A.6: Distributions of total transaction costs of stochastic volatility portfolios and constant volatility portfolios (shaded), $\lambda = .03$.

Appendix B

R source code

B.1 Support functions

```
##
# Master thesis
# Support functions
#

printex = function(table) {
  #
  # Convertes R tables to Latex table output.
  #
  rowNames = F
  if (!is.null(rownames(table))) { rowNames = T } else { rowNames = F }
  nRow = length(table[,1])
  nCol = length(table[1,])
  temp = ""
  for (i in 1:nRow) {
  if (rowNames) temp = paste(temp, rownames(table)[i], " & ", sep="")
  for (j in 1:nCol) {
    temp = paste(temp, table[i,j], sep="")
    if (j < nCol) temp = paste(temp, " & ", sep="")
  }
  temp = paste(temp, " \\", "\\", sep="")
  cat(temp, "\n", sep="")
  temp = ""
  }
}

is.zero = function(x) {
  #
  # Checks if elements of vector x == 0.
  #
  return(x == 0)
}

trimLast = function(x) {
  #
  # Removes last element of vector x.
  #
  n = length(x)
  return(x[-n])
```

```
}

strictlyIncreasing = function(x) {
  #
  # Checks if the elements of vector x is strictly increasing.
  #
  strictlyInc = T
  for (i in 2:length(x)) { strictlyInc = strictlyInc * ((x[i]/x[i-1])>1) }
  return(strictlyInc)
}

strictlyDecreasing = function(x) {
  #
  # Checks if the elements of vector x is strictly decreasing.
  #
  strictlyDec = T
  for (i in 2:length(x)) { strictlyDec = strictlyDec * ((x[i]/x[i-1])<1) }
  return(strictlyDec)
}

merge.list = function(x) {
  #
  # Merges list elements of list x.
  #
  n.x = length(x)
  merged = NULL
  for (k in 1:n.x) merged = c(merged,x[[k]])
  return(merged)
}

listDiff = function(listA,listB) {
  #
  # Computes the difference between lists.
  #
  listNames = names(listA)
  returnList = vector("list",length(listNames))
  names(returnList) = listNames
  for (k in 1:length(listNames)) { returnList[[k]] = listA[[k]] - listB[[k]] }
  return(returnList)
}

subsample = function(x,nSub=10000) {
  #
  # Downsamples vector x to length nSub.
  #
  inc = length(x) / nSub
  subsamples = 1:nSub*NA
  for (k in seq(0,nSub-2,2)) {
    actSubsample = x[(k*inc+1):((k+2)*inc)]
    minSubsample = min(actSubsample)
    maxSubsample = max(actSubsample)
    minIndex = match(minSubsample,actSubsample)
    maxIndex = match(maxSubsample,actSubsample)
    if (minIndex < maxIndex) { subsamples[k+1] = minSubsample; subsamples[k+2] = maxSubsample }
    else { subsamples[k+1] = maxSubsample; subsamples[k+2] = minSubsample }
  }
  indexList = seq(inc,length(x),inc)
  return(list(index=indexList,subsamples=subsamples))
}

niceplot = function(x,y,xTicks,yTicks,xLabels,yLabels,xTitle,yTitle,figsPerPage=4,caption=F,y.superscript=F,y.addCustom=0,nCol=1,multiPlot=F,newDev=T,plotHist=F,horizLines=F,downsample=F,nSub=10000,breaks,...) {
  #
```

```
  # Secures nice plots in Latex.
  #
  if (missing(y)) y = NULL
  if (caption) { yLength = c(20.18,6.33,4.05,2.48,1.88) }
  else { yLength = c(20.18,6.33,4.05,2.83,2.20) }
  if (newDev) {
    windows(11.9,yLength[figsPerPage])
    par(mfrow=c(1,nCol),cex.axis=.7,oma=c(0,0,0,0),mar=c(1.3,1.15,.55,0),mgp=c(2,.5,0),las=0,bty="l",lab=c(10,7,7))
    y.adj = y.addCustom
    if (y.superscript) y.adj = y.adj + .17
    if(!missing("xTitle") && missing("yTitle")) par(cex.lab=.7,mar=c(2.4,1.15,.55,0),mgp=c(1,.5,0))
    if(missing("xTitle") && !missing("yTitle")) par(cex.lab=.7,mar=c(1.3,2.15+y.adj,.55,0),mgp=c(1,.5,0))
    if(!missing("xTitle") && !missing("yTitle")) par(cex.lab=.7,mar=c(2.4,2.15+y.adj,.55,0),mgp=c(1,.5,0))
  }

  if (downsample) {
    if (is.null(y)) subsampleObject = subsample(x,nSub)
    else subsampleObject = subsample(y,nSub)
    x = subsampleObject$index
    y = subsampleObject$subsamples
  }

  if (!newDev && !multiPlot) lines(x,y,...)
  else {
    if (plotHist) {
      histObject = hist(x,breaks=breaks,freq=T,main="",axes=F,ann=F,...)
      box(bty="l")
    }
    else plot(x,y,type="l",xaxt="n",yaxt="n",ann=F,...)

    if (missing(xTicks)) xTicks = axis(1,labels=F)
    if (missing(xLabels)) { xLabels = sub("0[.]",".",format(xTicks,scientific=F)); xLabels = gsub(" ","",xLabels) }
    if (!any(xTicks==0) && min(xTicks)<=0 && max(xTicks)>=0) { xLabels = sort(c(0,xLabels)); xTicks = sort(c(0,xTicks)) }
    options(warn=-1)
    xLabels[as.numeric(xLabels)==0] = 0
    axis(1,xTicks,xLabels,padj=-.5)

    if (missing(yTicks)) yTicks = axis(2,labels=F)
    if (missing(yLabels)) yLabels = sub("0[.]",".",format(yTicks,scientific=F))
    if (!any(yTicks==0) && min(yTicks)<=0 && max(yTicks)>=0) { yLabels = sort(c(0,yLabels)); yTicks = sort(c(0,yTicks)) }
    yLabels[as.numeric(yLabels)==0] = 0
    axis(2,yTicks,yLabels,padj=-.1)
    if (horizLines) abline(h=yTicks,lty=3)

    if (!missing("xTitle")) title(xlab=xTitle,line=1.3)
    if (!missing("yTitle")) title(ylab=yTitle,line=1.55)

    if (plotHist) invisible(histObject)
  }
}

nicelines = function(x,y,...) { niceplot(x,y,newDev=F,...) }

nicehist = function(x,y,horizLines=F,breaks,...) {
  #
  # Secures nice histograms in Latex.
  #
  if (missing(breaks)) breaks = 10
```

```
  niceplot(x, plotHist=T, breaks=breaks, ...)
}

addHist = function(x, ...) {
  #
  # Superimposes a histogram on active plotting device.
  #
  histObject = hist(x, plot=F, ...)
  xLeft = trimLast(histObject$breaks)
  delta = xLeft[2] - xLeft[1]
  yBottom = trimLast(rep(0, length(xLeft)))
  xRight = trimLast(xLeft + delta)
  yTop = histObject$counts
  rect(xLeft, yBottom, xRight, yTop, ...)
  invisible(histObject)
}

nicelegend = function(...) {
  #
  # Makes nice plot legends.
  #
  legendObject = legend(..., plot=F)
  x.tune = 1/10
  x.left = legendObject$text$x - (legendObject$text$x-legendObject$rect$left)*x.
     tune
  y.bottom = legendObject$rect$top - legendObject$rect$h*.9
  x.right = x.left + legendObject$rect$w
  y.top = (legendObject$text$y + legendObject$rect$top) / 2
  rect(x.left, y.bottom, x.right, y.top, col="white", border="white")
  invisible(legend(...))
}

cumMean = function(x) {
  #
  # Calculates the cumulative mean along a vector.
  #
  cumulativeMean = cumsum(x) / 1:length(x)
  return(cumulativeMean)
}

cumSd = function(x) {
  #
  # Calculates the cumulative standard deviation along a vector.
  #
  nn = 1:length(x)
  cumulativeSd = sqrt((1/(nn-1)) * (cumsum(x^2)-cumsum(x)^2/nn))
}

colRange = function(x) {
  #
  # Calculates the ranges of the columns of matrix x.
  #
  n.col = ncol(x)
  ranges = matrix(NA, 2, n.col)
  for (k in 1:n.col) { ranges[,k] = range(x[,k]) }
  return(ranges)
}

colSds = function(X) {
  #
  # Computes the standard deviations along the columns of matrix X.
  #
  nCol = ncol(X)
  sds = 1:nCol*NA
  for (k in 1:nCol) { sds[k] = sd(X[,k]) }
```

```
  return(sds)
}

rowSds = function(X) {
  #
  # Computes the standard deviations along the rows of matrix X.
  #
  nRow = nrow(X)
  sds = 1:nRow*NA
  for (k in 1:nRow) { sds[k] = sd(X[k,]) }
  return(sds)
}

colCorrs = function(X,Y) {
  #
  # Computes the correlations between the columns of matrices X and Y,
  # respectively.
  #
  nCol = ncol(X)
  corrs = 1:nCol*NA
  for (k in 1:nCol) { corrs[k] = cor(X[,k],Y[,k]) }
  return(corrs)
}

colCumsums = function(X) {
  #
  # Calculates cumulative sums along columns of matrix X.
  #
  nRow = nrow(X)
  nCol = ncol(X)
  cumsums = matrix(NA,nRow,nCol)
  for (k in 1:nCol) { cumsums[,k] = cumsum(X[,k]) }
  return(cumsums)
}
```

B.2 Initialization and estimation

```
##
# Master Thesis
# Estimation of parameters
#

source("R/supportFunctions.R")
source("R/machinery_general.R")

graphics.off()

#
# Function declarations
#

logReturn = function(x) {
  #
  # Computes the log returns of a time series x.
  #
  n = length(x)
  x.up = x[2:n]
  x.low = x[1:(n-1)]
  logReturns = log(x.up/x.low)
  return(logReturns)
```

```
}

optimalControl = function(drift,volatility,rent,riskAversion) {
  #
  # Computes the optimal control following a power-type utility function.
  #
  control = pmax(pmin((drift-rent)/((1-riskAversion)*volatility^2),1),0)
  return(control)
}

#
# Loading OBX price and treasury bill data
#

obx = read.table("Datasett/OBX-finalSet.txt")
tbill = read.table("Datasett/tbill-finalSet.txt")

niceplot(obx[,2],yTitle="Price")
abline(v=3188,lty=3)
text(3188,min(obx[,2]),"Lehman brothers",adj=c(.05,-.4),cex=.7,srt=90)
savePlot("images/obx",type="eps")

#
# Estimation of the annual drift, volatility and rate of return
#

nTradingDays = 252
nTimePoints = 6048
obxLogReturns = logReturn(obx$price)
drift = nTradingDays * mean(obxLogReturns)
volatility = sqrt(nTradingDays) * sd(obxLogReturns)
tbillLogReturns = (1/252)*log(1+tbill$rent)
rent = 252*mean(tbillLogReturns)

niceplot(obxLogReturns,yTitle="Log return")
abline(h=0,lty=3)
abline(v=3187,lty=3)
text(3187,min(obxLogReturns),"Lehman brothers",adj=c(.05,-.4),cex=.7,srt=90)
savePlot("images/obxLogReturns",type="eps")

#
# Estimation of risk aversion
#

alpha = .01
wRisky = .5
wSure = 1 - wRisky
VaR = -(wRisky*quantile(obxLogReturns,alpha)+wSure*quantile(tbillLogReturns,
    alpha))
delta = 1/252
riskAve = riskAversion(drift,volatility,rent,VaR,delta,alpha)

#
# Estimation of optimal control
#

uStar = optimalControl(drift,volatility,rent,riskAve)

#
# Estimation of Heston parameters
#

shortTermVar = function(x,windowLength,delta=1) {
  #
  # Calculates volatility of short term window.
```

```
  #
  n = length(x)
  shortTermVar = 1:(n-windowLength+1) * NA
  for (k in 1:(n-windowLength+1)) { shortTermVar[k] = (1/delta) * var(x[k:(k+
      windowLength-1)]) }
  return(shortTermVar)
}

var.2 = shortTermVar(obxLogReturns,2,1/nTradingDays)
var.2.mean = mean(var.2)
var.2.sd = sd(var.2)
var.3 = shortTermVar(obxLogReturns,3,1/nTradingDays)
var.3.mean = mean(var.3)
var.3.sd = sd(var.3)
var.4 = shortTermVar(obxLogReturns,4,1/nTradingDays)
var.4.mean = mean(var.4)
var.4.sd = sd(var.4)
var.5 = shortTermVar(obxLogReturns,5,1/nTradingDays)
var.5.mean = mean(var.5)
var.5.sd = sd(var.5)
var.6 = shortTermVar(obxLogReturns,6,1/nTradingDays)
var.6.mean = mean(var.6)
var.6.sd = sd(var.6)
var.7 = shortTermVar(obxLogReturns,7,1/nTradingDays)
var.7.mean = mean(var.7)
var.7.sd = sd(var.7)

delta = 1 / nTimePoints

# Window length : 2
n = length(var.2)
var.2.up = var.2[2:n]
var.2.down = var.2[1:(n-1)]
y.2 = (var.2.up - var.2.down) / sqrt(var.2.down)
x1.2 = 1 / sqrt(var.2.down)
x2.2 = sqrt(var.2.down)
linreg.2 = lm(y.2 ~ x1.2 + x2.2 - 1)
summary(linreg.2)
beta1 = linreg.2$coeff[1]
beta2 = linreg.2$coeff[2]
var.long.2 = -beta1 / beta2
reversionRate.2 = -beta2 / delta
var.init.2 = var.long.2
var.inc.2 = diff(var.2)
volOfVol.2 = sd(var.inc.2)
correlation.2 = cor(obxLogReturns[1:length(var.inc.2)],var.inc.2)

# Window length : 3
n = length(var.3)
var.3.up = var.3[2:n]
var.3.down = var.3[1:(n-1)]
y.3 = (var.3.up - var.3.down) / sqrt(var.3.down)
x1.3 = 1 / sqrt(var.3.down)
x2.3 = sqrt(var.3.down)
linreg.3 = lm(y.3 ~ x1.3 + x2.3 - 1)
beta1 = linreg.3$coeff[1]
beta2 = linreg.3$coeff[2]
var.long.3 = -beta1 / beta2
reversionRate.3 = -beta2 / delta
var.init.3 = var.long.3
var.inc.3 = diff(var.3)
volOfVol.3 = sd(var.inc.3)
correlation.3 = cor(obxLogReturns[1:length(var.inc.3)],var.inc.3)

# Window length : 4
```

```
n = length(var.4)
var.4.up = var.4[2:n]
var.4.down = var.4[1:(n-1)]
y.4 = (var.4.up - var.4.down) / sqrt(var.4.down)
x1.4 = 1 / sqrt(var.4.down)
x2.4 = sqrt(var.4.down)
linreg.4 = lm(y.4 ~ x1.4 + x2.4 - 1)
beta1 = linreg.4$coeff[1]
beta2 = linreg.4$coeff[2]
var.long.4 = -beta1 / beta2
reversionRate.4 = -beta2 / delta
var.init.4 = var.long.4
var.inc.4 = diff(var.4)
volOfVol.4 = sd(var.inc.4)
correlation.4 = cor(obxLogReturns[1:length(var.inc.4)],var.inc.4)

# Window length : 5
n = length(var.5)
var.5.up = var.5[2:n]
var.5.down = var.5[1:(n-1)]
y.5 = (var.5.up - var.5.down) / sqrt(var.5.down)
x1.5 = 1 / sqrt(var.5.down)
x2.5 = sqrt(var.5.down)
linreg.5 = lm(y.5 ~ x1.5 + x2.5 - 1)
beta1 = linreg.5$coeff[1]
beta2 = linreg.5$coeff[2]
var.long.5 = -beta1 / beta2
reversionRate.5 = -beta2 / delta
var.init.5 = var.long.5
var.inc.5 = diff(var.5)
volOfVol.5 = sd(var.inc.5)
correlation.5 = cor(obxLogReturns[1:length(var.inc.5)],var.inc.5)

# Window length : 6
n = length(var.6)
var.6.up = var.6[2:n]
var.6.down = var.6[1:(n-1)]
y.6 = (var.6.up - var.6.down) / sqrt(var.6.down)
x1.6 = 1 / sqrt(var.6.down)
x2.6 = sqrt(var.6.down)
linreg.6 = lm(y.6 ~ x1.6 + x2.6 - 1)
beta1 = linreg.6$coeff[1]
beta2 = linreg.6$coeff[2]
var.long.6 = -beta1 / beta2
reversionRate.6 = -beta2 / delta
var.init.6 = var.long.6
var.inc.6 = diff(var.6)
volOfVol.6 = sd(var.inc.6)
correlation.6 = cor(obxLogReturns[1:length(var.inc.6)],var.inc.6)

# Window length : 7
n = length(var.7)
var.7.up = var.7[2:n]
var.7.down = var.7[1:(n-1)]
y.7 = (var.7.up - var.7.down) / sqrt(var.7.down)
x1.7 = 1 / sqrt(var.7.down)
x2.7 = sqrt(var.7.down)
linreg.7 = lm(y.7 ~ x1.7 + x2.7 - 1)
beta1 = linreg.7$coeff[1]
beta2 = linreg.7$coeff[2]
var.long.7 = -beta1 / beta2
reversionRate.7 = -beta2 / delta
var.init.7 = var.long.7
var.inc.7 = diff(var.7)
volOfVol.7 = sd(var.inc.7)
```

```
correlation.7 = cor(obxLogReturns[1:length(var.inc.7)],var.inc.7)

# Plotting and saving
y.range = range(var.2)
y.ticks = c(0,1,2,3,4)
niceplot(var.2,yTicks=y.ticks,yTitle="Volatility",figsPerPage=5,ylim=y.range)
nicelegend("topleft","(a) Window length = 2",bty="n",bg="white",cex=.7)
savePlot("images/volatility_winLength2",type="eps")
niceplot(var.7,yTicks=y.ticks,yTitle="Volatility",figsPerPage=5,ylim=y.range)
nicelegend("topleft","(b) Window length = 7",bty="n",bg="white",cex=.7)
savePlot("images/volatility_winLength7",type="eps")

# Construction of output table
tab = matrix(NA,6,6)

tab[1,] = c(2,var.init.2,reversionRate.2,var.long.2,volOfVol.2,correlation.2)
tab[2,] = c(3,var.init.3,reversionRate.3,var.long.3,volOfVol.3,correlation.3)
tab[3,] = c(4,var.init.4,reversionRate.4,var.long.4,volOfVol.4,correlation.4)
tab[4,] = c(5,var.init.5,reversionRate.5,var.long.5,volOfVol.5,correlation.5)
tab[5,] = c(6,var.init.6,reversionRate.6,var.long.6,volOfVol.6,correlation.6)
tab[6,] = c(7,var.init.7,reversionRate.7,var.long.7,volOfVol.7,correlation.7)

colNames = c("l","var.init","revRate","var.long","volOfVol","Correlation")
colnames(tab) = colNames
transformation = cbind(rep(1,6),rep(1e2,6),rep(1,6),rep(1e2,6),rep(1,6),rep(1e2,6))
tab = round(tab*transformation,4)
as.data.frame(tab)
for (k in 1:6) {
  tab[k,2] = paste(tab[k,2],"\\e{\\text-2}",sep="")
  tab[k,4] = paste(tab[k,4],"\\e{\\text-2}",sep="")
  tab[k,6] = paste(tab[k,6],"\\e{\\text-2}",sep="")
}
printex(tab)

# Plotting differences of 5-day variances
niceplot(diff(sqrt(var.5)),yTitle="Change of volatility")
savePlot("images/5dayVol_diff",type="eps")
```

```
##
# Master Thesis
# Initilization of parameters
#

# Basic parameters
initWealth       = 1
nTradingDays     = 252
nDailyIncrements = 24     # Hourly updates of portfolio value
nDailyRebs       = 12/252 # Monthly-rebalancing strategy
drift            = .0657
volatility       = .2537
rent             = .0449
riskAversion     = .5255
uStar            = optimalControl(drift,volatility,rent,riskAversion)

# Additional transaction cost parameters
costProp         = .03

# Additional stochastic volatility parameters
var.init         = 6.7456e-2
reversionRate    = 320.1192
var.long         = 6.7456e-2
volOfVol         = .0590
correlation      = 2.6706e-2
```

```
uStar.constVol    = optimalControl(drift,sqrt(var.long),rent,riskAversion)

# Setting up simulation model I input parameters
paramSet = c(initWealth,nTradingDays,nDailyIncrements,nDailyRebs,drift,
    volatility,rent,riskAversion,uStar)
names(paramSet) = c("initWealth","nTradingDays","nDailyIncrements","nDailyRebs
    ","drift","volatility","rent","riskAversion","uStar")

# Setting up simulation model II and III input parameters
paramSet.transCost = c(paramSet,costProp)
names(paramSet.transCost) = c("initWealth","nTradingDays","nDailyIncrements","
    nDailyRebs","drift","volatility","rent","riskAversion","uStar","costProp")

# Setting up simulation model IV input parameters
paramSet.constVol = c(initWealth,nTradingDays,nDailyIncrements,nDailyRebs,drift,
    sqrt(var.long),rent,riskAversion,uStar.constVol,costProp)
paramSet.stochVol = c(initWealth,nTradingDays,nDailyIncrements,nDailyRebs,drift,
    rent,riskAversion,costProp,var.init,reversionRate,var.long,volOfVol,
    correlation)
nParam.stochVol = length(paramSet.stochVol)
names(paramSet.stochVol) = c("initWealth","nTradingDays","nDailyIncrements","
    nDailyRebs","drift","rent","riskAversion","costProp","var.init","
    reversionRate","var.long","volOfVol","correlation")

# Calculating number of time points and equidistant time increment delta
nTimePoints = nTradingDays * nDailyIncrements
delta = 1 / nTimePoints
```

B.3 General simulation machinery

```
##
# Master Thesis
# General machinery
#

logReturn = function(x) {
  #
  # Computes the log returns of a time series x.
  #
  n = length(x)
  xUp = x[2:n]
  xLow = x[1:(n-1)]
  logReturns = log(xUp/xLow)
  return(logReturns)
}

riskAversion = function(drift,volatility,rent,VaR,delta,alpha) {
  #
  # Computes the risk aversion parameter of a power-type utility function
  # through Value at Risk.
  #
  qAlpha = qnorm(alpha)
  lengthVol = length(volatility)
  if (lengthVol==1) solution = 1:2*NA
  else solution = matrix(NA,lengthVol,2)
  a = drift - rent + qAlpha*volatility/sqrt(delta)
  b = 2*volatility^2*(VaR/delta+rent)
  if (lengthVol==1) {
    solution[1] = 1 + (drift-rent)*(a+sqrt(a^2+b))/b
    solution[2] = 1 + (drift-rent)*(a-sqrt(a^2+b))/b
```

```
  }
  else {
    solution[,1] = 1 + (drift-rent)*(a+sqrt(a^2+b))/b
    solution[,2] = 1 + (drift-rent)*(a-sqrt(a^2+b))/b
  }
  return(solution)
}

expectedWealth = function(initWealth, drift, rent, uStar, tp) {
  #
  # Computes the expected wealth.
  #
  return(initWealth*exp((drift*uStar + rent*(1-uStar))*tp))
}

stDevWealth = function(initWealth, drift, volatility, rent, uStar, tp) {
  #
  # Computes the expected standard deviation of the wealth.
  #
  expecWealth = expectedWealth(initWealth, drift, rent, uStar, tp)
  return(sqrt(expecWealth^2 * (exp(volatility^2*uStar^2*tp) - 1)))
}

expectedLogReturn = function(drift, volatility, rent, uStar, delta) {
  #
  # Computes the expected log return.
  #
  return((drift*uStar + rent*(1-uStar) - .5*volatility^2*uStar^2)*delta)
}

stDevLogReturn = function(volatility, uStar, delta) {
  #
  # Computes the expected standard deviation of the log returns.
  #
  return(sqrt(volatility^2*uStar^2*delta))
}

simRiskyAsset = function(initValue, drift, volatility, BM) {
  #
  # Calculates risky asset values according to Brownian motion BM. Uses
  # Euler-Maruyama approximation.
  #
  nTimePoints = length(BM)
  delta = 1 / nTimePoints
  inc = c(0, diff(BM))
  simValue = initWealth * (1 + drift*delta + volatility*inc[1])
  for (i in 2:nTimePoints) { simValue[i] = simValue[i-1] * (1 + drift*delta + volatility*inc[i]) }
  return(simValue)
}

riskFreeAsset = function(initValue, rent, nTimePoints) {
  #
  # Calculates risk-free asset values using Euler approximation.
  #
  delta = 1 / nTimePoints
  value = initValue * (1 + rent*delta)
  for (i in 2:nTimePoints) { value[i] = value[i-1] * (1 + rent*delta) }
  return(value)
}

expectedLogReturn = function(drift, volatility, rent, uStar, tp) {
  #
  # Computes the expected log return from time 0 to time tp.
  #
```

```
  return((drift*uStar + rent*(1-uStar) - .5*volatility^2*uStar^2)*tp)
}

stDevLogReturn = function(volatility,uStar,tp) {
  #
  # Computes the ex ante standard deviation of the log returns from time 0
  # to time tp.
  #
  return(volatility*uStar*sqrt(tp))
}

exAnteSharpeRatio = function(drift,volatility,rent,uStar,tp) {
  #
  # Computes the ex ante, that is the expected Sharpe ratio of the
  # portfolio.
  #
  expecLogReturn = expectedLogReturn(drift,volatility,rent,uStar,tp)
  sdLogReturn = stDevLogReturn(volatility,uStar,tp)
  return((expecLogReturn - rent) / sdLogReturn)
}

sharpeRatio = function(terminalWealth,rent,sdLogReturn,nTimePoints) {
  #
  # Computes the ex post Sharpe ratio given the terminal wealth of a time
  # series of wealths, the benchmark risk free rate of return and the
  # standard deviation of the log returns of the wealth series.
  #
  return((log(terminalWealth) - rent) / (nTimePoints*sdLogReturn))
}

kill = function() {
  #
  # Removes redundant doSMP workers.
  #
  rmSessions(all.names=T)
}

optimalControl = function(drift,volatility,rent,riskAversion) {
  #
  # Computes the optimal control following a power-type utility function.
  #
  control = pmax(pmin((drift-rent)/((1-riskAversion)*volatility^2),1),0)
  return(control)
}

utility = function(x,param,type="power") {
  #
  # Computes the power-type utility of a wealth x.
  #
  if (type=="power") utility = x^param
  return(utility)
}

brownianIncrement = function(nSims,nTimePoints,thread=1) {
  #
  # Generates nSims rows of nTimePoints Brownian increments each increment
  # with variance 1 / nTimePoints.
  #
  delta = 1 / nTimePoints
  N = nSims * nTimePoints
  brownianMatrix = t(matrix(rnorm(N,0,sqrt(delta)),nTimePoints,nSims))
  return(brownianMatrix)
}

multiSim = function(nSims,nCores,func,paramSet) {
```

```
  #
  # Splits nSims simulations into nSims/nCores simulations which are
  # simulated on nSims/nCores processor cores. The nSims/nCores subsets
  # are then put together and returned.
  #
  if (nSims/nCores != round(nSims/nCores)) stop("Number of simulations is not a multiple of number of cores.")
  cat("Doing",format(nSims,scientific=F),"simulation runs on",nCores,"core(s)...\n")
  flush.console()
  cat("Parameter set :",paramSet,"\n")
  flush.console()
  timeStart = proc.time()[3][[1]]
  workers = startWorkers(nCores)
  registerDoSMP(workers)
  multiSimObject = foreach(j=1:nCores) %dopar% func(nSims/nCores,paramSet)
  stopWorkers(workers)
  timeElapsed = proc.time()[3][[1]] - timeStart
  cat(format(nSims,scientific=F),"simulation runs completed in",timeElapsed," seconds.\n")
  if (is.list(multiSimObject[[1]])) {
    multiSimNames = names(multiSimObject[[1]])
    returnObject = vector("list",length(multiSimNames))
    names(returnObject) = multiSimNames
    for (k in 1:length(multiSimNames)) { returnObject[[k]] = c(sapply(multiSimObject,get,x=multiSimNames[k])) }
  }
  else if (is.matrix(multiSimObject[[1]])) { returnObject = abind(multiSimObject,along=1) }
  return(returnObject)
}

distribute = function(nSims,nCores,func,paramSets) {
  #
  # Apply-style wrapper function for simulating multiple parameter sets.
  #
  nParamSets = nrow(paramSets)
  cat("Simulating",nParamSets," parameter sets...\n")
  flush.console()
  returnList = list()
  timeStart = proc.time()[3][[1]]
  for (k in 1:nParamSets) { returnList[[k]] = multiSim(nSims,nCores,func,paramSets[k,]) }
  timeElapsed = proc.time()[3][[1]] - timeStart
  cat(nParamSets," parameter sets completed in",timeElapsed," seconds.\n")
  return(returnList)
}
```

B.4 Simulation model I

B.4.1 Simulation machinery

```
##
# Master Thesis
# Simulation model I
# Simulation algorithm
#

```

```r
simPortfolio = function(nSims,paramSet,brownianDataSet=NULL) {
  #
  # Simulates nSims portfolios following the 9 parameter values of paramSet
  # and returns terminal utilities of theoretical and simulated wealth.
  #

  logReturn = function(x) {
    #
    # Computes the log returns of a time series x.
    #
    n = length(x)
    xUp = x[2:n]
    xLow = x[1:(n-1)]
    logReturns = log(xUp/xLow)
    return(logReturns)
  }

  brownianIncrement = function(n,delta) {
    #
    # Simulates random series of n brownian increments with variance delta.
    #
    return(rnorm(n,0,sqrt(delta)))
  }

  #
  # Assigning variables.
  #
  nParams = length(paramSet)
  if (nParams != 9) stop(paste("Number of input parameters equals ",nParams,".
      Must equal 9.",sep=""))
  varNames = c("initWealth","nTradingDays","nDailyIncrements","nDailyRebs","
      drift","volatility","rent","riskAversion","uStar")
  for (j in 1:9) { assign(varNames[j],paramSet[j]) }

  #
  # Initializing the simulation structure.
  #
  simIndex = 1:nSims
  nTimePoints = nTradingDays * nDailyIncrements
  lastIndex = nTimePoints
  delta = 1 / nTimePoints
  timePoints = seq(delta,1,delta)
  nRebDelay = nDailyIncrements / nDailyRebs
  rebIndex = seq(nRebDelay,nTimePoints,nRebDelay)
  rebIndex.length = length(rebIndex)
  days = seq(delta*nTradingDays,nTradingDays,delta*nTradingDays)
  rebDays = days[rebIndex]
  ones = rep(1,nRebDelay)

  # Common structure
  simWealth = 1:nTimePoints * NA

  # Setting start time
  timeStart = proc.time()[3][[1]]

  #
  # If nSims = 1, the full simulation scheme is applied. If nSims > 1, to
  # gain speed, the compact form will be applied.
  #

  if (nSims == 1) {

    # Intialization of time series
    simWealth.risky = 1:nTimePoints * NA
    simWealth.riskfree = 1:nTimePoints * NA
```

```
    propInRisky = 1:nTimePoints * NA
    propInRiskfree = 1:nTimePoints * NA

    # Brownian increments and motion
    if (!is.null(brownianDataSet)) load(brownianDataSet)
    else inc = brownianIncrement(nTimePoints,delta)
    BM = cumsum(inc)

    # Calculation of theoretical wealth and relevant statistics
    thWealth = initWealth*exp((drift*uStar+rent*(1-uStar)-.5*volatility^2*uStar^2)*timePoints+volatility*uStar*BM)
    thTermWealth = tail(thWealth,1)
    sdThWealth = sd(thWealth)
    thLogReturn = logReturn(c(initWealth,thWealth))
    sdThLogReturn = sd(thLogReturn)

    # Initial time points to be simulated
    activeIndices = 1:nRebDelay
    rebPoint = tail(activeIndices,1)

    # Initial simulations
    simWealth.risky[activeIndices] = uStar*initWealth*cumprod(1+drift*delta+volatility*inc[activeIndices])
    simWealth.riskfree[activeIndices] = (1-uStar)*initWealth*cumprod((1+rent*delta)*ones)
    simWealth.risky.prime = simWealth.risky[rebPoint]
    simWealth.riskfree.prime = simWealth.riskfree[rebPoint]
    transQuantity = (1-uStar)*simWealth.risky.prime - uStar*simWealth.riskfree.prime
    simWealth.risky[rebPoint] = simWealth.risky.prime - transQuantity
    simWealth.riskfree[rebPoint] = simWealth.riskfree.prime + transQuantity
    simWealth[activeIndices] = simWealth.risky[activeIndices] + simWealth.riskfree[activeIndices]
    propInRisky[activeIndices] = simWealth.risky[activeIndices] / simWealth[activeIndices]
    propInRiskfree[activeIndices] = simWealth.riskfree[activeIndices] / simWealth[activeIndices]

    # Remainder of simulations
    for (j in rebIndex[-rebIndex.length] + 1) {

      activeIndices = j:(j+nRebDelay-1)
      rebPoint = tail(activeIndices,1)

      simWealth.risky[activeIndices] = uStar*simWealth[j-1]*cumprod(1+drift*delta+volatility*inc[activeIndices])
      simWealth.riskfree[activeIndices] = (1-uStar)*simWealth[j-1]*cumprod((1+rent*delta)*ones)
      simWealth.risky.prime = simWealth.risky[rebPoint]
      simWealth.riskfree.prime = simWealth.riskfree[rebPoint]
      transQuantity = (1-uStar)*simWealth.risky.prime - uStar*simWealth.riskfree.prime
      simWealth.risky[rebPoint] = simWealth.risky.prime - transQuantity
      simWealth.riskfree[rebPoint] = simWealth.riskfree.prime + transQuantity
      simWealth[activeIndices] = simWealth.risky[activeIndices] + simWealth.riskfree[activeIndices]
      propInRisky[activeIndices] = simWealth.risky[activeIndices] / simWealth[activeIndices]
      propInRiskfree[activeIndices] = simWealth.riskfree[activeIndices] / simWealth[activeIndices]
    }

    # Calculation of relevant statistics
    sdSimWealth = sd(simWealth)
    simTermWealth = tail(simWealth,1)
```

```
    simLogReturn = logReturn(c(initWealth,simWealth))
    sdSimLogReturn = sd(simLogReturn)
  }

  # nSims > 1 : Compact (rapid) simulation scheme
  else {

    # Initialization of vectors of relevant statistics
    sdThWealth = simIndex * NA
    thTermWealth = simIndex * NA
    sdThLogReturn = simIndex * NA
    sdSimWealth = simIndex * NA
    simTermWealth = simIndex * NA
    sdSimLogReturn = simIndex * NA

    # Doing nSims simulation runs
    for (k in 1:nSims) {

      # Brownian increments and motion
      inc = brownianIncrement(nTimePoints,delta)
      BM = cumsum(inc)

      # Calucaltion of theoretical wealth and relevant statistics
      thWealth = initWealth*exp((drift*uStar+rent*(1-uStar)-.5*volatility^2*
          uStar^2)*timePoints+volatility*uStar*BM)
      thTermWealth[k] = tail(thWealth,1)
      sdThWealth[k] = sd(thWealth)
      thLogReturn = logReturn(c(initWealth,thWealth))
      sdThLogReturn[k] = sd(thLogReturn)

      # Initial simulation time points
      activeIndices = 1:nRebDelay
      rebPoint = tail(activeIndices,1)

      # Initial simulations of wealth
      simWealth[activeIndices] = uStar*initWealth*cumprod(1+drift*delta+
          volatility*inc[activeIndices]) + (1-uStar)*initWealth*cumprod((1+rent*
          delta)*ones)

      # The rest of the wealty simulations
      for (j in rebIndex[-rebIndex.length] + 1) {

        activeIndices = j:(j+nRebDelay-1)
        rebPoint = tail(activeIndices,1)

        simWealth[activeIndices] = uStar*simWealth[j-1]*cumprod(1+drift*delta+
            volatility*inc[activeIndices]) + (1-uStar)*simWealth[j-1]*cumprod
            ((1+rent*delta)*ones)
      }

      # Calculation of relevant statistics of last simulation run
      sdSimWealth[k] = sd(simWealth)
      simTermWealth[k] = simWealth[lastIndex]
      simLogReturn = logReturn(c(initWealth,simWealth))
      sdSimLogReturn[k] = sd(simLogReturn)
    }
  }

  # Calculation of total simulation time
  timeElapsed = proc.time()[3][[1]] - timeStart
  cat(nSims," simulation(s) completed in",timeElapsed," seconds.\n")
  flush.console()

  # Construction of the list of data to be returned from the function.
  if (nSims == 1) {
```

```
    returnList = list(days,rebDays,inc,BM,propInRisky,thWealth,sdThWealth,thTermWealth,thLogReturn,sdThLogReturn,simWealth,sdSimWealth,simTermWealth,simLogReturn,sdSimLogReturn)
    names(returnList) = c("days","rebDays","brownianIncrements","brownianMotion","propInRisky","thWealth","sdThWealth","thTermWealth","thLogReturn","sdThLogReturn","simWealth","sdSimWealth","simTermWealth","simLogReturn","sdSimLogReturn")
  }
  else {
    returnList = list(thTermWealth,sdThWealth,sdThLogReturn,simTermWealth,sdSimWealth,sdSimLogReturn)
    names(returnList) = c("thTermWealth","sdThWealth","sdThLogReturn","simTermWealth","sdSimWealth","sdSimLogReturn")
  }

  return(returnList)
}
```

B.4.2 Execution

```
##
# Master Thesis
# Simulation model I
# Simulation
#

require(doSMP)
source("R/supportFunctions.R")
source("R/listArithmetic.R")
source("R/machinery_general.R")
source("R/machinery_basic.R")
source("R/initParameters.R")

delta = 1 / (nTradingDays*nDailyIncrements)
alpha = .05 # Significance level
qAlphaHalf = qnorm(1-alpha/2) # 1-alpha/2 percentile of std. norm. dist.

#
# Plot and analysis of one simulation test run (nSims=1)
#

nSims = 1
simObject = simPortfolio(nSims,paramSet)
save(simObject,file="Datasett/testRun.RData")

propInRisky = simObject$proportion.in.risky
days = simObject$days
rebDays = c(simObject$rebDays,252)
thWealth = simObject$thWealth
simWealth = simObject$simWealth
utilityThWealth = utility(thWealth,riskAversion)
utilitySimWealth = utility(simWealth,riskAversion)
diffUtility = utilityThWealth - utilitySimWealth

# Plotting test run
xTicks = c(0,rebDays)
xTitle = "Trading days"
yTitle = "Utility"
niceplot(c(0,days),c(initWealth,utilityThWealth),xTicks,xTitle=xTitle,yTitle=yTitle,horizLines=T,col="red")
lines(c(0,days),c(initWealth,utilitySimWealth),col="dodgerblue")
```

```
abline(v=rebDays, lty=3)
savePlot("images/testrun",type="eps")
niceplot(c(0,days),c(0,diffUtility),xTicks,xTitle=xTitle,yTitle=yTitle,
    horizLines=T)
abline(v=rebDays, lty=3)
savePlot("images/testrun_diff",type="eps")

# Plotting proportion in risky
yTitle = "Proportion of wealth in risky asset"
niceplot(c(0,days),c(uStar,propInRisky),xTicks,xTitle=xTitle,yTitle=yTitle)
abline(h=uStar, lty=3)
abline(v=rebDays, lty=3)
text(0,uStar,"u* = .6811",adj=c(.4,1.2),offset=.1,cex=.7)
savePlot("images/testrunPropInRisky",type="eps")

# Plotting risky asset, risk-free asset and portfolio values
BM = simObject$brownianMotion
simRisky = c(1,simRiskyAsset(1,drift,volatility,BM))
riskFree = riskFreeAsset(1,rent,6048)
yTitle = "Value"
niceplot(c(0,days),simRisky,xTicks,xTitle=xTitle,yTitle=yTitle,horizLines=T,col
    ="red")
abline(v=rebDays, lty=3)
nicelines(c(0,days),c(1,riskFree),col="dodgerblue")
nicelines(c(0,days),c(1,simWealth))
savePlot("images/testrun_wealth",type="eps")

#
# Simulating rebalancing strategy vs loss of utility, 1 mill. runs using
# 32 processor cores
#

# Simulating
nSims = 1000000
nCores = 32
nDailyRebs = c(24,6,1,1/2,1/12,1/21,1/42,1/126,1/252)
strategyNames = c("Hourly","Every 4th hour","Daily","Every 3rd day","Every 12th
    day","Monthly","Bimonthly","Semiannually","Annually")
paramSets = cbind(initWealth,nTradingDays,nDailyIncrements,nDailyRebs,drift,
    volatility,rent,riskAversion,uStar)
rebStrategy = distribute(nSims,nCores,simPortfolio,paramSets)
names(rebStrategy) = strategyNames
save(rebStrategy,file="Datasett/rebStrategy.RData")

# Calculating mean loss of utility
termWealth.th = lapply(rebStrategy,get,x="thTermWealth")
termUtility.th = lapply(termWealth.th,utility,param=riskAversion)
meanTermUtility.th = sapply(termUtility.th,mean)
termWealth.sim = lapply(rebStrategy,get,x="simTermWealth")
termUtility.sim = lapply(termWealth.sim,utility,param=riskAversion)
meanTermUtility.sim = sapply(termUtility.sim,mean)
LOU = listDiff(termUtility.th,termUtility.sim)
meansLOU = sapply(LOU,mean)
sdsLOU = sapply(LOU,sd)
sdMeansLOU = sdsLOU / sqrt(nSims)
lowerCL = meansLOU - qAlphaHalf*sdMeansLOU
upperCL = meansLOU + qAlphaHalf*sdMeansLOU

# Plotting mean losses of utility and 95% confidence intervals
yMin = min(lowerCL)
yMax = max(upperCL)
xTicks = 1:9
xTitle = "Rebalancing strategy"
yTitle = "Mean loss of utility"
```

```
niceplot(xTicks,xTitle=xTitle,yTitle=yTitle,meansLOU,xTicks,xLabels=
    strategyNames,ylim=c(yMin,yMax))
nicelines(xTicks,lowerCL,lty=3)
nicelines(xTicks,upperCL,lty=3)
abline(h=0,lty=3)
savePlot("images/reb_LOU",type="eps")

#
# Plotting and making histograms
#

# Plotting the losses of utility of the monthly-strategy
LOUmonthly = LOU$Monthly
save(LOUmonthly,file="Datasett/LOUmonthly.RData")
yTitle = "Loss of utility"
millionLabels = c(0,"100k","200k","300k","400k","500k","600k","700k","800k","900
    k","1M")
xTicks = seq(0,1000000,100000)
niceplot(LOUmonthly,xTicks=xTicks,xLabels=millionLabels,yTitle=yTitle,downsample
    =T)
abline(h=0,lty=3)
savePlot("images/LOUmonthly",type="eps")
yTitle = "Frequency"
xTitle = "Loss of utility"
nicehist(LOUmonthly,xTitle=xTitle,yTitle=yTitle,breaks=100,figsPerPage=4)
meanLOUmonthly = mean(LOUmonthly)
abline(v=meanLOUmonthly,lty=3)
savePlot("images/LOUfreqMonthly",type="eps")

# Histogram of the lower 1 percent of the monthly losses of utility
alpha = .05
qAlphaLOUmonthly = quantile(LOUmonthly,alpha)
nicehist(LOUmonthly[LOUmonthly<=qAlphaLOUmonthly],xTitle=xTitle,yTitle=yTitle,
    breaks=100,figsPerPage=4)
savePlot("images/LOUfreqLowerAlphaMonthly",type="eps")

# Cumulative mean of the losses of utility of the monthly strategy
cumMeanLOUmonthly = cumMean(LOUmonthly)
yTitle = "Mean loss of utility"
niceplot(cumMeanLOUmonthly,xLabels=millionLabels,yTitle=yTitle,figsPerPage=3,
    downsample=T,ylim=c(-.000025,.000025))
abline(h=0,lty=3)
cumSdMeanLOUmonthly = cumSd(LOUmonthly) / sqrt(nn)
qAlphaHalf = qnorm(1-alpha/2)
lowerCL = cumMeanLOUmonthly - qAlphaHalf*cumSdMeanLOUmonthly
upperCL = cumMeanLOUmonthly + qAlphaHalf*cumSdMeanLOUmonthly
nicelines(lowerCL,downsample=T,col="gray",lty=3)
nicelines(upperCL,downsample=T,col="gray",lty=3)
savePlot("images/meanLOUmonthly",type="eps")

# Cumulative mean of the losses of utility of the hourly strategy
transformation = 1e6
x = LOU$Hourly * transformation
cum.mean = cumMean(x)
nn = 1:length(x)
y.title = expression(paste("Mean loss of utility" %*% 10^-6))
niceplot(cum.mean,xLabels=millionLabels,yTitle=y.title,y.superscript=T,nCol=2,
    downsample=T,ylim=c(-1,1))
abline(h=0,lty=3)
cumMean.sd = cumSd(x) / sqrt(nn)
lowerCL = cum.mean - qAlphaHalf*cumMean.sd
upperCL = cum.mean + qAlphaHalf*cumMean.sd
nicelines(lowerCL,downsample=T,col="gray",lty=3)
nicelines(upperCL,downsample=T,col="gray",lty=3)
legendText = "(a) Rebalancing strategy: Hourly"
```

```
nicelegend("topright",legendText,bty="n",cex=.7,inset=c(.15,0))

# Cumulative mean of the losses of utility of the daily strategy
transformation = 1e6
x = LOU$Daily * transformation
cum.mean = cumMean(x)
nn = 1:length(x)
y.title = expression(paste("Mean loss of utility" %*% 10^-6))
niceplot(cum.mean,xLabels=millionLabels,yTitle=y.title,multiPlot=T,newDev=F, downsample=T,ylim=c(-3,3))
abline(h=0,lty=3)
cumMean.sd = cumSd(x) / sqrt(nn)
lowerCL = cum.mean - qAlphaHalf*cumMean.sd
upperCL = cum.mean + qAlphaHalf*cumMean.sd
nicelines(lowerCL,downsample=T,col="gray",lty=3)
nicelines(upperCL,downsample=T,col="gray",lty=3)
legendText = "(b) Rebalancing strategy: Daily"
nicelegend("topright",legendText,bty="n",cex=.7,inset=c(.15,0))
savePlot("images/cumMeanLOU_HourlyDaily",type="eps")

# Cumulative mean of the losses of utility of the 'every 3rd day' strategy
transformation = 1e6
x = LOU$"Every 3rd day" * transformation
cum.mean = cumMean(x)
nn = 1:length(x)
y.title = expression(paste("Mean loss of utility" %*% 10^-6))
niceplot(cum.mean,xLabels=millionLabels,yTitle=y.title,y.superscript=T,nCol=2, downsample=T,ylim=c(-3,3))
abline(h=0,lty=3)
cumMean.sd = cumSd(x) / sqrt(nn)
lowerCL = cum.mean - qAlphaHalf*cumMean.sd
upperCL = cum.mean + qAlphaHalf*cumMean.sd
nicelines(lowerCL,downsample=T,col="gray",lty=3)
nicelines(upperCL,downsample=T,col="gray",lty=3)
legendText = "(c) Rebalancing strategy: Every 3rd day"
nicelegend("topright",legendText,bty="n",cex=.7,inset=c(.17,0))

# Cumulative mean of the losses of utility of the 'every 12th day' strategy
transformation = 1e5
x = LOU$"Every 12th day" * transformation
cum.mean = cumMean(x)
nn = 1:length(x)
y.title = expression(paste("Mean loss of utility" %*% 10^-5))
niceplot(cum.mean,xLabels=millionLabels,yTitle=y.title,multiPlot=T,newDev=F, downsample=T,ylim=c(-.7,.7))
abline(h=0,lty=3)
cumMean.sd = cumSd(x) / sqrt(nn)
lowerCL = cum.mean - qAlphaHalf*cumMean.sd
upperCL = cum.mean + qAlphaHalf*cumMean.sd
nicelines(lowerCL,downsample=T,col="gray",lty=3)
nicelines(upperCL,downsample=T,col="gray",lty=3)
legendText = "(d) Rebalancing strategy: Every 12th day"
nicelegend("topright",legendText,bty="n",cex=.7,inset=c(.18,0))
savePlot("images/cumMeanLOU_3rd12th",type="eps")

# Cumulative mean of the losses of utility of the monthly strategy
transformation = 1e5
x = LOU$Monthly * transformation
cum.mean = cumMean(x)
nn = 1:length(x)
y.title = expression(paste("Mean loss of utility" %*% 10^-5))
niceplot(cum.mean,xLabels=millionLabels,yTitle=y.title,y.superscript=T,nCol=2, downsample=T,ylim=c(-1,1))
abline(h=0,lty=3)
cumMean.sd = cumSd(x) / sqrt(nn)
```

```r
lowerCL = cum.mean - qAlphaHalf*cumMean.sd
upperCL = cum.mean + qAlphaHalf*cumMean.sd
nicelines(lowerCL,downsample=T,col="gray",lty=3)
nicelines(upperCL,downsample=T,col="gray",lty=3)
legendText = "(e) Rebalancing strategy: Monthly"
nicelegend("topright",legendText,bty="n",cex=.7,inset=c(.15,0))

# Cumulative mean of the losses of utility of the bimonthly strategy
transformation = 1e5
x = LOU$Bimonthly * transformation
cum.mean = cumMean(x)
nn = 1:length(x)
y.title = expression(paste("Mean loss of utility" %*% 10^-5))
niceplot(cum.mean,xLabels=millionLabels,yTitle=y.title,multiPlot=T,newDev=F,downsample=T,ylim=c(-1.2,1.2))
abline(h=0,lty=3)
cumMean.sd = cumSd(x) / sqrt(nn)
lowerCL = cum.mean - qAlphaHalf*cumMean.sd
upperCL = cum.mean + qAlphaHalf*cumMean.sd
nicelines(lowerCL,downsample=T,col="gray",lty=3)
nicelines(upperCL,downsample=T,col="gray",lty=3)
legendText = "(f) Rebalancing strategy: Bimonthly"
nicelegend("topright",legendText,bty="n",cex=.7,inset=c(.16,0))
savePlot("images/cumMeanLOU_MonthlyBi",type="eps")

# Cumulative mean of the losses of utility of the semiannual strategy
transformation = 1e5
x = LOU$"Half-yearly" * transformation
cum.mean = cumMean(x)
nn = 1:length(x)
y.title = expression(paste("Mean loss of utility" %*% 10^-5))
niceplot(cum.mean,xLabels=millionLabels,yTitle=y.title,y.superscript=T,nCol=2,downsample=T,ylim=c(-3,3))
abline(h=0,lty=3)
cumMean.sd = cumSd(x) / sqrt(nn)
lowerCL = cum.mean - qAlphaHalf*cumMean.sd
upperCL = cum.mean + qAlphaHalf*cumMean.sd
nicelines(lowerCL,downsample=T,col="gray",lty=3)
nicelines(upperCL,downsample=T,col="gray",lty=3)
legendText = "(g) Rebalancing strategy: Semiannually"
nicelegend("topright",legendText,bty="n",cex=.7,inset=c(.18,0))

# Cumulative mean of the losses of utility of the annual strategy
transformation = 1e5
x = LOU$Yearly * transformation
cum.mean = cumMean(x)
nn = 1:length(x)
y.title = expression(paste("Mean loss of utility" %*% 10^-5))
niceplot(cum.mean,xLabels=millionLabels,yTitle=y.title,multiPlot=T,newDev=F,downsample=T,ylim=c(-5,5))
abline(h=0,lty=3)
cumMean.sd = cumSd(x) / sqrt(nn)
lowerCL = cum.mean - qAlphaHalf*cumMean.sd
upperCL = cum.mean + qAlphaHalf*cumMean.sd
nicelines(lowerCL,downsample=T,col="gray",lty=3)
nicelines(upperCL,downsample=T,col="gray",lty=3)
legendText = "(h) Rebalancing strategy: Annually"
nicelegend("topright",legendText,bty="n",cex=.7,inset=c(.15,0))
savePlot("images/cumMeanLOU_AnnuallySemi",type="eps")

#
# Rebalancing strategy vs Sharpe ratio
#

nTimePoints = 6048
```

```
expecLogReturn = expectedLogReturn(drift, volatility, rent, uStar, 1)
sdLogReturn = stDevLogReturn(volatility, uStar, 1)
exAnteSR = exAnteSharpeRatio(drift, volatility, rent, uStar, 1)
logAdjustedSR = logAdjustedSharpeRatio(drift, volatility, rent, uStar, 1)

# Calculating Sharpe ratios of theoretical portfolios
strategy_termWealth.th = sapply(rebStrategy, get, x="thTermWealth")
strategy_meanTermWealth.th = colMeans(strategy_termWealth.th)
strategy_logReturn.th = log(strategy_termWealth.th)
strategy_meanLogReturn.th = colMeans(strategy_logReturn.th)
strategy_excessReturn.th = strategy_logReturn.th - rent
strategy_meanExcessReturn.th = colMeans(strategy_excessReturn.th)
strategy_sdLogReturn.th = sapply(rebStrategy, get, x="sdThLogReturn")
strategy_meanSdLogReturn.th = colMeans(strategy_sdLogReturn.th)
strategy_annualizedSdLogReturn.th = strategy_sdLogReturn.th * sqrt(nTimePoints)
strategy_meanAnnualizedSdLogReturn.th = colMeans(strategy_annualizedSdLogReturn.th)
strategy_volOfVol.th = colSds(strategy_annualizedSdLogReturn.th)
strategy_correlation.th = colCorrs(strategy_logReturn.th, strategy_sdLogReturn.th)
strategy_SR.th = strategy_excessReturn.th / (sqrt(nTimePoints)*strategy_sdLogReturn.th)
save(strategy_SR.th, file="Datasett/strategy_SR.th.RData")
strategy_meanSR.th = colMeans(strategy_SR.th)
strategy_sdSR.th = colSds(strategy_SR.th)
strategy_sdMeanSR.th = strategy_sdSR.th / sqrt(nrow(strategy_SR.th))

# Calculating Sharpe ratios of simulated portfolios
strategy_termWealth.sim = sapply(rebStrategy, get, x="simTermWealth")
strategy_meanTermWealth.sim = colMeans(strategy_termWealth.sim)
strategy_lossOfWealth.sim = strategy_termWealth.th - strategy_termWealth.sim
strategy_meanLossOfWealth.sim = colMeans(strategy_lossOfWealth.sim)
strategy_logReturn.sim = log(strategy_termWealth.sim)
strategy_meanLogReturn.sim = colMeans(strategy_logReturn.sim)
strategy_sdTermLogReturn.sim = colSds(strategy_logReturn.sim)
strategy_excessReturn.sim = strategy_logReturn.sim - rent
strategy_meanExcessReturn.sim = colMeans(strategy_excessReturn.sim)
strategy_sdLogReturn.sim = sapply(rebStrategy, get, x="sdSimLogReturn")
strategy_meanSdLogReturn.sim = colMeans(strategy_sdLogReturn.sim)
strategy_annualizedSdLogReturn.sim = strategy_sdLogReturn.sim * sqrt(nTimePoints)
strategy_meanAnnualizedSdLogReturn.sim = colMeans(strategy_annualizedSdLogReturn.sim)
strategy_volOfVol.sim = colSds(strategy_annualizedSdLogReturn.sim)
strategy_correlation.sim = colCorrs(strategy_logReturn.sim, strategy_sdLogReturn.sim)
strategy_SR.sim = strategy_excessReturn.sim / (strategy_annualizedSdLogReturn.sim)
save(strategy_SR.sim, file="Datasett/strategy_SR.sim.RData")
strategy_meanSR.sim = colMeans(strategy_SR.sim)
strategy_ranking.sim = c(1,3,2,4,5,6,7,8,9)
strategy_sdSR.sim = colSds(strategy_SR.sim)
strategy_sdMeanSR.sim = strategy_sdSR.sim / sqrt(nrow(strategy_SR.sim))
strategy_testStat.sim = (strategy_meanSR.sim - exAnteSR) / strategy_sdMeanSR.sim
strategy_pValue.sim = 2*pnorm(-abs(strategy_testStat.sim))

# Calculating confidence intervals of the mean Sharpe ratios
strategy_lowerCL.sim = strategy_meanSR.sim - qAlphaHalf*strategy_sdMeanSR.sim
strategy_upperCL.sim = strategy_meanSR.sim + qAlphaHalf*strategy_sdMeanSR.sim

# Creating tables for printout
tab1 = matrix(NA,18,5)
for (i in 1:9) {
  tab1[2*i-1,] = c(strategy_meanTermWealth.th[i],0,meanTermUtility.th[i],0,0)
```

```
  tab1[2*i,] = c(strategy_meanTermWealth.sim[i],strategy_meanLossOfWealth.sim[i],meanTermUtility.sim[i],meansLOU[i],sdsLOU[i])
}
tab1[,1] = round(tab1[,1],4)
tab1[,2] = round(tab1[,2]*1e5,4)
tab1[,3] = round(tab1[,3],4)
tab1[,4] = round(tab1[,4]*1e5,4)
tab1[,5] = round(tab1[,5]*1e3,4)

for (k in 1:18) {
  tab1[k,2] = paste(tab1[k,2],"\\e{\\text-5}",sep="")
  tab1[k,4] = paste(tab1[k,4],"\\e{\\text-5}",sep="")
  tab1[k,5] = paste(tab1[k,5],"\\e{\\text-3}",sep="")
}

printex(tab1)

tab2 = matrix(NA,18,5)
for (i in 1:9) {
  tab2[2*i-1,] = c(strategy_meanLogReturn.th[i],strategy_meanAnnualizedSdLogReturn.th[i],strategy_meanSR.th[i],strategy_volOfVol.th[i],strategy_correlation.th[i])
  tab2[2*i,] = c(strategy_meanLogReturn.sim[i],strategy_meanAnnualizedSdLogReturn.sim[i],strategy_meanSR.sim[i],strategy_volOfVol.sim[i],strategy_correlation.sim[i])
}
colnames(tab2) = c("meanLogRet","annMeanSdLogRet","meanSR","volOfVol","corr")
tab2[,1] = round(tab2[,1]*1e2,4)
tab2[,2] = round(tab2[,2],4)
tab2[,3] = round(tab2[,3]*1e2,4)
tab2[,4] = round(tab2[,4]*1e2,4)
tab2[,5] = round(tab2[,5],4)

for (k in 1:18) {
  tab2[k,1] = paste(tab2[k,1],"\\e{\\text-2}",sep="")
  tab2[k,3] = paste(tab2[k,3],"\\e{\\text-2}",sep="")
  tab2[k,4] = paste(tab2[k,4],"\\e{\\text-2}",sep="")
}

printex(tab2)

# Plotting mean Sharpe ratios vs rebalancing strategies
xTicks = 1:9
xTitle = "Rebalancing strategy"
yTitle = "Sharpe ratio"
yMin = min(c(0,strategy_lowerCL.sim))
yMax = max(strategy_upperCL.sim)
niceplot(xTicks,xTitle=xTitle,yTitle=yTitle,strategy_meanSR.sim,xTicks,xLabels=strategyNames,ylim=c(yMin,yMax))
abline(v=xTicks,col="gray",lty=3)
nicelines(strategy_lowerCL.sim,lty=2)
nicelines(strategy_upperCL.sim,lty=2)
abline(h=exAnteSR,lty=3)
text(8.39,exAnteSR,paste("ex ante Sharpe ratio =",round(exAnteSR,4)),pos=1,offset=.2,cex=.7)
savePlot("images/exPostSharpeRatio",type="eps")

# Histogram of the losses of utility of the hourly-strategy superimposed
# on a histogram of the losses of utility of the annual-strategy.
histObject = hist(strategy_annualizedSdLogReturn.sim[,9],breaks=100,plot=F)
breakPoints = histObject$breaks
histObject = hist(strategy_annualizedSdLogReturn.sim[,1],breaks=breakPoints,plot=F)
yTitle = "Frequency"
xTitle = "Standard deviation"
```

```r
yMin = 0
yMax = max(histObject$counts)
yRange = c(yMin,yMax)
nicehist(strategy_annualizedSdLogReturn.sim[,9],xTitle=xTitle,yTitle=yTitle,breaks=breakPoints,ylim=yRange)
addHist(strategy_annualizedSdLogReturn.sim[,1],density=30)
savePlot("images/sdHourlyVsAnnually",type="eps")

strategy_sdAnnualizedSdLogReturn.sim = colSds(strategy_annualizedSdLogReturn.sim)
strategy_corrLogReturnSdLogReturn.sim = colCorrs(strategy_logReturn.sim,strategy_annualizedSdLogReturn.sim)
tab2 = cbind(strategy_meanAnnualizedSdLogReturn.sim,strategy_sdAnnualizedSdLogReturn.sim,strategy_corrLogReturnSdLogReturn.sim)
tab2[,2] = tab2[,2] * 1e2
rownames(tab2) = strategyNames
printex(trimLeadingZero(tab2))

# Histograms of losses of utility of the hourly and the daily strategy
x.title = "Loss of utility"
y.title = "Frequency"
breaksLength = 70
res = seq(min(LOU$Hourly),max(LOU$Hourly),length=breaksLength)
histObject = hist(LOU$Hourly,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.1
nicehist(LOU$Hourly,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c("(a) Rebalancing strategy: Hourly",expression(paste("Mean = -.0300" %*% 10^-5)),expression(paste("StDev = .1471" %*% 10^-3)))
nicelegend("topleft",legendText,bty="n",cex=.7)
res = seq(min(LOU$Daily),max(LOU$Daily),length=breaksLength)
histObject = hist(LOU$Daily,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.1
nicehist(LOU$Daily,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,breaks=res)
legendText = c("(b) Rebalancing strategy: Daily",expression(paste("Mean = .0670" %*% 10^-5)),expression(paste("StDev = .3623" %*% 10^-3)))
nicelegend("topleft",legendText,bty="n",cex=.7)
savePlot("images/histLouHourlyDaily",type="eps")

# Histograms of losses of utility of the 'every 3rd day'-strategy and the
# 'every 12th day'-strategy
res = seq(min(LOU$"Every 3rd day"),max(LOU$"Every 3rd day"),length=breaksLength)
histObject = hist(LOU$"Every 3rd day",breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.1
nicehist(LOU$"Every 3rd day",xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c("(c) Rebalancing strategy: Every 3rd day",expression(paste("Mean = .0414" %*% 10^-5)),expression(paste("StDev = .4947" %*% 10^-3)))
nicelegend("topleft",legendText,bty="n",cex=.7)
res = seq(min(LOU$"Every 12th day"),max(LOU$"Every 12th day"),length=breaksLength)
histObject = hist(LOU$"Every 12th day",breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.1
nicehist(LOU$"Every 12th day",xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,breaks=res)
legendText = c("(d) Rebalancing strategy: Every 12th day",expression(paste("Mean = .1424" %*% 10^-5)),expression(paste("StDev = 1.1764" %*% 10^-3)))
nicelegend("topleft",legendText,bty="n",cex=.7)
savePlot("images/histLou3rd12th",type="eps")

# Histograms of losses of utility of the monthly and the bimonthly strategy
res = seq(min(LOU$Monthly),max(LOU$Monthly),length=breaksLength)
histObject = hist(LOU$Monthly,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.1
nicehist(LOU$Monthly,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
```

```
legendText = c("(e) Rebalancing strategy: Monthly",expression(paste("Mean =
    -.1769" %*% 10^-5)),expression(paste("StDev = 1.5515" %*% 10^-3)))
nicelegend("topleft",legendText,bty="n",cex=.7)
res = seq(min(LOU$Bimonthly),max(LOU$Bimonthly),length=breaksLength)
histObject = hist(LOU$Bimonthly,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.1
nicehist(LOU$Bimonthly,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y
    .lim,breaks=res)
legendText = c("(f) Rebalancing strategy: Bimonthly",expression(paste("Mean =
    .2887" %*% 10^-5)),expression(paste("StDev = 2.1899" %*% 10^-3)))
nicelegend("topleft",legendText,bty="n",cex=.7)
savePlot("images/histLouMonthlyBi",type="eps")

# Histograms of losses of utility of the semi-annual and the annual strategy
res = seq(min(LOU$"Half-yearly"),max(LOU$"Half-yearly"),length=breaksLength)
histObject = hist(LOU$"Half-yearly",breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.1
nicehist(LOU$"Half-yearly",xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,
    breaks=res)
legendText = c("(g) Rebalancing strategy: Semiannually",expression(paste("Mean =
     1.2969" %*% 10^-5)),expression(paste("StDev = 3.7678" %*% 10^-3)))
nicelegend("topleft",legendText,bty="n",cex=.7)
res = seq(min(LOU$Yearly),max(LOU$Yearly),length=breaksLength)
histObject = hist(LOU$Yearly,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.1
nicehist(LOU$Yearly,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.
    lim,breaks=res)
legendText = c("(h) Rebalancing strategy: Annually",expression(paste("Mean =
    2.0172" %*% 10^-5)),expression(paste("StDev = 5.3237" %*% 10^-3)))
nicelegend("topleft",legendText,bty="n",cex=.7)
savePlot("images/histLouHalfYearly",type="eps")
```

B.5 Simulation model II and III

B.5.1 Simulation machinery

```
##
# Master Thesis
# Simulation model II and III
# Simulation algorithm
#

simPortfolio.transCost = function(nSims,paramSet,brownianFileName=NULL) {
  #
  # Simulates nSims portfolios following the 9 parameter values of paramSet
  # and returns terminal utilities of theoretical and simulated wealth and
  # the loss of utility. Includes transaction costs!
  #
  logReturn = function(x) {
    #
    # Computes the log returns of a time series x.
          #
    n = length(x)
          xUp = x[2:n]
          xLow = x[1:(n-1)]
          logReturns = log(xUp/xLow)
          return(logReturns)
  }
```

```

  brownianIncrement = function(n,delta) {
    #
    # Simulates random series of n brownian increments with variance delta.
    #
    return(rnorm(n,0,sqrt(delta)))
  }

  #
  # Assigning variables.
  #
  nParams = length(paramSet)
  if (nParams != 10) stop(paste("Number of input parameters equals ",nParams,". Must equal 10.",sep=""))
  varNames = c("initWealth","nTradingDays","nDailyIncrements","nDailyRebs","drift","volatility","rent","riskAversion","uStar","costProp")
  for (j in 1:10) { assign(varNames[j],paramSet[j]) }

  #
  # Initializing the simulation structure.
  #

  simIndex = 1:nSims
  nTimePoints = nTradingDays * nDailyIncrements
  lastIndex = nTimePoints
  delta = 1 / nTimePoints
  timePoints = seq(delta,1,delta)
  nRebDelay = nDailyIncrements / nDailyRebs
  rebIndex = seq(nRebDelay,nTimePoints,nRebDelay)
  #rebIndex = rebIndex[-length(rebIndex)]
  days = seq(delta*nTradingDays,nTradingDays,delta*nTradingDays)
  rebDays = days[rebIndex]
  ones = rep(1,nRebDelay)

  # Start of simulation time
  timeStart = proc.time()[3][[1]]

  # Common structure
  simWealth = NA
  simWealth.pre = NA
  simWealth.sub = NA

  #
  # Using full simulation scheme if nSims = 1
  #

  if (nSims == 1) {

    # Intializing other statistics
    riskyReturn = 1:nTimePoints * NA
    riskfreeReturn = 1:nTimePoints * NA

    # Intializing simulated wealth without transaction costs
    simWealth.risky = NA
    simWealth.riskfree = NA
    transQuantity = 1:nTimePoints * 0
    propInRisky = NA
    propInRiskfree = NA

    # Intializing simulated wealth with preceding transaction costs
    simWealth.risky.pre = NA
    simWealth.riskfree.pre = NA
    transQuantity.pre = 1:nTimePoints * 0
    transCost.pre = 1:nTimePoints * 0
    propInRisky.pre = NA
```

```
    propInRiskfree.pre = NA

    # Intializing simulated wealth with subsequent transaction costs
    simWealth.risky.sub = NA
    simWealth.riskfree.sub = NA
    transQuantity.sub = 1:nTimePoints * 0
    transCost.sub = 1:nTimePoints * 0
    propInRisky.sub = NA
    propInRiskfree.sub = NA

    # Generation of Brownian motion
    if (!is.null(brownianFileName) && file.exists(brownianFileName,sep="")) {
        cat("Loading brownian increments...\n"); load(brownianFileName) }
    else { inc = brownianIncrement(nTimePoints,delta) }
    if (!is.null(brownianFileName) && !file.exists(brownianFileName)) { cat("
        Saving brownian increments...\n"); save(inc,file=brownianFileName) }
    if (exists("dualInc")) { inc = dualInc[,1] }
    BM = cumsum(inc)

    # Initialization and calculation of theoretical wealth
    initRiskyPrice = 1
    riskyPrice = initRiskyPrice*exp((drift-.5*volatility^2)*timePoints+
        volatility*BM)
    initRiskfreePrice = 1
    riskfreePrice = initRiskfreePrice*exp(rent*timePoints)
    thWealth = initWealth*exp((drift*uStar+rent*(1-uStar)-.5*volatility^2*uStar
        ^2)*timePoints+volatility*uStar*BM)

    #
    # First part of the simulations
    #

    # Time points to be simulated
    activeIndices = 1:nRebDelay
    rebPoint = tail(activeIndices,1)

    # Calculating risky and risk free returns
    riskyReturn[activeIndices] = cumprod(1+drift*delta+volatility*inc[
        activeIndices]) - 1
    riskfreeReturn[activeIndices] = cumprod((1+rent*delta)*ones) - 1

    # Without transaction costs
    simWealth.risky = uStar*initWealth*cumprod(1+drift*delta+volatility*inc[
        activeIndices])
    simWealth.riskfree = (1-uStar)*initWealth*cumprod((1+rent*delta)*ones)
    simWealth = simWealth.risky + simWealth.riskfree
    simWealth.risky.prime = simWealth.risky[rebPoint]
    simWealth.riskfree.prime = simWealth.riskfree[rebPoint]
    transQuantity[rebPoint] = ((1-uStar)*simWealth.risky.prime - uStar*simWealth
        .riskfree.prime)
    simWealth.risky[rebPoint] = simWealth.risky.prime - transQuantity[rebPoint]
    simWealth.riskfree[rebPoint] = simWealth.riskfree.prime + transQuantity[
        rebPoint]
    simWealth[rebPoint] = simWealth.risky[rebPoint] + simWealth.riskfree[
        rebPoint]
    propInRisky[activeIndices] = simWealth.risky / simWealth
    propInRiskfree[activeIndices] = simWealth.riskfree / simWealth

    # Preceding transaction costs
    simWealth.risky.pre = uStar*initWealth*cumprod(1+drift*delta+volatility*inc[
        activeIndices])
    simWealth.riskfree.pre = (1-uStar)*initWealth*cumprod((1+rent*delta)*ones)
    simWealth.pre = simWealth.risky.pre + simWealth.riskfree.pre
    simWealth.risky.pre.prime = simWealth.risky.pre[rebPoint]
    simWealth.riskfree.pre.prime = simWealth.riskfree.pre[rebPoint]
```

```r
    signDiffReturn.pre = sign(prod(1+drift*delta+volatility*inc[activeIndices]) - prod((1+rent*delta)*ones))
    transQuantity.pre[rebPoint] = ((1-uStar)*simWealth.risky.pre.prime - uStar*simWealth.riskfree.pre.prime) / (1-signDiffReturn.pre*costProp*uStar)
    transCost.pre[rebPoint] = abs(costProp*transQuantity.pre[rebPoint])
    simWealth.risky.pre[rebPoint] = simWealth.risky.pre.prime - transQuantity.pre[rebPoint]
    simWealth.riskfree.pre[rebPoint] = simWealth.riskfree.pre.prime + transQuantity.pre[rebPoint] - costProp*abs(transQuantity.pre[rebPoint])
    simWealth.pre[rebPoint] = simWealth.risky.pre[rebPoint] + simWealth.riskfree.pre[rebPoint]
    propInRisky.pre[activeIndices] = simWealth.risky.pre / simWealth.pre
    propInRiskfree.pre[activeIndices] = simWealth.riskfree.pre / simWealth.pre

    # Subsequent transaction costs
    simWealth.risky.sub = simWealth.risky.pre
    simWealth.riskfree.sub = simWealth.riskfree.pre
    simWealth.sub = simWealth.pre
    simWealth.risky.sub.prime = simWealth.risky.pre.prime
    simWealth.riskfree.sub.prime = simWealth.riskfree.pre.prime
    transQuantity.sub[rebPoint] = (1-uStar)*simWealth.risky.sub.prime - uStar*simWealth.riskfree.sub.prime
    transCost.sub[rebPoint] = abs(costProp*transQuantity.sub[rebPoint])
    simWealth.risky.sub[rebPoint] = simWealth.risky.sub.prime - transQuantity.sub[rebPoint]
    simWealth.riskfree.sub[rebPoint] = simWealth.riskfree.sub.prime + transQuantity.sub[rebPoint] - costProp*abs(transQuantity.sub[rebPoint])
    simWealth.sub[rebPoint] = simWealth.risky.sub[rebPoint] + simWealth.riskfree.sub[rebPoint]
    propInRisky.sub[activeIndices] = simWealth.risky.sub / simWealth.sub
    propInRiskfree.sub[activeIndices] = simWealth.riskfree.sub / simWealth.sub

    for (j in rebIndex[-length(rebIndex)] + 1) {
                activeIndices = j:(j+nRebDelay-1)
                rebPoint = tail(activeIndices,1)

                # Calculating risky and risk free returns
      riskyReturn[activeIndices] = cumprod(1+drift*delta+volatility*inc[activeIndices]) - 1
      riskfreeReturn[activeIndices] = cumprod((1+rent*delta)*ones) - 1

              # Without transaction costs
      simWealth.risky[activeIndices] = uStar*simWealth[j-1]*cumprod(1+drift*delta+volatility*inc[activeIndices])
      simWealth.riskfree[activeIndices] = (1-uStar)*simWealth[j-1]*cumprod((1+rent*delta)*ones)
      simWealth[activeIndices] = simWealth.risky[activeIndices] + simWealth.riskfree[activeIndices]
      simWealth.risky.prime = simWealth.risky[rebPoint]
      simWealth.riskfree.prime = simWealth.riskfree[rebPoint]
      transQuantity[rebPoint] = ((1-uStar)*simWealth.risky.prime - uStar*simWealth.riskfree.prime)
      simWealth.risky[rebPoint] = simWealth.risky.prime - transQuantity[rebPoint]
      simWealth.riskfree[rebPoint] = simWealth.riskfree.prime + transQuantity[rebPoint]
      simWealth[rebPoint] = simWealth.risky[rebPoint] + simWealth.riskfree[rebPoint]
      propInRisky[activeIndices] = simWealth.risky[activeIndices] / simWealth[activeIndices]
      propInRiskfree[activeIndices] = simWealth.riskfree[activeIndices] / simWealth[activeIndices]

      # Preceding transaction costs
```

```
      simWealth.risky.pre[activeIndices] = uStar*simWealth.pre[j-1]*cumprod(1+
          drift*delta+volatility*inc[activeIndices])
      simWealth.riskfree.pre[activeIndices] = (1-uStar)*simWealth.pre[j-1]*
          cumprod((1+rent*delta)*ones)
      simWealth.pre[activeIndices] = simWealth.risky.pre[activeIndices] +
          simWealth.riskfree.pre[activeIndices]
      simWealth.risky.pre.prime = simWealth.risky.pre[rebPoint]
      simWealth.riskfree.pre.prime = simWealth.riskfree.pre[rebPoint]
      signDiffReturn.pre = sign(prod(1+drift*delta+volatility*inc[activeIndices
          ]) - prod((1+rent*delta)*ones))
      transQuantity.pre[rebPoint] = ((1-uStar)*simWealth.risky.pre.prime - uStar
          *simWealth.riskfree.pre.prime) / (1-signDiffReturn.pre*costProp*uStar)
      transCost.pre[rebPoint] = abs(costProp*transQuantity.pre[rebPoint])
      simWealth.risky.pre[rebPoint] = simWealth.risky.pre.prime - transQuantity.
          pre[rebPoint]
      simWealth.riskfree.pre[rebPoint] = simWealth.riskfree.pre.prime +
          transQuantity.pre[rebPoint] - costProp*abs(transQuantity.pre[rebPoint
          ])
      simWealth.pre[rebPoint] = simWealth.risky.pre[rebPoint] + simWealth.
          riskfree.pre[rebPoint]
      propInRisky.pre[activeIndices] = simWealth.risky.pre[activeIndices] /
          simWealth.pre[activeIndices]
      propInRiskfree.pre[activeIndices] = simWealth.riskfree.pre[activeIndices]
          / simWealth.pre[activeIndices]

      # Subsequent transaction costs
      simWealth.risky.sub[activeIndices] = uStar*simWealth.sub[j-1]*cumprod(1+
          drift*delta+volatility*inc[activeIndices])
      simWealth.riskfree.sub[activeIndices] = (1-uStar)*simWealth.sub[j-1]*
          cumprod((1+rent*delta)*ones)
      simWealth.sub[activeIndices] = simWealth.risky.sub[activeIndices] +
          simWealth.riskfree.sub[activeIndices]
      simWealth.risky.sub.prime = simWealth.risky.sub[rebPoint]
      simWealth.riskfree.sub.prime = simWealth.riskfree.sub[rebPoint]
      transQuantity.sub[rebPoint] = (1-uStar)*simWealth.risky.sub.prime - uStar*
          simWealth.riskfree.sub.prime
      transCost.sub[rebPoint] = abs(costProp*transQuantity.sub[rebPoint])
      simWealth.risky.sub[rebPoint] = simWealth.risky.sub.prime - transQuantity.
          sub[rebPoint]
      simWealth.riskfree.sub[rebPoint] = simWealth.riskfree.sub.prime +
          transQuantity.sub[rebPoint] - costProp*abs(transQuantity.sub[rebPoint
          ])
      simWealth.sub[rebPoint] = simWealth.risky.sub[rebPoint] + simWealth.
          riskfree.sub[rebPoint]
      propInRisky.sub[activeIndices] = simWealth.risky.sub[activeIndices] /
          simWealth.sub[activeIndices]
      propInRiskfree.sub[activeIndices] = simWealth.riskfree.sub[activeIndices]
          / simWealth.sub[activeIndices]
                }
  }

  #
  # Using compact form of simulation scheme if nSims > 1
  #

  else {

    thWealth.sd = simIndex * NA
    thWealth.terminal = simIndex * NA
    thWealth.logReturn.sd = simIndex * NA

    simWealth.sd = simIndex * NA
    simWealth.terminal = simIndex * NA
    simWealth.logReturn.sd = simIndex * NA

```

```
    simWealth.pre.sd = simIndex * NA
    simWealth.pre.terminal = simIndex * NA
    simWealth.pre.logReturn.sd = simIndex * NA
    totalTransCost.pre = simIndex * 0

    simWealth.sub.sd = simIndex * NA
    simWealth.sub.terminal = simIndex * NA
    simWealth.sub.logReturn.sd = simIndex * NA
    totalTransCost.sub = simIndex * 0

    for (k in 1:nSims) {

      # Generation of Brownian motion
      inc = brownianIncrement(nTimePoints, delta)
      BM = cumsum(inc)

      # Initialization and calculation of theoretical wealth
      thWealth = initWealth*exp((drift*uStar+rent*(1-uStar)-.5*volatility^2*
          uStar^2)*timePoints+volatility*uStar*BM)

      #
      # Simulated wealths until first rebalancing time point
      #

      # Common quantities
      activeIndices = 1:nRebDelay
      rebPoint = tail(activeIndices,1)
      return.risky = prod(1+drift*delta+volatility*inc[activeIndices])
      return.riskfree = prod((1+rent*delta)*ones)
      diffReturn = return.risky - return.riskfree

      # No transaction costs
      simWealth[activeIndices] = uStar*initWealth*cumprod(1+drift*delta+
          volatility*inc[activeIndices]) + (1-uStar)*initWealth*cumprod((1+rent*
          delta)*ones)

      # Preceding transaction costs
      simWealth.pre[activeIndices] = uStar*initWealth*cumprod(1+drift*delta+
          volatility*inc[activeIndices]) + (1-uStar)*initWealth*cumprod((1+rent*
          delta)*ones)
      signDiffReturn = sign(diffReturn)
      transCost.pre = costProp * abs((uStar*(1-uStar)*initWealth*diffReturn) /
          (1-signDiffReturn*costProp*uStar))
      totalTransCost.pre[k] = totalTransCost.pre[k] + transCost.pre
      simWealth.pre[rebPoint] = uStar*initWealth*return.risky + (1-uStar)*
          initWealth*return.riskfree - transCost.pre

      # Subsequent transaction costs
      simWealth.sub[activeIndices] = uStar*initWealth*cumprod(1+drift*delta+
          volatility*inc[activeIndices]) + (1-uStar)*initWealth*cumprod((1+rent*
          delta)*ones)
      transCost.sub = costProp * abs(uStar*(1-uStar)*initWealth*diffReturn)
      totalTransCost.sub[k] = totalTransCost.sub[k] + transCost.sub
      simWealth.sub[rebPoint] = uStar*initWealth*return.risky + (1-uStar)*
          initWealth*return.riskfree - transCost.sub

      #
      # The rest of the simulated wealths
      #

      for (j in rebIndex[-length(rebIndex)] + 1) {

        # Common quantities
        activeIndices = j:(j+nRebDelay-1)
              rebPoint = tail(activeIndices,1)
```

```
              return.risky = prod(1+drift*delta+volatility*inc[activeIndices])
              return.riskfree = prod((1+rent*delta)*ones)
              diffReturn = return.risky - return.riskfree

          # No transaction costs
          simWealth[activeIndices] = uStar*simWealth[j-1]*cumprod(1+drift*delta+volatility*inc[activeIndices]) + (1-uStar)*simWealth[j-1]*cumprod((1+rent*delta)*ones)

          # Preceding transaction costs
          simWealth.pre[activeIndices] = uStar*simWealth.pre[j-1]*cumprod(1+drift*delta+volatility*inc[activeIndices]) + (1-uStar)*simWealth.pre[j-1]*cumprod((1+rent*delta)*ones)
          signDiffReturn = sign(diffReturn)
          transCost.pre = costProp * abs((uStar*(1-uStar)*simWealth.pre[j-1]*diffReturn) / (1-signDiffReturn*costProp*uStar))
          totalTransCost.pre[k] = totalTransCost.pre[k] + transCost.pre
          simWealth.pre[rebPoint] = uStar*simWealth.pre[j-1]*return.risky + (1-uStar)*simWealth.pre[j-1]*return.riskfree - transCost.pre

          # Subsequent transaction costs
          simWealth.sub[activeIndices] = uStar*simWealth.sub[j-1]*cumprod(1+drift*delta+volatility*inc[activeIndices]) + (1-uStar)*simWealth.sub[j-1]*cumprod((1+rent*delta)*ones)
          transCost.sub = costProp * abs(uStar*(1-uStar)*simWealth.sub[j-1]*diffReturn)
          totalTransCost.sub[k] = totalTransCost.sub[k] + transCost.sub
          simWealth.sub[rebPoint] = uStar*simWealth.sub[j-1]*return.risky + (1-uStar)*simWealth.sub[j-1]*return.riskfree - transCost.sub
                }

          thWealth.sd[k] = sd(thWealth)
        thWealth.terminal[k] = thWealth[lastIndex]
        thWealth.logReturn = logReturn(c(initWealth,thWealth))
        thWealth.logReturn.sd[k] = sd(thWealth.logReturn)

        simWealth.sd[k] = sd(simWealth)
        simWealth.terminal[k] = simWealth[lastIndex]
        simWealth.logReturn = logReturn(c(initWealth,simWealth))
        simWealth.logReturn.sd[k] = sd(simWealth.logReturn)

        simWealth.pre.sd[k] = sd(simWealth.pre)
        simWealth.pre.terminal[k] = simWealth.pre[lastIndex]
        simWealth.pre.logReturn = logReturn(c(initWealth,simWealth.pre))
        simWealth.pre.logReturn.sd[k] = sd(simWealth.pre.logReturn)

        simWealth.sub.sd[k] = sd(simWealth.sub)
        simWealth.sub.terminal[k] = simWealth.sub[lastIndex]
        simWealth.sub.logReturn = logReturn(c(initWealth,simWealth.sub))
        simWealth.sub.logReturn.sd[k] = sd(simWealth.sub.logReturn)
    }
  }

  # Calculation of total simulation time
  timeElapsed = proc.time()[3][[1]] - timeStart
  cat(nSims," simulation(s) completed in",timeElapsed," seconds.\n")
  flush.console()

  # Construction of the list of data to be returned from the function.
  if (nSims == 1) {
    stdNames = c("simWealth.risky","simWealth.riskfree","simWealth","transQuantity","transCost","propInRisky","propInRiskfree")
    returnList.without = list(simWealth.risky,simWealth.riskfree,simWealth,transQuantity,propInRisky,propInRiskfree)
    names(returnList.without) = stdNames[-5]
```

```
    returnList.pre = list(simWealth.risky.pre,simWealth.riskfree.pre,simWealth.
        pre,transQuantity.pre,transCost.pre,propInRisky.pre,propInRiskfree.pre)
    names(returnList.pre) = stdNames
    returnList.sub = list(simWealth.risky.sub,simWealth.riskfree.sub,simWealth.
        sub,transQuantity.sub,transCost.sub,propInRisky.sub,propInRiskfree.sub)
    names(returnList.sub) = stdNames
    returnList = list(days,rebDays,rebIndex,inc,BM,riskyPrice,riskfreePrice,
        thWealth,riskyReturn,riskfreeReturn,returnList.without,returnList.pre,
        returnList.sub)
    names(returnList) = c("days","rebDays","rebIndex","BM.increments","BM","
        riskyPrice","riskfreePrice","thWealth","riskyReturn","riskfreeReturn","
        withoutTransCost","precedingTransCost","subsequentTransCost")
  }
  else {
    stdNames = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","
        totalTransCost")
    returnList.th = list(thWealth.terminal,thWealth.sd,thWealth.logReturn.sd)
    names(returnList.th) = c("thWealth.terminal","thWealth.sd","thWealth.
        logReturn.sd")
    returnList.without = list(simWealth.terminal,simWealth.sd,simWealth.
        logReturn.sd)
    names(returnList.without) = stdNames[-4]
    returnList.pre = list(simWealth.pre.terminal,simWealth.pre.sd,simWealth.pre.
        logReturn.sd,totalTransCost.pre)
    names(returnList.pre) = stdNames
    returnList.sub = list(simWealth.sub.terminal,simWealth.sub.sd,simWealth.sub.
        logReturn.sd,totalTransCost.sub)
    names(returnList.sub) = stdNames
    returnList = list(returnList.th,returnList.without,returnList.pre,returnList
        .sub)
    names(returnList) = c("theoretical","noTransCost","precedingTransCost","
        subsequentTransCost")
  }

  return(returnList)
}
```

B.5.2 Execution

```
##
# Master thesis
# Simulation with transaction costs
# Two single runs
#

source("R/supportFunctions.R")
source("R/machinery_general.R")
source("R/initParameters.R")
source("R/machinery_transCost.R")

alpha = .05
qAlpha.half = qnorm(1-alpha/2)
graphics.off()

#
# Looking at difference in transaction cost
#

transCostDiffConstant.posDiff = function(lambda,uStar) { (lambda^2*uStar^2*(1-
    uStar)) / (1-lambda*uStar) }
```

```
transCostDiffConstant.negDiff = function(lambda,uStar) { (lambda^2*uStar^2*(1-
    uStar)) / (1+lambda*uStar) }

lambdaSeries = 0:350 / 10000

xTitle = expression(paste("Transaction cost proportion " * lambda))
yTitle = expression(italic(f(lambda * "|" * u * "*")) %*% 10^-4)
constant = transCostDiffConstant.posDiff(lambdaSeries,uStar) * 1e4
y.range = range(constant)
niceplot(lambdaSeries,constant,xTitle=xTitle,yTitle=yTitle,figsPerPage=3,y.
    superscript=T,nCol=2)
lambda = .01
lines(c(lambda,lambda),c(-1,constant[lambda*10000]),lty=3)
lines(c(-1,lambda),c(constant[lambda*10000],constant[lambda*10000]),lty=3)
text(0,constant[lambda*10000],substitute(paste(number %*% 10^-4),list(number=
    round(constant[lambda*10000],4),costProp=lambda)),adj=c(0,-.2),cex=.7)
lambda = .02
lines(c(lambda,lambda),c(-1,constant[lambda*10000]),lty=3)
lines(c(-1,lambda),c(constant[lambda*10000],constant[lambda*10000]),lty=3)
text(0,constant[lambda*10000],substitute(paste(number %*% 10^-4),list(number=
    round(constant[lambda*10000],4),costProp=lambda)),adj=c(0,-.2),cex=.7)
lambda = .03
lines(c(lambda,lambda),c(-1,constant[lambda*10000]),lty=3)
lines(c(-1,lambda),c(constant[lambda*10000],constant[lambda*10000]),lty=3)
text(0,constant[lambda*10000],substitute(paste(number %*% 10^-4),list(number=
    round(constant[lambda*10000],4),costProp=lambda)),adj=c(0,-.2),cex=.7)
legendText = expression(paste("(a)",~D[k]>=0))
legend("topleft",legendText,bty="n",cex=.7)

constant = transCostDiffConstant.negDiff(lambdaSeries,uStar) * 1e4
niceplot(lambdaSeries,constant,xTitle=xTitle,yTitle=yTitle,y.superscript=T,
    multiPlot=T,newDev=F,ylim=y.range)
lambda = .01
lines(c(lambda,lambda),c(-1,constant[lambda*10000]),lty=3)
lines(c(-1,lambda),c(constant[lambda*10000],constant[lambda*10000]),lty=3)
text(0,constant[lambda*10000],substitute(paste(number %*% 10^-4),list(number=
    round(constant[lambda*10000],4),costProp=lambda)),adj=c(0,-.2),cex=.7)
lambda = .02
lines(c(lambda,lambda),c(-1,constant[lambda*10000]),lty=3)
lines(c(-1,lambda),c(constant[lambda*10000],constant[lambda*10000]),lty=3)
text(0,constant[lambda*10000],substitute(paste(number %*% 10^-4),list(number=
    round(constant[lambda*10000],4),costProp=lambda)),adj=c(0,-.2),cex=.7)
lambda = .03
lines(c(lambda,lambda),c(-1,constant[lambda*10000]),lty=3)
lines(c(-1,lambda),c(constant[lambda*10000],constant[lambda*10000]),lty=3)
text(0,constant[lambda*10000],substitute(paste(number %*% 10^-4),list(number=
    round(constant[lambda*10000],4),costProp=lambda)),adj=c(0,-.2),cex=.7)
legendText = expression(paste("(b)",~D[k]<0))
legend("topleft",legendText,bty="n",cex=.7)

savePlot("images/transCostConstant",type="eps")

#
# One simulation run: strong risky asset development
#

nSims = 1

if (file.exists("Datasett/singleRun_transCost_01.RData")) {
  cat("Loading simulated portfolios...\n")
  load("Datasett/singleRun_transCost_01.RData")
} else {
  cat("Simulating portfolios...\n")
  simObject.01 = simPortfolio.transCost(nSims,paramSet,costProp=.01,loadBrownian
      =T)
```

```
  save(simObject.01,file="Datasett/singleRun_transCost_01.RData")
}

if (file.exists("Datasett/singleRun_transCost_02.RData")) {
  cat("Loading simulated portfolios...\n")
  load("Datasett/singleRun_transCost_02.RData")
} else {
  cat("Simulating portfolios...\n")
  simObject.02 = simPortfolio.transCost(nSims,paramSet,costProp=.02,loadBrownian=T)
  save(simObject.02,file="Datasett/singleRun_transCost_02.RData")
}

if (file.exists("Datasett/singleRun_transCost_03.RData")) {
  cat("Loading simulated portfolios...\n")
  load("Datasett/singleRun_transCost_03.RData")
} else {
  cat("Simulating portfolios...\n")
  simObject.03 = simPortfolio.transCost(nSims,paramSet,costProp=.03,loadBrownian=T)
  save(simObject.03,file="Datasett/singleRun_transCost_03.RData")
}

days = c(0,simObject.01$days)
rebDays = simObject.01$rebDays

# Plotting risky and risk free asset prices as benchmark
xTicks = c(0,rebDays)
xTitle = "Trading days"
yTitle = "Asset price"
riskyPrice = c(1,simObject.01$riskyPrice)
riskfreePrice = c(1,simObject.01$riskfreePrice)
niceplot(days,riskyPrice,xTicks,xTitle=xTitle,yTitle=yTitle,figsPerPage=5,y.superscript=T,horizLines=T,col="red")
nicelines(days,riskfreePrice,col="blue")
abline(v=xTicks,lty=3)
legendText = "(a)"
legend("topleft",legendText,bty="n",cex=.7)
savePlot("images/riskyPrice_risklessPrice",type="eps")

# Plotting risky and risk free asset period returns
rebIndex = simObject.01$rebIndex
yTitle = "Asset return during period"
y.max = max(c(0,simObject.01$riskyReturn[rebIndex]))
niceplot(xTicks,c(0,simObject.01$riskyReturn[rebIndex]),xTicks,xTitle=xTitle,yTitle=yTitle,figsPerPage=5,y.superscript=T,horizLines=T,col="red")
nicelines(xTicks,c(0,simObject.01$riskfreeReturn[rebIndex]),col="blue")
abline(v=xTicks,lty=3)
legendText = "(b)"
legend("topleft",legendText,bty="n",cex=.7)
savePlot("images/riskyReturn_risklessReturn",type="eps")

# Transaction cost differences, lambda = .01
costProp = .03
transCost.03.pre = abs(costProp * simObject.03$precedingTransCost$transQuantity[rebIndex])
transCost.03.sub = abs(costProp * simObject.03$subsequentTransCost$transQuantity[rebIndex])
transCost.03.diff = transCost.03.pre - transCost.03.sub
y.range = range(transCost.03.diff*1e5)
yTitle = expression(paste("Trans. cost difference",phantom(0) %*% 10^5))
niceplot(xTicks,c(0,transCost.03.diff*1e5),xTicks,xTitle=xTitle,yTitle=yTitle,figsPerPage=5,y.superscript=T)
abline(v=xTicks,lty=3)
abline(h=0,lty=3)
```

```
legendText = expression(paste("(e) ",lambda*"=.03"))
legend("topleft",legendText,bty="n",cex=.7)
savePlot("images/pre_sub_diff_03",type="eps")

# Transaction cost differences, lambda = .02
costProp = .01
transCost.01.pre = abs(costProp * simObject.01$precedingTransCost$transQuantity[
    rebIndex])
transCost.01.sub = abs(costProp * simObject.01$subsequentTransCost$transQuantity
    [rebIndex])
transCost.01.diff = transCost.01.pre - transCost.01.sub
niceplot(xTicks,c(0,transCost.01.diff*1e5),xTicks,xTitle=xTitle,yTitle=yTitle,
    figsPerPage=5,y.superscript=T,ylim=y.range)
abline(v=xTicks,lty=3)
abline(h=0,lty=3)
legendText = expression(paste("(c) ",lambda*"=.01"))
legend("topleft",legendText,bty="n",cex=.7)
savePlot("images/pre_sub_diff_01",type="eps")

# Transaction cost differences, lambda = .03
costProp = .02
transCost.02.pre = abs(costProp * simObject.02$precedingTransCost$transQuantity[
    rebIndex]) * 1e5
transCost.02.sub = abs(costProp * simObject.02$subsequentTransCost$transQuantity
    [rebIndex]) * 1e5
niceplot(xTicks,c(0,transCost.02.pre-transCost.02.sub),xTicks,xTitle=xTitle,
    yTitle=yTitle,figsPerPage=5,y.superscript=T,ylim=y.range)
abline(v=xTicks,lty=3)
abline(h=0,lty=3)
legendText = expression(paste("(d) ",lambda*"=.02"))
legend("topleft",legendText,bty="n",cex=.7)
savePlot("images/pre_sub_diff_02",type="eps")

# Transaction cost different ratio
transCost.03_01.diff.ratio = transCost.03.diff / transCost.01.diff
print(transCost.03_01.diff.ratio)
yTitle = expression(paste("Trans. cost ratio ",lambda==.03," vs ",lambda==.01))
niceplot(xTicks[-1],transCost.03_01.diff.ratio,xTicks,xTitle=xTitle,yTitle=
    yTitle)
abline(v=xTicks[-1],lty=3)
savePlot("images/transCost_diff_ratio",type="eps")

# Creating summarizing table
withoutTransCost = simObject.01$withoutTransCost
precedingTransCost.01 = simObject.01$precedingTransCost
subsequentTransCost.01 = simObject.01$subsequentTransCost

precedingTransCost.02 = simObject.02$precedingTransCost
subsequentTransCost.02 = simObject.02$subsequentTransCost

precedingTransCost.03 = simObject.03$precedingTransCost
subsequentTransCost.03 = simObject.03$subsequentTransCost

terminalWealth.th = last(simObject.01$thWealth)
terminalUtility.th = utility(terminalWealth.th,riskAversion)

terminalWealth.without = last(withoutTransCost$simWealth)
lossOfWealth.without = terminalWealth.th - terminalWealth.without
terminalUtility.without = utility(terminalWealth.without,riskAversion)
lossOfUtility.without = terminalUtility.th - terminalUtility.without

costProp = .01
terminalWealth.pre.01 = last(precedingTransCost.01$simWealth)
lossOfWealth.pre.01 = terminalWealth.th - terminalWealth.pre.01
terminalUtility.pre.01 = utility(terminalWealth.pre.01,riskAversion)
```

```
lossOfUtility.pre.01 = terminalUtility.th - terminalUtility.pre.01
totalTransCost.pre.01 = sum(precedingTransCost.01$transCost)
terminalWealth.sub.01 = last(subsequentTransCost.01$simWealth)
lossOfWealth.sub.01 = terminalWealth.th - terminalWealth.sub.01
terminalUtility.sub.01 = utility(terminalWealth.sub.01,riskAversion)
lossOfUtility.sub.01 = terminalUtility.th - terminalUtility.sub.01
totalTransCost.sub.01 = sum(subsequentTransCost.01$transCost)

costProp = .02
terminalWealth.pre.02 = last(precedingTransCost.02$simWealth)
lossOfWealth.pre.02 = terminalWealth.th - terminalWealth.pre.02
terminalUtility.pre.02 = utility(terminalWealth.pre.02,riskAversion)
lossOfUtility.pre.02 = terminalUtility.th - terminalUtility.pre.02
totalTransCost.pre.02 = sum(precedingTransCost.02$transCost)
terminalWealth.sub.02 = last(subsequentTransCost.02$simWealth)
lossOfWealth.sub.02 = terminalWealth.th - terminalWealth.sub.02
terminalUtility.sub.02 = utility(terminalWealth.sub.02,riskAversion)
lossOfUtility.sub.02 = terminalUtility.th - terminalUtility.sub.02
totalTransCost.sub.02 = sum(subsequentTransCost.02$transCost)

costProp = .03
terminalWealth.pre.03 = last(precedingTransCost.03$simWealth)
lossOfWealth.pre.03 = terminalWealth.th - terminalWealth.pre.03
terminalUtility.pre.03 = utility(terminalWealth.pre.03,riskAversion)
lossOfUtility.pre.03 = terminalUtility.th - terminalUtility.pre.03
totalTransCost.pre.03 = sum(precedingTransCost.03$transCost)
terminalWealth.sub.03 = last(subsequentTransCost.03$simWealth)
lossOfWealth.sub.03 = terminalWealth.th - terminalWealth.sub.03
terminalUtility.sub.03 = utility(terminalWealth.sub.03,riskAversion)
lossOfUtility.sub.03 = terminalUtility.th - terminalUtility.sub.03
totalTransCost.sub.03 = sum(subsequentTransCost.03$transCost)

tab = matrix(NA,8,5)
tab[1,] = c(terminalWealth.th,0,terminalUtility.th,0,0)
tab[2,] = c(terminalWealth.without,lossOfWealth.without,terminalUtility.without,
    lossOfUtility.without,0)
tab[3,] = c(terminalWealth.pre.01,lossOfWealth.pre.01,terminalUtility.pre.01,
    lossOfUtility.pre.01,totalTransCost.pre.01)
tab[4,] = c(terminalWealth.pre.02,lossOfWealth.pre.02,terminalUtility.pre.02,
    lossOfUtility.pre.02,totalTransCost.pre.02)
tab[5,] = c(terminalWealth.pre.03,lossOfWealth.pre.03,terminalUtility.pre.03,
    lossOfUtility.pre.03,totalTransCost.pre.03)
tab[6,] = c(terminalWealth.sub.01,lossOfWealth.sub.01,terminalUtility.sub.01,
    lossOfUtility.sub.01,totalTransCost.sub.01)
tab[7,] = c(terminalWealth.sub.02,lossOfWealth.sub.02,terminalUtility.sub.02,
    lossOfUtility.sub.02,totalTransCost.sub.02)
tab[8,] = c(terminalWealth.sub.03,lossOfWealth.sub.03,terminalUtility.sub.03,
    lossOfUtility.sub.03,totalTransCost.sub.03)

tab.orig = tab
print(tab)
tab[,2] = tab[,2] * 1e3
tab[,4:5] = tab[,4:5] * 1e3
tab = round(tab,4)
as.data.frame(tab)

for (k in 1:8) {
  tab[k,2] = paste(tab[k,2],"\\e{\\text-3}",sep="")
  tab[k,4] = paste(tab[k,4],"\\e{\\text-3}",sep="")
  tab[k,5] = paste(tab[k,5],"\\e{\\text-3}",sep="")
}
printex(tab)

#
# Second simulation run: weak risky asset development
```

```
#

if (file.exists("Datasett/singleRun2_transCost_01.RData")) {
  cat("Loading simulated portfolios...\n")
  load("Datasett/singleRun2_transCost_01.RData")
} else {
  cat("Simulating portfolios...\n")
  simObject.01 = simPortfolio.transCost(nSims,paramSet,costProp=.01,
      brownianFileName="Datasett/brownianIncrements2.RData")
  while (last(simObject.01$riskyPrice) > .75) {
    file.remove("Datasett/brownianIncrements2.RData")
    simObject.01 = simPortfolio.transCost(nSims,paramSet,costProp=.01,
        brownianFileName="Datasett/brownianIncrements2.RData")
  }
  save(simObject.01,file="Datasett/singleRun2_transCost_01.RData")
}

if (file.exists("Datasett/singleRun2_transCost_02.RData")) {
  cat("Loading simulated portfolios...\n")
  load("Datasett/singleRun2_transCost_02.RData")
} else {
  cat("Simulating portfolios...\n")
  simObject.02 = simPortfolio.transCost(nSims,paramSet,costProp=.02,
      brownianFileName="Datasett/brownianIncrements2.RData")
  save(simObject.02,file="Datasett/singleRun2_transCost_02.RData")
}

if (file.exists("Datasett/singleRun2_transCost_03.RData")) {
  cat("Loading simulated portfolios...\n")
  load("Datasett/singleRun2_transCost_03.RData")
} else {
  cat("Simulating portfolios...\n")
  simObject.03 = simPortfolio.transCost(nSims,paramSet,costProp=.03,
      brownianFileName="Datasett/brownianIncrements2.RData")
  save(simObject.03,file="Datasett/singleRun2_transCost_03.RData")
}

rebIndex = simObject.01$rebIndex
days = c(0,simObject.01$days)
rebDays = simObject.01$rebDays
xTicks = c(0,rebDays)
xTitle = "Trading days"

# Plotting risky and risk free asset prices as benchmark
yTitle = "Asset price"
riskyPrice = c(1,simObject.01$riskyPrice)
riskfreePrice = c(1,simObject.01$riskfreePrice)
niceplot(days,riskyPrice,xTicks,xTitle=xTitle,yTitle=yTitle,figsPerPage=5,y.
    superscript=T,horizLines=T,col="red")
nicelines(days,riskfreePrice,col="blue")
abline(v=xTicks,lty=3)
legendText = "(a)"
legend("topleft",legendText,bty="n",cex=.7)
savePlot("images/riskyPrice_risklessPrice2",type="eps")

# Plotting risky and risk free asset period returns
yTitle = "Asset return during period"
niceplot(xTicks,c(0,simObject.01$riskyReturn[rebIndex]),xTicks,xTitle=xTitle,
    yTitle=yTitle,figsPerPage=5,y.superscript=T,horizLines=T,col="red")
nicelines(xTicks,c(0,simObject.01$riskfreeReturn[rebIndex]),col="blue")
abline(v=xTicks,lty=3)
legendText = "(b)"
legend("topleft",legendText,bty="n",cex=.7)
savePlot("images/riskyReturn_risklessReturn2",type="eps")

```

```
# Creating summarizing table
withoutTransCost = simObject.01$withoutTransCost
precedingTransCost.01 = simObject.01$precedingTransCost
subsequentTransCost.01 = simObject.01$subsequentTransCost

precedingTransCost.02 = simObject.02$precedingTransCost
subsequentTransCost.02 = simObject.02$subsequentTransCost

precedingTransCost.03 = simObject.03$precedingTransCost
subsequentTransCost.03 = simObject.03$subsequentTransCost

terminalWealth.th = last(simObject.01$thWealth)
terminalUtility.th = utility(terminalWealth.th,riskAversion)

terminalWealth.without = last(withoutTransCost$simWealth)
lossOfWealth.without = terminalWealth.th - terminalWealth.without
terminalUtility.without = utility(terminalWealth.without,riskAversion)
lossOfUtility.without = terminalUtility.th - terminalUtility.without

costProp = .01
terminalWealth.pre.01 = last(precedingTransCost.01$simWealth)
lossOfWealth.pre.01 = terminalWealth.th - terminalWealth.pre.01
terminalUtility.pre.01 = utility(terminalWealth.pre.01,riskAversion)
lossOfUtility.pre.01 = terminalUtility.th - terminalUtility.pre.01
totalTransCost.pre.01 = sum(precedingTransCost.01$transCost)
terminalWealth.sub.01 = last(subsequentTransCost.01$simWealth)
lossOfWealth.sub.01 = terminalWealth.th - terminalWealth.sub.01
terminalUtility.sub.01 = utility(terminalWealth.sub.01,riskAversion)
lossOfUtility.sub.01 = terminalUtility.th - terminalUtility.sub.01
totalTransCost.sub.01 = sum(subsequentTransCost.01$transCost)

costProp = .02
terminalWealth.pre.02 = last(precedingTransCost.02$simWealth)
lossOfWealth.pre.02 = terminalWealth.th - terminalWealth.pre.02
terminalUtility.pre.02 = utility(terminalWealth.pre.02,riskAversion)
lossOfUtility.pre.02 = terminalUtility.th - terminalUtility.pre.02
totalTransCost.pre.02 = sum(precedingTransCost.02$transCost)
terminalWealth.sub.02 = last(subsequentTransCost.02$simWealth)
lossOfWealth.sub.02 = terminalWealth.th - terminalWealth.sub.02
terminalUtility.sub.02 = utility(terminalWealth.sub.02,riskAversion)
lossOfUtility.sub.02 = terminalUtility.th - terminalUtility.sub.02
totalTransCost.sub.02 = sum(subsequentTransCost.02$transCost)

costProp = .03
terminalWealth.pre.03 = last(precedingTransCost.03$simWealth)
lossOfWealth.pre.03 = terminalWealth.th - terminalWealth.pre.03
terminalUtility.pre.03 = utility(terminalWealth.pre.03,riskAversion)
lossOfUtility.pre.03 = terminalUtility.th - terminalUtility.pre.03
totalTransCost.pre.03 = sum(precedingTransCost.03$transCost)
terminalWealth.sub.03 = last(subsequentTransCost.03$simWealth)
lossOfWealth.sub.03 = terminalWealth.th - terminalWealth.sub.03
terminalUtility.sub.03 = utility(terminalWealth.sub.03,riskAversion)
lossOfUtility.sub.03 = terminalUtility.th - terminalUtility.sub.03
totalTransCost.sub.03 = sum(subsequentTransCost.03$transCost)

tab = matrix(NA,8,5)
tab[1,] = c(terminalWealth.th,0,terminalUtility.th,0,0)
tab[2,] = c(terminalWealth.without,lossOfWealth.without,terminalUtility.without,
    lossOfUtility.without,0)
tab[3,] = c(terminalWealth.pre.01,lossOfWealth.pre.01,terminalUtility.pre.01,
    lossOfUtility.pre.01,totalTransCost.pre.01)
tab[4,] = c(terminalWealth.pre.02,lossOfWealth.pre.02,terminalUtility.pre.02,
    lossOfUtility.pre.02,totalTransCost.pre.02)
tab[5,] = c(terminalWealth.pre.03,lossOfWealth.pre.03,terminalUtility.pre.03,
    lossOfUtility.pre.03,totalTransCost.pre.03)
```

```
tab[6,] = c(terminalWealth.sub.01,lossOfWealth.sub.01,terminalUtility.sub.01,
     lossOfUtility.sub.01,totalTransCost.sub.01)
tab[7,] = c(terminalWealth.sub.02,lossOfWealth.sub.02,terminalUtility.sub.02,
     lossOfUtility.sub.02,totalTransCost.sub.02)
tab[8,] = c(terminalWealth.sub.03,lossOfWealth.sub.03,terminalUtility.sub.03,
     lossOfUtility.sub.03,totalTransCost.sub.03)

tab.orig = tab
print(tab)
tab[,2] = tab[,2] * 1e3
tab[,4:5] = tab[,4:5] * 1e3
tab = round(tab,4)
as.data.frame(tab)

for (k in 1:8) {
  tab[k,2] = paste(tab[k,2],"\\e{\\text-3}",sep="")
  tab[k,4] = paste(tab[k,4],"\\e{\\text-3}",sep="")
  tab[k,5] = paste(tab[k,5],"\\e{\\text-3}",sep="")
}
printex(tab)
```

```
##
# Master thesis
# Simulations with transaction costs
# Loss of utility and Sharpe ratio
#

#
# Initialization
#

require(doSMP)
source("R/supportFunctions.R")
source("R/machinery_general.R")
source("R/initParameters.R")
source("R/machinery_transCost.R")

alpha = .05
qAlpha.half = qnorm(1-alpha/2)
graphics.off()

##
# Rebalancing strategy vs loss of utility
#

# Common parameter settings
nSims = 100000
nCores = 25
nDailyRebs = c(24,6,1,1/2,1/12,1/21,1/42,1/126,1/252)
strategyNames = c("Hourly","Every 4th hour","Daily","Every 3rd day","Every 12th
     day","Monthly","Bimonthly","Semiannually","Annually")

#
# Performing simulations, transaction cost proportion = .01
#

costProp = .01
paramSets.transCost = cbind(initWealth,nTradingDays,nDailyIncrements,nDailyRebs,
     drift,volatility,rent,riskAversion,uStar,costProp)
rebStrategy.transCost.01 = distribute(nSims,nCores,simPortfolio.transCost,
     paramSets.transCost)
names(rebStrategy.transCost.01) = strategyNames

# Organizing returned data
```

```
n.entries = length(rebStrategy.transCost.01)
for (k in 1:n.entries) {

  th = rebStrategy.transCost.01[[k]]$theoretical
  rebStrategy.transCost.01[[k]]$theoretical = list(merge.list(th[seq(1,3*nCores-2,3)]), merge.list(th[seq(2,3*nCores-1,3)]), merge.list(th[seq(3,3*nCores,3)]))
  names(rebStrategy.transCost.01[[k]]$theoretical) = c("thWealth.terminal","thWealth.sd","thWealth.logReturn.sd")

  no = rebStrategy.transCost.01[[k]]$noTransCost
  rebStrategy.transCost.01[[k]]$noTransCost = list(merge.list(no[seq(1,3*nCores-2,3)]), merge.list(no[seq(2,3*nCores-1,3)]), merge.list(no[seq(3,3*nCores,3)]))
  names(rebStrategy.transCost.01[[k]]$noTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd")

  pre = rebStrategy.transCost.01[[k]]$precedingTransCost
  rebStrategy.transCost.01[[k]]$precedingTransCost = list(merge.list(pre[seq(1,4*nCores-3,4)]), merge.list(pre[seq(2,4*nCores-2,4)]), merge.list(pre[seq(3,4*nCores-1,4)]), merge.list(pre[seq(4,4*nCores,4)]))
  names(rebStrategy.transCost.01[[k]]$precedingTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")

  sub = rebStrategy.transCost.01[[k]]$subsequentTransCost
  rebStrategy.transCost.01[[k]]$subsequentTransCost = list(merge.list(sub[seq(1,4*nCores-3,4)]), merge.list(sub[seq(2,4*nCores-2,4)]), merge.list(sub[seq(3,4*nCores-1,4)]), merge.list(sub[seq(4,4*nCores,4)]))
  names(rebStrategy.transCost.01[[k]]$subsequentTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")
}

save(rebStrategy.transCost.01, file="Datasett/rebStrategy_transCost_01.RData")

#
# Performing simulations, transaction cost proportion = .02
#

costProp = .02
paramSets.transCost = cbind(initWealth, nTradingDays, nDailyIncrements, nDailyRebs, drift, volatility, rent, riskAversion, uStar, costProp)
rebStrategy.transCost.02 = distribute(nSims, nCores, simPortfolio.transCost, paramSets.transCost)
names(rebStrategy.transCost.02) = strategyNames

n.entries = length(rebStrategy.transCost.02)
for (k in 1:n.entries) {

  th = rebStrategy.transCost.02[[k]]$theoretical
  rebStrategy.transCost.02[[k]]$theoretical = list(merge.list(th[seq(1,3*nCores-2,3)]), merge.list(th[seq(2,3*nCores-1,3)]), merge.list(th[seq(3,3*nCores,3)]))
  names(rebStrategy.transCost.02[[k]]$theoretical) = c("thWealth.terminal","thWealth.sd","thWealth.logReturn.sd")

  no = rebStrategy.transCost.02[[k]]$noTransCost
  rebStrategy.transCost.02[[k]]$noTransCost = list(merge.list(no[seq(1,3*nCores-2,3)]), merge.list(no[seq(2,3*nCores-1,3)]), merge.list(no[seq(3,3*nCores,3)]))
  names(rebStrategy.transCost.02[[k]]$noTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd")

  pre = rebStrategy.transCost.02[[k]]$precedingTransCost
  rebStrategy.transCost.02[[k]]$precedingTransCost = list(merge.list(pre[seq(1,4*nCores-3,4)]), merge.list(pre[seq(2,4*nCores-2,4)]), merge.list(pre[
```

```
      seq(3,4*nCores-1,4)]), merge.list(pre[seq(4,4*nCores,4)]))
  names(rebStrategy.transCost.02[[k]]$precedingTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")

  sub = rebStrategy.transCost.02[[k]]$subsequentTransCost
  rebStrategy.transCost.02[[k]]$subsequentTransCost = list(merge.list(sub[seq(1,4*nCores-3,4)]), merge.list(sub[seq(2,4*nCores-2,4)]), merge.list(sub[seq(3,4*nCores-1,4)]), merge.list(sub[seq(4,4*nCores,4)]))
  names(rebStrategy.transCost.02[[k]]$subsequentTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")
}

save(rebStrategy.transCost.02, file="Datasett/rebStrategy_transCost_02.RData")

#
# Performing simulations, transaction cost proportion = .03
#

costProp = .03
paramSets.transCost = cbind(initWealth, nTradingDays, nDailyIncrements, nDailyRebs, drift, volatility, rent, riskAversion, uStar, costProp)
rebStrategy.transCost.03 = distribute(nSims, nCores, simPortfolio.transCost, paramSets.transCost)
names(rebStrategy.transCost.03) = strategyNames
load("Datasett/rebStrategy_transCost_03.RData")

n.entries = length(rebStrategy.transCost.03)
for (k in 1:n.entries) {

  th = rebStrategy.transCost.03[[k]]$theoretical
  rebStrategy.transCost.03[[k]]$theoretical = list(merge.list(th[seq(1,3*nCores-2,3)]), merge.list(th[seq(2,3*nCores-1,3)]), merge.list(th[seq(3,3*nCores,3)]))
  names(rebStrategy.transCost.03[[k]]$theoretical) = c("thWealth.terminal","thWealth.sd","thWealth.logReturn.sd")

  no = rebStrategy.transCost.03[[k]]$noTransCost
  rebStrategy.transCost.03[[k]]$noTransCost = list(merge.list(no[seq(1,3*nCores-2,3)]), merge.list(no[seq(2,3*nCores-1,3)]), merge.list(no[seq(3,3*nCores,3)]))
  names(rebStrategy.transCost.03[[k]]$noTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd")

  pre = rebStrategy.transCost.03[[k]]$precedingTransCost
  rebStrategy.transCost.03[[k]]$precedingTransCost = list(merge.list(pre[seq(1,4*nCores-3,4)]), merge.list(pre[seq(2,4*nCores-2,4)]), merge.list(pre[seq(3,4*nCores-1,4)]), merge.list(pre[seq(4,4*nCores,4)]))
  names(rebStrategy.transCost.03[[k]]$precedingTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")

  sub = rebStrategy.transCost.03[[k]]$subsequentTransCost
  rebStrategy.transCost.03[[k]]$subsequentTransCost = list(merge.list(sub[seq(1,4*nCores-3,4)]), merge.list(sub[seq(2,4*nCores-2,4)]), merge.list(sub[seq(3,4*nCores-1,4)]), merge.list(sub[seq(4,4*nCores,4)]))
  names(rebStrategy.transCost.03[[k]]$subsequentTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")
}

save(rebStrategy.transCost.03 ,file="Datasett/rebStrategy_transCost_03.RData")

#
# Calculating relevant statistics and plotting
# Transaction cost proportion = .01
# Preceding transaction costs
#
```

```
x.labels = c(0,"10k","20k","30k","40k","50k","60k","70k","80k","90k","100k")
nn = 1:nSims
sel4 = c(1,3,6,9)

# Theoretical
terminalWealth.th.01 = matrix(NA,nSims,n.entries)
sdWealth.th.01 = matrix(NA,nSims,n.entries)
sdLogReturn.th.01 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.th.01[,k] = rebStrategy.transCost.01[[c(k,1)]]$thWealth.
      terminal
  sdWealth.th.01[,k] = rebStrategy.transCost.01[[c(k,1)]]$thWealth.sd
  sdLogReturn.th.01[,k] = rebStrategy.transCost.01[[c(k,1)]]$thWealth.logReturn.
      sd
}
colnames(terminalWealth.th.01) = strategyNames
terminalWealth.th.01.mean = colMeans(terminalWealth.th.01)
terminalWealth.th.01.mean.sel4 = terminalWealth.th.01.mean[sel4]
sdWealth.th.01.mean = colMeans(sdWealth.th.01)
sdWealth.th.01.mean.sel4 = sdWealth.th.01.mean[sel4]
terminalWealth.th.01.sd = colSds(terminalWealth.th.01)
terminalWealth.th.01.sd.sel4 = terminalWealth.th.01.sd[sel4]
terminalUtility.th.01 = utility(terminalWealth.th.01,riskAversion)
terminalUtility.th.01.mean = colMeans(terminalUtility.th.01)
terminalUtility.th.01.mean.sel4 = terminalUtility.th.01.mean[sel4]
terminalLogReturn.th.01 = log(terminalWealth.th.01)
terminalLogReturn.th.01.mean = colMeans(terminalLogReturn.th.01)
terminalLogReturn.th.01.mean.sel4 = terminalLogReturn.th.01.mean[sel4]
sdLogReturn.th.01.mean = colMeans(sdLogReturn.th.01)
sdLogReturn.th.01.mean.sel4 = sdLogReturn.th.01.mean[sel4]
annualizedSdLogReturn.th.01 = sdLogReturn.th.01 * sqrt(nTimePoints)
annualizedSdLogReturn.th.01.mean = colMeans(annualizedSdLogReturn.th.01)
terminalLogReturn.th.01.sd = colSds(terminalLogReturn.th.01)
terminalLogReturn.th.01.sd.sel4 = terminalLogReturn.th.01.sd[sel4]
excessReturn.th.01 = terminalLogReturn.th.01 - rent
sharpeRatio.th.01 = excessReturn.th.01 / (sqrt(nTimePoints)*sdLogReturn.th.01)
sharpeRatio.th.01.mean = colMeans(sharpeRatio.th.01)
sharpeRatio.th.01.mean.sel4 = sharpeRatio.th.01.mean[sel4]
volOfVol.th.01 = colSds(annualizedSdLogReturn.th.01)
correlation.th.01 = colCorrs(terminalLogReturn.th.01,annualizedSdLogReturn.th
    .01)

# Simulated, no transaction costs
terminalWealth.none.01 = matrix(NA,nSims,n.entries)
sdWealth.none.01 = matrix(NA,nSims,n.entries)
sdLogReturn.none.01 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.none.01[,k] = rebStrategy.transCost.01[[c(k,2)]]$simWealth.
      terminal
  sdWealth.none.01[,k] = rebStrategy.transCost.01[[c(k,2)]]$simWealth.sd
  sdLogReturn.none.01[,k] = rebStrategy.transCost.01[[c(k,2)]]$simWealth.
      logReturn.sd
}
colnames(terminalWealth.none.01) = strategyNames
terminalWealth.none.01.mean = colMeans(terminalWealth.none.01)
terminalWealth.none.01.mean.sel4 = terminalWealth.none.01.mean[sel4]
sdWealth.none.01.mean = colMeans(sdWealth.none.01)
sdWealth.none.01.mean.sel4 = sdWealth.none.01.mean[sel4]
terminalWealth.none.01.sd = colSds(terminalWealth.none.01)
terminalWealth.none.01.sd.sel4 = terminalWealth.none.01.sd[sel4]
lossOfWealth.none.01 = terminalWealth.th.01 - terminalWealth.none.01
lossOfWealth.none.01.mean = colMeans(lossOfWealth.none.01)
lossOfWealth.none.01.mean.sel4 = lossOfWealth.none.01.mean[sel4]
terminalUtility.none.01 = utility(terminalWealth.none.01,riskAversion)
```

```
terminalUtility.none.01.mean = colMeans(terminalUtility.none.01)
terminalUtility.none.01.mean.sel4 = terminalUtility.none.01.mean[sel4]
lossOfUtility.none.01 = terminalUtility.th.01 - terminalUtility.none.01
lossOfUtility.none.01.mean = colMeans(lossOfUtility.none.01)
lossOfUtility.none.01.mean.sel4 = lossOfUtility.none.01.mean[sel4]
lossOfUtility.none.01.sd = colSds(lossOfUtility.none.01)
terminalLogReturn.none.01 = log(terminalWealth.none.01)
terminalLogReturn.none.01.mean = colMeans(terminalLogReturn.none.01)
terminalLogReturn.none.01.mean.sel4 = terminalLogReturn.none.01.mean[sel4]
sdLogReturn.none.01.mean = colMeans(sdLogReturn.none.01)
sdLogReturn.none.01.mean.sel4 = sdLogReturn.none.01.mean[sel4]
annualizedSdLogReturn.none.01 = sdLogReturn.none.01 * sqrt(nTimePoints)
annualizedSdLogReturn.none.01.mean = colMeans(annualizedSdLogReturn.none.01)
terminalLogReturn.none.01.sd = colSds(terminalLogReturn.none.01)
terminalLogReturn.none.01.sd.sel4 = terminalLogReturn.none.01.sd[sel4]
excessReturn.none.01 = terminalLogReturn.none.01 - rent
sharpeRatio.none.01 = excessReturn.none.01 / (sqrt(nTimePoints)*sdLogReturn.none
    .01)
sharpeRatio.none.01.mean = colMeans(sharpeRatio.none.01)
sharpeRatio.none.01.mean.sel4 = sharpeRatio.none.01.mean[sel4]
volOfVol.none.01 = colSds(annualizedSdLogReturn.none.01)
correlation.none.01 = colCorrs(terminalLogReturn.none.01,annualizedSdLogReturn.
    none.01)

# Simulated, preceding transaction costs
terminalWealth.pre.01 = matrix(NA,nSims,n.entries)
sdWealth.pre.01 = matrix(NA,nSims,n.entries)
sdLogReturn.pre.01 = matrix(NA,nSims,n.entries)
totalTransCost.pre.01 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.pre.01[,k] = rebStrategy.transCost.01[[c(k,3)]]$simWealth.
      terminal
  sdWealth.pre.01[,k] = rebStrategy.transCost.01[[c(k,3)]]$simWealth.sd
  sdLogReturn.pre.01[,k] = rebStrategy.transCost.01[[c(k,3)]]$simWealth.
      logReturn.sd
  totalTransCost.pre.01[,k] = rebStrategy.transCost.01[[c(k,3)]]$totalTransCost
}
colnames(terminalWealth.pre.01) = strategyNames
terminalWealth.pre.01.mean = colMeans(terminalWealth.pre.01)
terminalWealth.pre.01.mean.sel4 = terminalWealth.pre.01.mean[sel4]
sdWealth.pre.01.mean = colMeans(sdWealth.pre.01)
sdWealth.pre.01.mean.sel4 = sdWealth.pre.01.mean[sel4]
terminalWealth.pre.01.sd = colSds(terminalWealth.pre.01)
terminalWealth.pre.01.sd.sel4 = terminalWealth.pre.01.sd[sel4]
lossOfWealth.pre.01 = terminalWealth.th.01 - terminalWealth.pre.01
lossOfWealth.pre.01.mean = colMeans(lossOfWealth.pre.01)
lossOfWealth.pre.01.mean.sel4 = lossOfWealth.pre.01.mean[sel4]
terminalUtility.pre.01 = utility(terminalWealth.pre.01,riskAversion)
terminalUtility.pre.01.mean = colMeans(terminalUtility.pre.01)
terminalUtility.pre.01.mean.sel4 = terminalUtility.pre.01.mean[sel4]
lossOfUtility.pre.01 = terminalUtility.th.01 - terminalUtility.pre.01
lossOfUtility.pre.01.sel4 = lossOfUtility.pre.01[,sel4]
lossOfUtility.pre.01.mean = colMeans(lossOfUtility.pre.01)
lossOfUtility.pre.01.mean.sel4 = lossOfUtility.pre.01.mean[sel4]
lossOfUtility.pre.01.sd = colSds(lossOfUtility.pre.01)
lossOfUtility.pre.01.cumMean = apply(lossOfUtility.pre.01,2,cumMean)
lossOfUtility.pre.01.cumMean.sel4 = lossOfUtility.pre.01.cumMean[,sel4]
lossOfUtility.pre.01.cumSd = apply(lossOfUtility.pre.01,2,cumSd)
lossOfUtility.pre.01.cumSd.sel4 = lossOfUtility.pre.01.cumSd[,sel4]
lossOfUtility.pre.01.sdCumMean = apply(lossOfUtility.pre.01.cumSd,2,function(x)
    {x/sqrt(nn)})
lossOfUtility.pre.01.sdCumMean.sel4 = lossOfUtility.pre.01.sdCumMean[,sel4]
lossOfUtility.pre.01.cumMean.lowerCL = lossOfUtility.pre.01.cumMean - qAlpha.
    half * lossOfUtility.pre.01.sdCumMean
lossOfUtility.pre.01.cumMean.upperCL = lossOfUtility.pre.01.cumMean + qAlpha.
```

```
half * lossOfUtility.pre.01.sdCumMean
lossOfUtility.pre.01.cumMean.lowerCL.sel4 = lossOfUtility.pre.01.cumMean.lowerCL[,sel4]
lossOfUtility.pre.01.cumMean.upperCL.sel4 = lossOfUtility.pre.01.cumMean.upperCL[,sel4]
terminalLogReturn.pre.01 = log(terminalWealth.pre.01)
terminalLogReturn.pre.01.mean = colMeans(terminalLogReturn.pre.01)
terminalLogReturn.pre.01.mean.sel4 = terminalLogReturn.pre.01.mean[sel4]
sdLogReturn.pre.01.mean = colMeans(sdLogReturn.pre.01)
sdLogReturn.pre.01.mean.sel4 = sdLogReturn.pre.01.mean[sel4]
annualizedSdLogReturn.pre.01 = sdLogReturn.pre.01 * sqrt(nTimePoints)
annualizedSdLogReturn.pre.01.mean = colMeans(annualizedSdLogReturn.pre.01)
terminalLogReturn.pre.01.sd = colSds(terminalLogReturn.pre.01)
terminalLogReturn.pre.01.sd.sel4 = terminalLogReturn.pre.01.sd[sel4]
excessReturn.pre.01 = terminalLogReturn.pre.01 - rent
sharpeRatio.pre.01 = excessReturn.pre.01 / (sqrt(nTimePoints)*sdLogReturn.pre.01)
sharpeRatio.pre.01.mean = colMeans(sharpeRatio.pre.01)
sharpeRatio.pre.01.mean.sel4 = sharpeRatio.pre.01.mean[sel4]
volOfVol.pre.01 = colSds(annualizedSdLogReturn.pre.01)
correlation.pre.01 = colCorrs(terminalLogReturn.pre.01,annualizedSdLogReturn.pre.01)
totalTransCost.pre.01.mean = colMeans(totalTransCost.pre.01)
totalTransCost.pre.01.mean.sel4 = totalTransCost.pre.01.mean[sel4]

y.rangeDiff.pre.01.sel4 = colRange(lossOfUtility.pre.01.cumMean.sel4)[2,] - colRange(lossOfUtility.pre.01.cumMean.sel4)[1,]
y.lim.pre.01.sel4 = rbind(lossOfUtility.pre.01.mean.sel4 - y.rangeDiff.pre.01.sel4/25,lossOfUtility.pre.01.mean.sel4 + y.rangeDiff.pre.01.sel4/25)

transformation = 1e2
y.title = expression(paste("Mean loss of utility",phantom(0) %*% 10^2))
niceplot(lossOfUtility.pre.01.cumMean.sel4[,1]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,nCol=2,horizLines=T,downsample=T,ylim=y.lim.pre.01.sel4[,1]*transformation)
nicelines(lossOfUtility.pre.01.cumMean.lowerCL.sel4[,1]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.pre.01.cumMean.upperCL.sel4[,1]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(a) ",lambda*"=.01")),"Transaction costs strategy : Preceding","Rebalancing strategy : Hourly"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)

niceplot(lossOfUtility.pre.01.cumMean.sel4[,2]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,multiPlot=T,newDev=F,horizLines=T,downsample=T,ylim=y.lim.pre.01.sel4[,2]*transformation*c(1,1.0001))
nicelines(lossOfUtility.pre.01.cumMean.lowerCL.sel4[,2]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.pre.01.cumMean.upperCL.sel4[,2]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(b) ",lambda*"=.01")),"Transaction costs strategy : Preceding","Rebalancing strategy : Daily"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_01_pre_Hourly_Daily",type="eps")

niceplot(lossOfUtility.pre.01.cumMean.sel4[,3]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,nCol=2,horizLines=T,downsample=T,ylim=y.lim.pre.01.sel4[,3]*transformation)
nicelines(lossOfUtility.pre.01.cumMean.lowerCL.sel4[,3]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.pre.01.cumMean.upperCL.sel4[,3]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(c) ",lambda*"=.01")),"Transaction costs strategy : Preceding","Rebalancing strategy : Monthly"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
```

```
niceplot(lossOfUtility.pre.01.cumMean.sel4[,4]*transformation,xLabels=x.labels,
    yTitle=y.title,figsPerPage=4,y.addCustom=.2,multiPlot=T,newDev=F,horizLines=
    T,downsample=T,ylim=y.lim.pre.01.sel4[,4]*transformation)
nicelines(lossOfUtility.pre.01.cumMean.lowerCL.sel4[,4]*transformation,
    downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.pre.01.cumMean.upperCL.sel4[,4]*transformation,
    downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(d) ",lambda*"=.01")),"Transaction costs
    strategy : Preceding","Rebalancing strategy : Annually"))
legendObject = nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_01_pre_Monthly_Annually",type="eps")

x.ticks = 1:9
x.title = "Rebalancing strategy"
niceplot(x.ticks,lossOfUtility.pre.01.mean*transformation,xLabels=strategyNames,
    xTitle=x.title,yTitle=y.title,y.addCustom=.2)
abline(v=x.ticks,lty=3)
legendText = expression(paste("(a) ",lambda*"=.01"))
nicelegend("left",legendText,horiz=T,bty="n",bg="white",cex=.7)
savePlot("images/rebStrategy_v_lossOfUtility_transCost_01",type="eps")

y.title = "Sharpe ratio"
niceplot(x.ticks,sharpeRatio.pre.01.mean,xLabels=strategyNames,xTitle=x.title,
    yTitle=y.title)
abline(v=x.ticks,lty=3)
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/rebStrategy_v_sharpeRatio_transCost_01",type="eps")

# Simulated, subsequent transaction costs
terminalWealth.sub.01 = matrix(NA,nSims,n.entries)
sdWealth.sub.01 = matrix(NA,nSims,n.entries)
sdLogReturn.sub.01 = matrix(NA,nSims,n.entries)
totalTransCost.sub.01 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.sub.01[,k] = rebStrategy.transCost.01[[c(k,4)]]$simWealth.
      terminal
  sdWealth.sub.01[,k] = rebStrategy.transCost.01[[c(k,4)]]$simWealth.sd
  sdLogReturn.sub.01[,k] = rebStrategy.transCost.01[[c(k,4)]]$simWealth.
      logReturn.sd
  totalTransCost.sub.01[,k] = rebStrategy.transCost.01[[c(k,4)]]$totalTransCost
}
colnames(terminalWealth.sub.01) = strategyNames
terminalWealth.sub.01.mean = colMeans(terminalWealth.sub.01)
terminalWealth.sub.01.mean.sel4 = terminalWealth.sub.01.mean[sel4]
sdWealth.sub.01.mean = colMeans(sdWealth.sub.01)
sdWealth.sub.01.mean.sel4 = sdWealth.sub.01.mean[sel4]
terminalWealth.sub.01.sd = colSds(terminalWealth.sub.01)
terminalWealth.sub.01.sd.sel4 = terminalWealth.sub.01.sd[sel4]
lossOfWealth.sub.01 = terminalWealth.th.01 - terminalWealth.sub.01
lossOfWealth.sub.01.mean = colMeans(lossOfWealth.sub.01)
lossOfWealth.sub.01.mean.sel4 = lossOfWealth.sub.01.mean[sel4]
terminalUtility.sub.01 = utility(terminalWealth.sub.01,riskAversion)
terminalUtility.sub.01.mean = colMeans(terminalUtility.sub.01)
terminalUtility.sub.01.mean.sel4 = terminalUtility.sub.01.mean[sel4]
lossOfUtility.sub.01 = terminalUtility.th.01 - terminalUtility.sub.01
lossOfUtility.sub.01.sel4 = lossOfUtility.sub.01[,sel4]
lossOfUtility.sub.01.mean = colMeans(lossOfUtility.sub.01)
lossOfUtility.sub.01.mean.sel4 = lossOfUtility.sub.01.mean[sel4]
lossOfUtility.sub.01.sd = colSds(lossOfUtility.sub.01)
lossOfUtility.sub.01.cumMean = apply(lossOfUtility.sub.01,2,cumMean)
lossOfUtility.sub.01.cumMean.sel4 = lossOfUtility.sub.01.cumMean[,sel4]
lossOfUtility.sub.01.cumSd = apply(lossOfUtility.sub.01,2,cumSd)
lossOfUtility.sub.01.cumSd.sel4 = lossOfUtility.sub.01.cumSd[,sel4]
lossOfUtility.sub.01.sdCumMean = apply(lossOfUtility.sub.01.cumSd,2,function(x)
```

```
    {x/sqrt(nn)})
lossOfUtility.sub.01.sdCumMean.sel4 = lossOfUtility.sub.01.sdCumMean[,sel4]
lossOfUtility.sub.01.cumMean.lowerCL = lossOfUtility.sub.01.cumMean - qAlpha.half * lossOfUtility.sub.01.sdCumMean
lossOfUtility.sub.01.cumMean.upperCL = lossOfUtility.sub.01.cumMean + qAlpha.half * lossOfUtility.sub.01.sdCumMean
lossOfUtility.sub.01.cumMean.lowerCL.sel4 = lossOfUtility.sub.01.cumMean.lowerCL[,sel4]
lossOfUtility.sub.01.cumMean.upperCL.sel4 = lossOfUtility.sub.01.cumMean.upperCL[,sel4]
terminalLogReturn.sub.01 = log(terminalWealth.sub.01)
terminalLogReturn.sub.01.mean = colMeans(terminalLogReturn.sub.01)
terminalLogReturn.sub.01.mean.sel4 = terminalLogReturn.sub.01.mean[sel4]
sdLogReturn.sub.01.mean = colMeans(sdLogReturn.sub.01)
sdLogReturn.sub.01.mean.sel4 = sdLogReturn.sub.01.mean[sel4]
annualizedSdLogReturn.sub.01 = sdLogReturn.sub.01 * sqrt(nTimePoints)
annualizedSdLogReturn.sub.01.mean = colMeans(annualizedSdLogReturn.sub.01)
terminalLogReturn.sub.01.sd = colSds(terminalLogReturn.sub.01)
terminalLogReturn.sub.01.sd.sel4 = terminalLogReturn.sub.01.sd[sel4]
excessReturn.sub.01 = terminalLogReturn.sub.01 - rent
sharpeRatio.sub.01 = excessReturn.sub.01 / (sqrt(nTimePoints)*sdLogReturn.sub.01)
sharpeRatio.sub.01.mean = colMeans(sharpeRatio.sub.01)
sharpeRatio.sub.01.mean.sel4 = sharpeRatio.sub.01.mean[sel4]
volOfVol.sub.01 = colSds(annualizedSdLogReturn.sub.01)
correlation.sub.01 = colCorrs(terminalLogReturn.sub.01,annualizedSdLogReturn.sub.01)
totalTransCost.sub.01.mean = colMeans(totalTransCost.sub.01)
totalTransCost.sub.01.mean.sel4 = totalTransCost.sub.01.mean[sel4]

# Plotting
y.rangeDiff.sub.01.sel4 = colRange(lossOfUtility.sub.01.cumMean.sel4)[2,] - colRange(lossOfUtility.sub.01.cumMean.sel4)[1,]
y.lim.sub.01.sel4 = rbind(lossOfUtility.sub.01.mean.sel4 - y.rangeDiff.sub.01.sel4/25,lossOfUtility.sub.01.mean.sel4 + y.rangeDiff.sub.01.sel4/25)

transformation = 1e2
y.title = expression(paste("Mean loss of utility",phantom(0) %*% 10^2))
niceplot(lossOfUtility.sub.01.cumMean.sel4[,1]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,nCol=2,horizLines=T,downsample=T,ylim=y.lim.sub.01.sel4[,1]*transformation)
nicelines(lossOfUtility.sub.01.cumMean.lowerCL.sel4[,1]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.sub.01.cumMean.upperCL.sel4[,1]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(e) ",lambda*"=.01")),"Transaction costs strategy : Subsequent","Rebalancing strategy : Hourly"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)

niceplot(lossOfUtility.sub.01.cumMean.sel4[,2]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,multiPlot=T,newDev=F,horizLines=T,downsample=T,ylim=y.lim.sub.01.sel4[,2]*transformation)
nicelines(lossOfUtility.sub.01.cumMean.lowerCL.sel4[,2]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.sub.01.cumMean.upperCL.sel4[,2]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(f) ",lambda*"=.01")),"Transaction costs strategy : Subsequent","Rebalancing strategy : Daily"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_01_sub_Hourly_Daily",type="eps")

niceplot(lossOfUtility.sub.01.cumMean.sel4[,3]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,nCol=2,horizLines=T,downsample=T,ylim=y.lim.sub.01.sel4[,3]*transformation)
nicelines(lossOfUtility.sub.01.cumMean.lowerCL.sel4[,3]*transformation,
```

```
    downsample=T, col="darkgray", lty=3)
nicelines(lossOfUtility.sub.01.cumMean.upperCL.sel4[,3]*transformation, downsample=T, col="darkgray", lty=3)
legendText = c(expression(paste("(g) ",lambda*"=.01")),"Transaction costs strategy : Subsequent","Rebalancing strategy : Monthly"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)

niceplot(lossOfUtility.sub.01.cumMean.sel4[,4]*transformation,xLabels=x.labels, yTitle=y.title,figsPerPage=4,y.addCustom=.2,multiPlot=T,newDev=F,horizLines=T,downsample=T,ylim=y.lim.sub.01.sel4[,4]*transformation)
nicelines(lossOfUtility.sub.01.cumMean.lowerCL.sel4[,4]*transformation, downsample=T, col="darkgray", lty=3)
nicelines(lossOfUtility.sub.01.cumMean.upperCL.sel4[,4]*transformation, downsample=T, col="darkgray", lty=3)
legendText = c(expression(paste("(h) ",lambda*"=.01")),"Transaction costs strategy : Subsequent","Rebalancing strategy : Annually"))
legendObject = nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_01_sub_Monthly_Annually",type="eps")

# Setting up summarizing tables
tab1 = matrix(NA,36,5)
for (k in 1:9) {
  tab1[k*4-3,] = c(terminalWealth.th.01.mean[k],0,terminalUtility.th.01.mean[k],0,0)
  tab1[k*4-2,] = c(terminalWealth.none.01.mean[k],0,terminalUtility.none.01.mean[k],lossOfUtility.none.01.mean[k],lossOfUtility.none.01.sd[k])
  tab1[k*4-1,] = c(terminalWealth.pre.01.mean[k],totalTransCost.pre.01.mean[k],terminalUtility.pre.01.mean[k],lossOfUtility.pre.01.mean[k],lossOfUtility.pre.01.sd[k])
  tab1[k*4,]   = c(terminalWealth.sub.01.mean[k],totalTransCost.sub.01.mean[k],terminalUtility.sub.01.mean[k],lossOfUtility.sub.01.mean[k],lossOfUtility.sub.01.sd[k])
}

tab1[,2] = tab1[,2] * 1e2
tab1[,4] = tab1[,4] * 1e2
tab1[,5] = tab1[,5] * 1e3

tab1 = round(tab1,4)

for (k in 1:36) {
  tab1[k,2] = paste(tab1[k,2],"\\e{\\text-2}",sep="")
  tab1[k,4] = paste(tab1[k,4],"\\e{\\text-2}",sep="")
  tab1[k,5] = paste(tab1[k,5],"\\e{\\text-3}",sep="")
}

printex(tab1)

tab2 = matrix(NA,36,5)
for (k in 1:9) {
  tab2[k*4-3,] = c(terminalLogReturn.th.01.mean[k],annualizedSdLogReturn.th.01.mean[k],sharpeRatio.th.01.mean[k],volOfVol.th.01[k],correlation.th.01[k])
  tab2[k*4-2,] = c(terminalLogReturn.none.01.mean[k],annualizedSdLogReturn.none.01.mean[k],sharpeRatio.none.01.mean[k],volOfVol.none.01[k],correlation.none.01[k])
  tab2[k*4-1,] = c(terminalLogReturn.pre.01.mean[k],annualizedSdLogReturn.pre.01.mean[k],sharpeRatio.pre.01.mean[k],volOfVol.pre.01[k],correlation.pre.01[k])
  tab2[k*4,]   = c(terminalLogReturn.sub.01.mean[k],annualizedSdLogReturn.sub.01.mean[k],sharpeRatio.sub.01.mean[k],volOfVol.sub.01[k],correlation.sub.01[k])
}

tab2[,1] = tab2[,1] * 1e2
tab2[,4] = tab2[,4] * 1e3
```

```

tab2 = round(tab2,4)

for (k in 1:36) {
  tab2[k,1] = paste(tab2[k,1],"\\e{\\text-2}",sep="")
  tab2[k,4] = paste(tab2[k,4],"\\e{\\text-3}",sep="")
}

printex(tab2)

#
# Calculating relevant statistics and plotting
# Transaction cost proportion = .02
# Preceding transaction costs
#

# Theoretical
terminalWealth.th.02 = matrix(NA,nSims,n.entries)
sdWealth.th.02 = matrix(NA,nSims,n.entries)
sdLogReturn.th.02 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.th.02[,k] = rebStrategy.transCost.02[[c(k,1)]]$thWealth.terminal
  sdWealth.th.02[,k] = rebStrategy.transCost.02[[c(k,1)]]$thWealth.sd
  sdLogReturn.th.02[,k] = rebStrategy.transCost.02[[c(k,1)]]$thWealth.logReturn.sd
}
colnames(terminalWealth.th.02) = strategyNames
terminalWealth.th.02.mean = colMeans(terminalWealth.th.02)
terminalWealth.th.02.mean.sel4 = terminalWealth.th.02.mean[sel4]
sdWealth.th.02.mean = colMeans(sdWealth.th.02)
sdWealth.th.02.mean.sel4 = sdWealth.th.02.mean[sel4]
terminalWealth.th.02.sd = colSds(terminalWealth.th.02)
terminalWealth.th.02.sd.sel4 = terminalWealth.th.02.sd[sel4]
terminalUtility.th.02 = utility(terminalWealth.th.02,riskAversion)
terminalUtility.th.02.mean = colMeans(terminalUtility.th.02)
terminalUtility.th.02.mean.sel4 = terminalUtility.th.02.mean[sel4]
terminalLogReturn.th.02 = log(terminalWealth.th.02)
terminalLogReturn.th.02.mean = colMeans(terminalLogReturn.th.02)
terminalLogReturn.th.02.mean.sel4 = terminalLogReturn.th.02.mean[sel4]
sdLogReturn.th.02.mean = colMeans(sdLogReturn.th.02)
sdLogReturn.th.02.mean.sel4 = sdLogReturn.th.02.mean[sel4]
annualizedSdLogReturn.th.02 = sdLogReturn.th.02 * sqrt(nTimePoints)
annualizedSdLogReturn.th.02.mean = colMeans(annualizedSdLogReturn.th.02)
terminalLogReturn.th.02.sd = colSds(terminalLogReturn.th.02)
terminalLogReturn.th.02.sd.sel4 = terminalLogReturn.th.02.sd[sel4]
excessReturn.th.02 = terminalLogReturn.th.02 - rent
sharpeRatio.th.02 = excessReturn.th.02 / (sqrt(nTimePoints)*sdLogReturn.th.02)
sharpeRatio.th.02.mean = colMeans(sharpeRatio.th.02)
sharpeRatio.th.02.mean.sel4 = sharpeRatio.th.02.mean[sel4]
volOfVol.th.02 = colSds(annualizedSdLogReturn.th.02)
correlation.th.02 = colCorrs(terminalLogReturn.th.02,annualizedSdLogReturn.th.02)

# Simulated, no transaction costs
terminalWealth.none.02 = matrix(NA,nSims,n.entries)
sdWealth.none.02 = matrix(NA,nSims,n.entries)
sdLogReturn.none.02 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.none.02[,k] = rebStrategy.transCost.02[[c(k,2)]]$simWealth.terminal
  sdWealth.none.02[,k] = rebStrategy.transCost.02[[c(k,2)]]$simWealth.sd
  sdLogReturn.none.02[,k] = rebStrategy.transCost.02[[c(k,2)]]$simWealth.logReturn.sd
}
```

```
colnames(terminalWealth.none.02) = strategyNames
terminalWealth.none.02.mean = colMeans(terminalWealth.none.02)
terminalWealth.none.02.mean.sel4 = terminalWealth.none.02.mean[sel4]
sdWealth.none.02.mean = colMeans(sdWealth.none.02)
sdWealth.none.02.mean.sel4 = sdWealth.none.02.mean[sel4]
terminalWealth.none.02.sd = colSds(terminalWealth.none.02)
terminalWealth.none.02.sd.sel4 = terminalWealth.none.02.sd[sel4]
lossOfWealth.none.02 = terminalWealth.th.02 - terminalWealth.none.02
lossOfWealth.none.02.mean = colMeans(lossOfWealth.none.02)
lossOfWealth.none.02.mean.sel4 = lossOfWealth.none.02.mean[sel4]
terminalUtility.none.02 = utility(terminalWealth.none.02,riskAversion)
terminalUtility.none.02.mean = colMeans(terminalUtility.none.02)
terminalUtility.none.02.mean.sel4 = terminalUtility.none.02.mean[sel4]
lossOfUtility.none.02 = terminalUtility.th.02 - terminalUtility.none.02
lossOfUtility.none.02.mean = colMeans(lossOfUtility.none.02)
lossOfUtility.none.02.mean.sel4 = lossOfUtility.none.02.mean[sel4]
lossOfUtility.none.02.sd = colSds(lossOfUtility.none.02)
terminalLogReturn.none.02 = log(terminalWealth.none.02)
terminalLogReturn.none.02.mean = colMeans(terminalLogReturn.none.02)
terminalLogReturn.none.02.mean.sel4 = terminalLogReturn.none.02.mean[sel4]
sdLogReturn.none.02.mean = colMeans(sdLogReturn.none.02)
sdLogReturn.none.02.mean.sel4 = sdLogReturn.none.02.mean[sel4]
annualizedSdLogReturn.none.02 = sdLogReturn.none.02 * sqrt(nTimePoints)
annualizedSdLogReturn.none.02.mean = colMeans(annualizedSdLogReturn.none.02)
terminalLogReturn.none.02.sd = colSds(terminalLogReturn.none.02)
terminalLogReturn.none.02.sd.sel4 = terminalLogReturn.none.02.sd[sel4]
excessReturn.none.02 = terminalLogReturn.none.02 - rent
sharpeRatio.none.02 = excessReturn.none.02 / (sqrt(nTimePoints)*sdLogReturn.none
    .02)
sharpeRatio.none.02.mean = colMeans(sharpeRatio.none.02)
sharpeRatio.none.02.mean.sel4 = sharpeRatio.none.02.mean[sel4]
volOfVol.none.02 = colSds(annualizedSdLogReturn.none.02)
correlation.none.02 = colCorrs(terminalLogReturn.none.02,annualizedSdLogReturn.
    none.02)

# Simulated, preceding transaction costs
terminalWealth.pre.02 = matrix(NA,nSims,n.entries)
sdWealth.pre.02 = matrix(NA,nSims,n.entries)
sdLogReturn.pre.02 = matrix(NA,nSims,n.entries)
totalTransCost.pre.02 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.pre.02[,k] = rebStrategy.transCost.02[[c(k,3)]]$simWealth.
      terminal
  sdWealth.pre.02[,k] = rebStrategy.transCost.02[[c(k,3)]]$simWealth.sd
  sdLogReturn.pre.02[,k] = rebStrategy.transCost.02[[c(k,3)]]$simWealth.
      logReturn.sd
  totalTransCost.pre.02[,k] = rebStrategy.transCost.02[[c(k,3)]]$totalTransCost
}
colnames(terminalWealth.pre.02) = strategyNames
terminalWealth.pre.02.mean = colMeans(terminalWealth.pre.02)
terminalWealth.pre.02.mean.sel4 = terminalWealth.pre.02.mean[sel4]
sdWealth.pre.02.mean = colMeans(sdWealth.pre.02)
sdWealth.pre.02.mean.sel4 = sdWealth.pre.02.mean[sel4]
terminalWealth.pre.02.sd = colSds(terminalWealth.pre.02)
terminalWealth.pre.02.sd.sel4 = terminalWealth.pre.02.sd[sel4]
lossOfWealth.pre.02 = terminalWealth.th.02 - terminalWealth.pre.02
lossOfWealth.pre.02.mean = colMeans(lossOfWealth.pre.02)
lossOfWealth.pre.02.mean.sel4 = lossOfWealth.pre.02.mean[sel4]
terminalUtility.pre.02 = utility(terminalWealth.pre.02,riskAversion)
terminalUtility.pre.02.mean = colMeans(terminalUtility.pre.02)
terminalUtility.pre.02.mean.sel4 = terminalUtility.pre.02.mean[sel4]
lossOfUtility.pre.02 = terminalUtility.th.02 - terminalUtility.pre.02
lossOfUtility.pre.02.sel4 = lossOfUtility.pre.02[,sel4]
lossOfUtility.pre.02.mean = colMeans(lossOfUtility.pre.02)
lossOfUtility.pre.02.mean.sel4 = lossOfUtility.pre.02.mean[sel4]
```

```
lossOfUtility.pre.02.sd = colSds(lossOfUtility.pre.02)
lossOfUtility.pre.02.cumMean = apply(lossOfUtility.pre.02,2,cumMean)
lossOfUtility.pre.02.cumMean.sel4 = lossOfUtility.pre.02.cumMean[,sel4]
lossOfUtility.pre.02.cumSd = apply(lossOfUtility.pre.02,2,cumSd)
lossOfUtility.pre.02.cumSd.sel4 = lossOfUtility.pre.02.cumSd[,sel4]
lossOfUtility.pre.02.sdCumMean = apply(lossOfUtility.pre.02.cumSd,2,function(x){x/sqrt(nn)})
lossOfUtility.pre.02.sdCumMean.sel4 = lossOfUtility.pre.02.sdCumMean[,sel4]
lossOfUtility.pre.02.cumMean.lowerCL = lossOfUtility.pre.02.cumMean - qAlpha.half * lossOfUtility.pre.02.sdCumMean
lossOfUtility.pre.02.cumMean.upperCL = lossOfUtility.pre.02.cumMean + qAlpha.half * lossOfUtility.pre.02.sdCumMean
lossOfUtility.pre.02.cumMean.lowerCL.sel4 = lossOfUtility.pre.02.cumMean.lowerCL[,sel4]
lossOfUtility.pre.02.cumMean.upperCL.sel4 = lossOfUtility.pre.02.cumMean.upperCL[,sel4]
terminalLogReturn.pre.02 = log(terminalWealth.pre.02)
terminalLogReturn.pre.02.mean = colMeans(terminalLogReturn.pre.02)
terminalLogReturn.pre.02.mean.sel4 = terminalLogReturn.pre.02.mean[sel4]
sdLogReturn.pre.02.mean = colMeans(sdLogReturn.pre.02)
sdLogReturn.pre.02.mean.sel4 = sdLogReturn.pre.02.mean[sel4]
annualizedSdLogReturn.pre.02 = sdLogReturn.pre.02 * sqrt(nTimePoints)
annualizedSdLogReturn.pre.02.mean = colMeans(annualizedSdLogReturn.pre.02)
terminalLogReturn.pre.02.sd = colSds(terminalLogReturn.pre.02)
terminalLogReturn.pre.02.sd.sel4 = terminalLogReturn.pre.02.sd[sel4]
excessReturn.pre.02 = terminalLogReturn.pre.02 - rent
sharpeRatio.pre.02 = excessReturn.pre.02 / (sqrt(nTimePoints)*sdLogReturn.pre.02)
sharpeRatio.pre.02.mean = colMeans(sharpeRatio.pre.02)
sharpeRatio.pre.02.mean.sel4 = sharpeRatio.pre.02.mean[sel4]
volOfVol.pre.02 = colSds(annualizedSdLogReturn.pre.02)
correlation.pre.02 = colCorrs(terminalLogReturn.pre.02,annualizedSdLogReturn.pre.02)
totalTransCost.pre.02.mean = colMeans(totalTransCost.pre.02)
totalTransCost.pre.02.mean.sel4 = totalTransCost.pre.02.mean[sel4]

y.rangeDiff.pre.02.sel4 = colRange(lossOfUtility.pre.02.cumMean.sel4)[2,] - colRange(lossOfUtility.pre.02.cumMean.sel4)[1,]
y.lim.pre.02.sel4 = rbind(lossOfUtility.pre.02.mean.sel4 - y.rangeDiff.pre.02.sel4/25,lossOfUtility.pre.02.mean.sel4 + y.rangeDiff.pre.02.sel4/25)

transformation = 1e2
y.title = expression(paste("Mean loss of utility",phantom(0) %*% 10^2))
niceplot(lossOfUtility.pre.02.cumMean.sel4[,1]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,nCol=2,horizLines=T,downsample=T,ylim=y.lim.pre.02.sel4[,1]*transformation)
nicelines(lossOfUtility.pre.02.cumMean.lowerCL.sel4[,1]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.pre.02.cumMean.upperCL.sel4[,1]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(a) ",lambda*"=.02")),"Transaction costs strategy : Preceding","Rebalancing strategy : Hourly"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)

niceplot(lossOfUtility.pre.02.cumMean.sel4[,2]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,multiPlot=T,newDev=F,horizLines=T,downsample=T,ylim=y.lim.pre.02.sel4[,2]*transformation*c(1,1.0001))
nicelines(lossOfUtility.pre.02.cumMean.lowerCL.sel4[,2]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.pre.02.cumMean.upperCL.sel4[,2]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(b) ",lambda*"=.02")),"Transaction costs strategy : Preceding","Rebalancing strategy : Daily"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_02_pre_Hourly_Daily",type="eps")
```

```
niceplot(lossOfUtility.pre.02.cumMean.sel4[,3]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,nCol=2,horizLines=T,downsample=T,ylim=y.lim.pre.02.sel4[,3]*transformation)
nicelines(lossOfUtility.pre.02.cumMean.lowerCL.sel4[,3]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.pre.02.cumMean.upperCL.sel4[,3]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(c) ",lambda*"=.02")),"Transaction costs strategy : Preceding","Rebalancing strategy : Monthly"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)

niceplot(lossOfUtility.pre.02.cumMean.sel4[,4]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,multiPlot=T,newDev=F,horizLines=T,downsample=T,ylim=y.lim.pre.02.sel4[,4]*transformation)
nicelines(lossOfUtility.pre.02.cumMean.lowerCL.sel4[,4]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.pre.02.cumMean.upperCL.sel4[,4]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(d) ",lambda*"=.02")),"Transaction costs strategy : Preceding","Rebalancing strategy : Annually"))
legendObject = nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_02_pre_Monthly_Annually",type="eps")

x.ticks = 1:9
x.title = "Rebalancing strategy"
niceplot(x.ticks,lossOfUtility.pre.02.mean*transformation,xLabels=strategyNames,xTitle=x.title,yTitle=y.title,y.addCustom=.2)
abline(v=x.ticks,lty=3)
legendText = expression(paste("(b) ",lambda*"=.02"))
nicelegend("left",legendText,horiz=T,bty="n",bg="white",cex=.7)
savePlot("images/rebStrategy_v_lossOfUtility_transCost_02",type="eps")

y.title = "Sharpe ratio"
niceplot(x.ticks,sharpeRatio.pre.02.mean,xLabels=strategyNames,xTitle=x.title,yTitle=y.title)
abline(v=x.ticks,lty=3)
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/rebStrategy_v_sharpeRatio_transCost_02",type="eps")

# Simulated, subsequent transaction costs
terminalWealth.sub.02 = matrix(NA,nSims,n.entries)
sdWealth.sub.02 = matrix(NA,nSims,n.entries)
sdLogReturn.sub.02 = matrix(NA,nSims,n.entries)
totalTransCost.sub.02 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.sub.02[,k] = rebStrategy.transCost.02[[c(k,4)]]$simWealth.terminal
  sdWealth.sub.02[,k] = rebStrategy.transCost.02[[c(k,4)]]$simWealth.sd
  sdLogReturn.sub.02[,k] = rebStrategy.transCost.02[[c(k,4)]]$simWealth.logReturn.sd
  totalTransCost.sub.02[,k] = rebStrategy.transCost.02[[c(k,4)]]$totalTransCost
}
colnames(terminalWealth.sub.02) = strategyNames
terminalWealth.sub.02.mean = colMeans(terminalWealth.sub.02)
terminalWealth.sub.02.mean.sel4 = terminalWealth.sub.02.mean[sel4]
sdWealth.sub.02.mean = colMeans(sdWealth.sub.02)
sdWealth.sub.02.mean.sel4 = sdWealth.sub.02.mean[sel4]
terminalWealth.sub.02.sd = colSds(terminalWealth.sub.02)
terminalWealth.sub.02.sd.sel4 = terminalWealth.sub.02.sd[sel4]
lossOfWealth.sub.02 = terminalWealth.th.02 - terminalWealth.sub.02
lossOfWealth.sub.02.mean = colMeans(lossOfWealth.sub.02)
lossOfWealth.sub.02.mean.sel4 = lossOfWealth.sub.02.mean[sel4]
terminalUtility.sub.02 = utility(terminalWealth.sub.02,riskAversion)
terminalUtility.sub.02.mean = colMeans(terminalUtility.sub.02)
```

```
terminalUtility.sub.02.mean.sel4 = terminalUtility.sub.02.mean[sel4]
lossOfUtility.sub.02 = terminalUtility.th.02 - terminalUtility.sub.02
lossOfUtility.sub.02.sel4 = lossOfUtility.sub.02[,sel4]
lossOfUtility.sub.02.mean = colMeans(lossOfUtility.sub.02)
lossOfUtility.sub.02.mean.sel4 = lossOfUtility.sub.02.mean[sel4]
lossOfUtility.sub.02.sd = colSds(lossOfUtility.sub.02)
lossOfUtility.sub.02.cumMean = apply(lossOfUtility.sub.02,2,cumMean)
lossOfUtility.sub.02.cumMean.sel4 = lossOfUtility.sub.02.cumMean[,sel4]
lossOfUtility.sub.02.cumSd = apply(lossOfUtility.sub.02,2,cumSd)
lossOfUtility.sub.02.cumSd.sel4 = lossOfUtility.sub.02.cumSd[,sel4]
lossOfUtility.sub.02.sdCumMean = apply(lossOfUtility.sub.02.cumSd,2,function(x)
    {x/sqrt(nn)})
lossOfUtility.sub.02.sdCumMean.sel4 = lossOfUtility.sub.02.sdCumMean[,sel4]
lossOfUtility.sub.02.cumMean.lowerCL = lossOfUtility.sub.02.cumMean - qAlpha.
    half * lossOfUtility.sub.02.sdCumMean
lossOfUtility.sub.02.cumMean.upperCL = lossOfUtility.sub.02.cumMean + qAlpha.
    half * lossOfUtility.sub.02.sdCumMean
lossOfUtility.sub.02.cumMean.lowerCL.sel4 = lossOfUtility.sub.02.cumMean.lowerCL
    [,sel4]
lossOfUtility.sub.02.cumMean.upperCL.sel4 = lossOfUtility.sub.02.cumMean.upperCL
    [,sel4]
terminalLogReturn.sub.02 = log(terminalWealth.sub.02)
terminalLogReturn.sub.02.mean = colMeans(terminalLogReturn.sub.02)
terminalLogReturn.sub.02.mean.sel4 = terminalLogReturn.sub.02.mean[sel4]
sdLogReturn.sub.02.mean = colMeans(sdLogReturn.sub.02)
sdLogReturn.sub.02.mean.sel4 = sdLogReturn.sub.02.mean[sel4]
annualizedSdLogReturn.sub.02 = sdLogReturn.sub.02 * sqrt(nTimePoints)
annualizedSdLogReturn.sub.02.mean = colMeans(annualizedSdLogReturn.sub.02)
terminalLogReturn.sub.02.sd = colSds(terminalLogReturn.sub.02)
terminalLogReturn.sub.02.sd.sel4 = terminalLogReturn.sub.02.sd[sel4]
excessReturn.sub.02 = terminalLogReturn.sub.02 - rent
sharpeRatio.sub.02 = excessReturn.sub.02 / (sqrt(nTimePoints)*sdLogReturn.sub
    .02)
sharpeRatio.sub.02.mean = colMeans(sharpeRatio.sub.02)
sharpeRatio.sub.02.mean.sel4 = sharpeRatio.sub.02.mean[sel4]
volOfVol.sub.02 = colSds(annualizedSdLogReturn.sub.02)
correlation.sub.02 = colCorrs(terminalLogReturn.sub.02,annualizedSdLogReturn.sub
    .02)
totalTransCost.sub.02.mean = colMeans(totalTransCost.sub.02)
totalTransCost.sub.02.mean.sel4 = totalTransCost.sub.02.mean[sel4]

# Plotting
y.rangeDiff.sub.02.sel4 = colRange(lossOfUtility.sub.02.cumMean.sel4)[2,] -
    colRange(lossOfUtility.sub.02.cumMean.sel4)[1,]
y.lim.sub.02.sel4 = rbind(lossOfUtility.sub.02.mean.sel4 - y.rangeDiff.sub.02.
    sel4/25,lossOfUtility.sub.02.mean.sel4 + y.rangeDiff.sub.02.sel4/25)

transformation = 1e2
y.title = expression(paste("Mean loss of utility",phantom(0) %*% 10^2))
niceplot(lossOfUtility.sub.02.cumMean.sel4[,1]*transformation,xLabels=x.labels,
    yTitle=y.title,figsPerPage=4,y.addCustom=.2,nCol=2,horizLines=T,downsample=T
    ,ylim=y.lim.sub.02.sel4[,1]*transformation)
nicelines(lossOfUtility.sub.02.cumMean.lowerCL.sel4[,1]*transformation,
    downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.sub.02.cumMean.upperCL.sel4[,1]*transformation,
    downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(e) ",lambda*"=.02")),"Transaction costs
    strategy : Subsequent","Rebalancing strategy : Hourly"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)

niceplot(lossOfUtility.sub.02.cumMean.sel4[,2]*transformation,xLabels=x.labels,
    yTitle=y.title,figsPerPage=4,y.addCustom=.2,multiPlot=T,newDev=F,horizLines=
    T,downsample=T,ylim=y.lim.sub.02.sel4[,2]*transformation)
nicelines(lossOfUtility.sub.02.cumMean.lowerCL.sel4[,2]*transformation,
    downsample=T,col="darkgray",lty=3)
```

```
nicelines(lossOfUtility.sub.02.cumMean.upperCL.sel4[,2]*transformation, downsample=T, col="darkgray", lty=3)
legendText = c(expression(paste("(f) ",lambda*"=.02")),"Transaction costs strategy : Subsequent","Rebalancing strategy : Daily"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_02_sub_Hourly_Daily",type="eps")

niceplot(lossOfUtility.sub.02.cumMean.sel4[,3]*transformation,xLabels=x.labels, yTitle=y.title,figsPerPage=4,y.addCustom=.2,nCol=2,horizLines=T,downsample=T,ylim=y.lim.sub.02.sel4[,3]*transformation)
nicelines(lossOfUtility.sub.02.cumMean.lowerCL.sel4[,3]*transformation, downsample=T, col="darkgray", lty=3)
nicelines(lossOfUtility.sub.02.cumMean.upperCL.sel4[,3]*transformation, downsample=T, col="darkgray", lty=3)
legendText = c(expression(paste("(g) ",lambda*"=.02")),"Transaction costs strategy : Subsequent","Rebalancing strategy : Monthly"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)

niceplot(lossOfUtility.sub.02.cumMean.sel4[,4]*transformation,xLabels=x.labels, yTitle=y.title,figsPerPage=4,y.addCustom=.2,multiPlot=T,newDev=F,horizLines=T,downsample=T,ylim=y.lim.sub.02.sel4[,4]*transformation)
nicelines(lossOfUtility.sub.02.cumMean.lowerCL.sel4[,4]*transformation, downsample=T, col="darkgray", lty=3)
nicelines(lossOfUtility.sub.02.cumMean.upperCL.sel4[,4]*transformation, downsample=T, col="darkgray", lty=3)
legendText = c(expression(paste("(h) ",lambda*"=.02")),"Transaction costs strategy : Subsequent","Rebalancing strategy : Annually"))
legendObject = nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_02_sub_Monthly_Annually",type="eps")

# Setting up summarizing tables
tab1 = matrix(NA,36,5)
for (k in 1:9) {
  tab1[k*4-3,] = c(terminalWealth.th.02.mean[k],0,terminalUtility.th.02.mean[k],0,0)
  tab1[k*4-2,] = c(terminalWealth.none.02.mean[k],0,terminalUtility.none.02.mean[k],lossOfUtility.none.02.mean[k],lossOfUtility.none.02.sd[k])
  tab1[k*4-1,] = c(terminalWealth.pre.02.mean[k],totalTransCost.pre.02.mean[k],terminalUtility.pre.02.mean[k],lossOfUtility.pre.02.mean[k],lossOfUtility.pre.02.sd[k])
  tab1[k*4,]   = c(terminalWealth.sub.02.mean[k],totalTransCost.sub.02.mean[k],terminalUtility.sub.02.mean[k],lossOfUtility.sub.02.mean[k],lossOfUtility.sub.02.sd[k])
}

tab1[,2] = tab1[,2] * 1e2
tab1[,4] = tab1[,4] * 1e2
tab1[,5] = tab1[,5] * 1e3

tab1 = round(tab1,4)

for (k in 1:36) {
  tab1[k,2] = paste(tab1[k,2],"\\e{\\text-2}",sep="")
  tab1[k,4] = paste(tab1[k,4],"\\e{\\text-2}",sep="")
  tab1[k,5] = paste(tab1[k,5],"\\e{\\text-3}",sep="")
}

printex(tab1)

tab2 = matrix(NA,36,5)
for (k in 1:9) {
  tab2[k*4-3,] = c(terminalLogReturn.th.02.mean[k],annualizedSdLogReturn.th.02.mean[k],sharpeRatio.th.02.mean[k],volOfVol.th.02[k],correlation.th.02[k])
  tab2[k*4-2,] = c(terminalLogReturn.none.02.mean[k],annualizedSdLogReturn.none.02.mean[k],sharpeRatio.none.02.mean[k],volOfVol.none.02[k],correlation.
```

```
        none.02[k])
  tab2[k*4-1,] = c(terminalLogReturn.pre.02.mean[k],annualizedSdLogReturn.pre
        .02.mean[k],sharpeRatio.pre.02.mean[k],volOfVol.pre.02[k],correlation.pre
        .02[k])
  tab2[k*4,]   = c(terminalLogReturn.sub.02.mean[k],annualizedSdLogReturn.sub
        .02.mean[k],sharpeRatio.sub.02.mean[k],volOfVol.sub.02[k],correlation.sub
        .02[k])
}

tab2[,1] = tab2[,1] * 1e2
tab2[,4] = tab2[,4] * 1e3

tab2 = round(tab2,4)

for (k in 1:36) {
  tab2[k,1] = paste(tab2[k,1],"\\e{\\text-2}",sep="")
  tab2[k,4] = paste(tab2[k,4],"\\e{\\text-3}",sep="")
}

printex(tab2)

#
# Calculating relevant statistics and plotting
# Transaction cost proportion = .03
#

# Theoretical
terminalWealth.th.03 = matrix(NA,nSims,n.entries)
sdWealth.th.03 = matrix(NA,nSims,n.entries)
sdLogReturn.th.03 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.th.03[,k] = rebStrategy.transCost.03[[c(k,1)]]$thWealth.
        terminal
  sdWealth.th.03[,k] = rebStrategy.transCost.03[[c(k,1)]]$thWealth.sd
  sdLogReturn.th.03[,k] = rebStrategy.transCost.03[[c(k,1)]]$thWealth.logReturn.
        sd
}
colnames(terminalWealth.th.03) = strategyNames
terminalWealth.th.03.mean = colMeans(terminalWealth.th.03)
terminalWealth.th.03.mean.sel4 = terminalWealth.th.03.mean[sel4]
sdWealth.th.03.mean = colMeans(sdWealth.th.03)
sdWealth.th.03.mean.sel4 = sdWealth.th.03.mean[sel4]
terminalWealth.th.03.sd = colSds(terminalWealth.th.03)
terminalWealth.th.03.sd.sel4 = terminalWealth.th.03.sd[sel4]
terminalUtility.th.03 = utility(terminalWealth.th.03,riskAversion)
terminalUtility.th.03.mean = colMeans(terminalUtility.th.03)
terminalUtility.th.03.mean.sel4 = terminalUtility.th.03.mean[sel4]
terminalLogReturn.th.03 = log(terminalWealth.th.03)
terminalLogReturn.th.03.mean = colMeans(terminalLogReturn.th.03)
terminalLogReturn.th.03.mean.sel4 = terminalLogReturn.th.03.mean[sel4]
sdLogReturn.th.03.mean = colMeans(sdLogReturn.th.03)
sdLogReturn.th.03.mean.sel4 = sdLogReturn.th.03.mean[sel4]
annualizedSdLogReturn.th.03 = sdLogReturn.th.03 * sqrt(nTimePoints)
annualizedSdLogReturn.th.03.mean = colMeans(annualizedSdLogReturn.th.03)
terminalLogReturn.th.03.sd = colSds(terminalLogReturn.th.03)
terminalLogReturn.th.03.sd.sel4 = terminalLogReturn.th.03.sd[sel4]
excessReturn.th.03 = terminalLogReturn.th.03 - rent
sharpeRatio.th.03 = excessReturn.th.03 / (sqrt(nTimePoints)*sdLogReturn.th.03)
sharpeRatio.th.03.mean = colMeans(sharpeRatio.th.03)
sharpeRatio.th.03.mean.sel4 = sharpeRatio.th.03.mean[sel4]
volOfVol.th.03 = colSds(annualizedSdLogReturn.th.03)
correlation.th.03 = colCorrs(terminalLogReturn.th.03,annualizedSdLogReturn.th
      .03)

# Simulated, no transaction costs
```

```
terminalWealth.none.03 = matrix(NA,nSims,n.entries)
sdWealth.none.03 = matrix(NA,nSims,n.entries)
sdLogReturn.none.03 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.none.03[,k] = rebStrategy.transCost.03[[c(k,2)]]$simWealth.terminal
  sdWealth.none.03[,k] = rebStrategy.transCost.03[[c(k,2)]]$simWealth.sd
  sdLogReturn.none.03[,k] = rebStrategy.transCost.03[[c(k,2)]]$simWealth.logReturn.sd
}
colnames(terminalWealth.none.03) = strategyNames
terminalWealth.none.03.mean = colMeans(terminalWealth.none.03)
terminalWealth.none.03.mean.sel4 = terminalWealth.none.03.mean[sel4]
sdWealth.none.03.mean = colMeans(sdWealth.none.03)
sdWealth.none.03.mean.sel4 = sdWealth.none.03.mean[sel4]
terminalWealth.none.03.sd = colSds(terminalWealth.none.03)
terminalWealth.none.03.sd.sel4 = terminalWealth.none.03.sd[sel4]
lossOfWealth.none.03 = terminalWealth.th.03 - terminalWealth.none.03
lossOfWealth.none.03.mean = colMeans(lossOfWealth.none.03)
lossOfWealth.none.03.mean.sel4 = lossOfWealth.none.03.mean[sel4]
terminalUtility.none.03 = utility(terminalWealth.none.03,riskAversion)
terminalUtility.none.03.mean = colMeans(terminalUtility.none.03)
terminalUtility.none.03.mean.sel4 = terminalUtility.none.03.mean[sel4]
lossOfUtility.none.03 = terminalUtility.th.03 - terminalUtility.none.03
lossOfUtility.none.03.mean = colMeans(lossOfUtility.none.03)
lossOfUtility.none.03.mean.sel4 = lossOfUtility.none.03.mean[sel4]
lossOfUtility.none.03.sd = colSds(lossOfUtility.none.03)
terminalLogReturn.none.03 = log(terminalWealth.none.03)
terminalLogReturn.none.03.mean = colMeans(terminalLogReturn.none.03)
terminalLogReturn.none.03.mean.sel4 = terminalLogReturn.none.03.mean[sel4]
sdLogReturn.none.03.mean = colMeans(sdLogReturn.none.03)
sdLogReturn.none.03.mean.sel4 = sdLogReturn.none.03.mean[sel4]
annualizedSdLogReturn.none.03 = sdLogReturn.none.03 * sqrt(nTimePoints)
annualizedSdLogReturn.none.03.mean = colMeans(annualizedSdLogReturn.none.03)
terminalLogReturn.none.03.sd = colSds(terminalLogReturn.none.03)
terminalLogReturn.none.03.sd.sel4 = terminalLogReturn.none.03.sd[sel4]
excessReturn.none.03 = terminalLogReturn.none.03 - rent
sharpeRatio.none.03 = excessReturn.none.03 / (sqrt(nTimePoints)*sdLogReturn.none.03)
sharpeRatio.none.03.mean = colMeans(sharpeRatio.none.03)
sharpeRatio.none.03.mean.sel4 = sharpeRatio.none.03.mean[sel4]
volOfVol.none.03 = colSds(annualizedSdLogReturn.none.03)
correlation.none.03 = colCorrs(terminalLogReturn.none.03,annualizedSdLogReturn.none.03)

# Simulated, preceding transaction costs
terminalWealth.pre.03 = matrix(NA,nSims,n.entries)
sdWealth.pre.03 = matrix(NA,nSims,n.entries)
sdLogReturn.pre.03 = matrix(NA,nSims,n.entries)
totalTransCost.pre.03 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.pre.03[,k] = rebStrategy.transCost.03[[c(k,3)]]$simWealth.terminal
  sdWealth.pre.03[,k] = rebStrategy.transCost.03[[c(k,3)]]$simWealth.sd
  sdLogReturn.pre.03[,k] = rebStrategy.transCost.03[[c(k,3)]]$simWealth.logReturn.sd
  totalTransCost.pre.03[,k] = rebStrategy.transCost.03[[c(k,3)]]$totalTransCost
}
colnames(terminalWealth.pre.03) = strategyNames
terminalWealth.pre.03.mean = colMeans(terminalWealth.pre.03)
terminalWealth.pre.03.mean.sel4 = terminalWealth.pre.03.mean[sel4]
sdWealth.pre.03.mean = colMeans(sdWealth.pre.03)
sdWealth.pre.03.mean.sel4 = sdWealth.pre.03.mean[sel4]
terminalWealth.pre.03.sd = colSds(terminalWealth.pre.03)
terminalWealth.pre.03.sd.sel4 = terminalWealth.pre.03.sd[sel4]
```

```r
lossOfWealth.pre.03 = terminalWealth.th.03 - terminalWealth.pre.03
lossOfWealth.pre.03.mean = colMeans(lossOfWealth.pre.03)
lossOfWealth.pre.03.mean.sel4 = lossOfWealth.pre.03.mean[sel4]
terminalUtility.pre.03 = utility(terminalWealth.pre.03,riskAversion)
terminalUtility.pre.03.mean = colMeans(terminalUtility.pre.03)
terminalUtility.pre.03.mean.sel4 = terminalUtility.pre.03.mean[sel4]
lossOfUtility.pre.03 = terminalUtility.th.03 - terminalUtility.pre.03
lossOfUtility.pre.03.sel4 = lossOfUtility.pre.03[,sel4]
lossOfUtility.pre.03.mean = colMeans(lossOfUtility.pre.03)
lossOfUtility.pre.03.mean.sel4 = lossOfUtility.pre.03.mean[sel4]
lossOfUtility.pre.03.sd = colSds(lossOfUtility.pre.03)
lossOfUtility.pre.03.cumMean = apply(lossOfUtility.pre.03,2,cumMean)
lossOfUtility.pre.03.cumMean.sel4 = lossOfUtility.pre.03.cumMean[,sel4]
lossOfUtility.pre.03.cumSd = apply(lossOfUtility.pre.03,2,cumSd)
lossOfUtility.pre.03.cumSd.sel4 = lossOfUtility.pre.03.cumSd[,sel4]
lossOfUtility.pre.03.sdCumMean = apply(lossOfUtility.pre.03.cumSd,2,function(x){x/sqrt(nn)})
lossOfUtility.pre.03.sdCumMean.sel4 = lossOfUtility.pre.03.sdCumMean[,sel4]
lossOfUtility.pre.03.cumMean.lowerCL = lossOfUtility.pre.03.cumMean - qAlpha.half * lossOfUtility.pre.03.sdCumMean
lossOfUtility.pre.03.cumMean.upperCL = lossOfUtility.pre.03.cumMean + qAlpha.half * lossOfUtility.pre.03.sdCumMean
lossOfUtility.pre.03.cumMean.lowerCL.sel4 = lossOfUtility.pre.03.cumMean.lowerCL[,sel4]
lossOfUtility.pre.03.cumMean.upperCL.sel4 = lossOfUtility.pre.03.cumMean.upperCL[,sel4]
terminalLogReturn.pre.03 = log(terminalWealth.pre.03)
terminalLogReturn.pre.03.mean = colMeans(terminalLogReturn.pre.03)
terminalLogReturn.pre.03.mean.sel4 = terminalLogReturn.pre.03.mean[sel4]
sdLogReturn.pre.03.mean = colMeans(sdLogReturn.pre.03)
sdLogReturn.pre.03.mean.sel4 = sdLogReturn.pre.03.mean[sel4]
annualizedSdLogReturn.pre.03 = sdLogReturn.pre.03 * sqrt(nTimePoints)
annualizedSdLogReturn.pre.03.mean = colMeans(annualizedSdLogReturn.pre.03)
terminalLogReturn.pre.03.sd = colSds(terminalLogReturn.pre.03)
terminalLogReturn.pre.03.sd.sel4 = terminalLogReturn.pre.03.sd[sel4]
excessReturn.pre.03 = terminalLogReturn.pre.03 - rent
sharpeRatio.pre.03 = excessReturn.pre.03 / (sqrt(nTimePoints)*sdLogReturn.pre.03)
sharpeRatio.pre.03.mean = colMeans(sharpeRatio.pre.03)
sharpeRatio.pre.03.mean.sel4 = sharpeRatio.pre.03.mean[sel4]
volOfVol.pre.03 = colSds(annualizedSdLogReturn.pre.03)
correlation.pre.03 = colCorrs(terminalLogReturn.pre.03,annualizedSdLogReturn.pre.03)
totalTransCost.pre.03.mean = colMeans(totalTransCost.pre.03)
totalTransCost.pre.03.mean.sel4 = totalTransCost.pre.03.mean[sel4]

y.rangeDiff.pre.03.sel4 = colRange(lossOfUtility.pre.03.cumMean.sel4)[2,] - colRange(lossOfUtility.pre.03.cumMean.sel4)[1,]
y.lim.pre.03.sel4 = rbind(lossOfUtility.pre.03.mean.sel4 - y.rangeDiff.pre.03.sel4/25,lossOfUtility.pre.03.mean.sel4 + y.rangeDiff.pre.03.sel4/25)

transformation = 1e2
y.title = expression(paste("Mean loss of utility",phantom(0) %*% 10^2))
niceplot(lossOfUtility.pre.03.cumMean.sel4[,1]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,nCol=2,horizLines=T,downsample=T,ylim=y.lim.pre.03.sel4[,1]*transformation)
nicelines(lossOfUtility.pre.03.cumMean.lowerCL.sel4[,1]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.pre.03.cumMean.upperCL.sel4[,1]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(a) ",lambda*"=.03")),"Transaction costs strategy : Preceding","Rebalancing strategy : Hourly"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)

niceplot(lossOfUtility.pre.03.cumMean.sel4[,2]*transformation,xLabels=x.labels,
```

```
    yTitle=y.title,figsPerPage=4,y.addCustom=.2,multiPlot=T,newDev=F,horizLines=T,downsample=T,ylim=y.lim.pre.03.sel4[,2]*transformation*c(1,1.0001))
nicelines(lossOfUtility.pre.03.cumMean.lowerCL.sel4[,2]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.pre.03.cumMean.upperCL.sel4[,2]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(b) ",lambda*"=.03")),"Transaction costs strategy : Preceding","Rebalancing strategy : Daily"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_03_pre_Hourly_Daily",type="eps")

niceplot(lossOfUtility.pre.03.cumMean.sel4[,3]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,nCol=2,horizLines=T,downsample=T,ylim=y.lim.pre.03.sel4[,3]*transformation)
nicelines(lossOfUtility.pre.03.cumMean.lowerCL.sel4[,3]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.pre.03.cumMean.upperCL.sel4[,3]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(c) ",lambda*"=.03")),"Transaction costs strategy : Preceding","Rebalancing strategy : Monthly"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)

niceplot(lossOfUtility.pre.03.cumMean.sel4[,4]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,multiPlot=T,newDev=F,horizLines=T,downsample=T,ylim=y.lim.pre.03.sel4[,4]*transformation)
nicelines(lossOfUtility.pre.03.cumMean.lowerCL.sel4[,4]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.pre.03.cumMean.upperCL.sel4[,4]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(d) ",lambda*"=.03")),"Transaction costs strategy : Preceding","Rebalancing strategy : Annually"))
legendObject = nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_03_pre_Monthly_Annually",type="eps")

x.ticks = 1:9
x.title = "Rebalancing strategy"
niceplot(x.ticks,lossOfUtility.pre.03.mean*transformation,xLabels=strategyNames,xTitle=x.title,yTitle=y.title,y.addCustom=.2)
abline(v=x.ticks,lty=3)
legendText = expression(paste("(c) ",lambda*"=.03"))
nicelegend("left",legendText,horiz=T,bty="n",bg="white",cex=.7)
savePlot("images/rebStrategy_v_lossOfUtility_transCost_03",type="eps")

y.title = "Sharpe ratio"
niceplot(x.ticks,sharpeRatio.pre.03.mean,xLabels=strategyNames,xTitle=x.title,yTitle=y.title)
abline(v=x.ticks,lty=3)
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/rebStrategy_v_sharpeRatio_transCost_03",type="eps")

# Simulated, subsequent transaction costs
terminalWealth.sub.03 = matrix(NA,nSims,n.entries)
sdWealth.sub.03 = matrix(NA,nSims,n.entries)
sdLogReturn.sub.03 = matrix(NA,nSims,n.entries)
totalTransCost.sub.03 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.sub.03[,k] = rebStrategy.transCost.03[[c(k,4)]]$simWealth.terminal
  sdWealth.sub.03[,k] = rebStrategy.transCost.03[[c(k,4)]]$simWealth.sd
  sdLogReturn.sub.03[,k] = rebStrategy.transCost.03[[c(k,4)]]$simWealth.logReturn.sd
  totalTransCost.sub.03[,k] = rebStrategy.transCost.03[[c(k,4)]]$totalTransCost
}
colnames(terminalWealth.sub.03) = strategyNames
terminalWealth.sub.03.mean = colMeans(terminalWealth.sub.03)
```

```
terminalWealth.sub.03.mean.sel4 = terminalWealth.sub.03.mean[sel4]
sdWealth.sub.03.mean = colMeans(sdWealth.sub.03)
sdWealth.sub.03.mean.sel4 = sdWealth.sub.03.mean[sel4]
terminalWealth.sub.03.sd = colSds(terminalWealth.sub.03)
terminalWealth.sub.03.sd.sel4 = terminalWealth.sub.03.sd[sel4]
lossOfWealth.sub.03 = terminalWealth.th.03 - terminalWealth.sub.03
lossOfWealth.sub.03.mean = colMeans(lossOfWealth.sub.03)
lossOfWealth.sub.03.mean.sel4 = lossOfWealth.sub.03.mean[sel4]
terminalUtility.sub.03 = utility(terminalWealth.sub.03,riskAversion)
terminalUtility.sub.03.mean = colMeans(terminalUtility.sub.03)
terminalUtility.sub.03.mean.sel4 = terminalUtility.sub.03.mean[sel4]
lossOfUtility.sub.03 = terminalUtility.th.03 - terminalUtility.sub.03
lossOfUtility.sub.03.sel4 = lossOfUtility.sub.03[,sel4]
lossOfUtility.sub.03.mean = colMeans(lossOfUtility.sub.03)
lossOfUtility.sub.03.mean.sel4 = lossOfUtility.sub.03.mean[sel4]
lossOfUtility.sub.03.sd = colSds(lossOfUtility.sub.03)
lossOfUtility.sub.03.cumMean = apply(lossOfUtility.sub.03,2,cumMean)
lossOfUtility.sub.03.cumMean.sel4 = lossOfUtility.sub.03.cumMean[,sel4]
lossOfUtility.sub.03.cumSd = apply(lossOfUtility.sub.03,2,cumSd)
lossOfUtility.sub.03.cumSd.sel4 = lossOfUtility.sub.03.cumSd[,sel4]
lossOfUtility.sub.03.sdCumMean = apply(lossOfUtility.sub.03.cumSd,2,function(x)
    {x/sqrt(nn)})
lossOfUtility.sub.03.sdCumMean.sel4 = lossOfUtility.sub.03.sdCumMean[,sel4]
lossOfUtility.sub.03.cumMean.lowerCL = lossOfUtility.sub.03.cumMean - qAlpha.
    half * lossOfUtility.sub.03.sdCumMean
lossOfUtility.sub.03.cumMean.upperCL = lossOfUtility.sub.03.cumMean + qAlpha.
    half * lossOfUtility.sub.03.sdCumMean
lossOfUtility.sub.03.cumMean.lowerCL.sel4 = lossOfUtility.sub.03.cumMean.lowerCL
    [,sel4]
lossOfUtility.sub.03.cumMean.upperCL.sel4 = lossOfUtility.sub.03.cumMean.upperCL
    [,sel4]
terminalLogReturn.sub.03 = log(terminalWealth.sub.03)
terminalLogReturn.sub.03.mean = colMeans(terminalLogReturn.sub.03)
terminalLogReturn.sub.03.mean.sel4 = terminalLogReturn.sub.03.mean[sel4]
sdLogReturn.sub.03.mean = colMeans(sdLogReturn.sub.03)
sdLogReturn.sub.03.mean.sel4 = sdLogReturn.sub.03.mean[sel4]
annualizedSdLogReturn.sub.03 = sdLogReturn.sub.03 * sqrt(nTimePoints)
annualizedSdLogReturn.sub.03.mean = colMeans(annualizedSdLogReturn.sub.03)
terminalLogReturn.sub.03.sd = colSds(terminalLogReturn.sub.03)
terminalLogReturn.sub.03.sd.sel4 = terminalLogReturn.sub.03.sd[sel4]
excessReturn.sub.03 = terminalLogReturn.sub.03 - rent
sharpeRatio.sub.03 = excessReturn.sub.03 / (sqrt(nTimePoints)*sdLogReturn.sub
    .03)
sharpeRatio.sub.03.mean = colMeans(sharpeRatio.sub.03)
sharpeRatio.sub.03.mean.sel4 = sharpeRatio.sub.03.mean[sel4]
volOfVol.sub.03 = colSds(annualizedSdLogReturn.sub.03)
correlation.sub.03 = colCorrs(terminalLogReturn.sub.03,annualizedSdLogReturn.sub
    .03)
totalTransCost.sub.03.mean = colMeans(totalTransCost.sub.03)
totalTransCost.sub.03.mean.sel4 = totalTransCost.sub.03.mean[sel4]

# Plotting
y.rangeDiff.sub.03.sel4 = colRange(lossOfUtility.sub.03.cumMean.sel4)[2,] -
    colRange(lossOfUtility.sub.03.cumMean.sel4)[1,]
y.lim.sub.03.sel4 = rbind(lossOfUtility.sub.03.mean.sel4 - y.rangeDiff.sub.03.
    sel4/25,lossOfUtility.sub.03.mean.sel4 + y.rangeDiff.sub.03.sel4/25)

transformation = 1e2
y.title = expression(paste("Mean loss of utility",phantom(0) %*% 10^2))
niceplot(lossOfUtility.sub.03.cumMean.sel4[,1]*transformation,xLabels=x.labels,
    yTitle=y.title,figsPerPage=4,y.addCustom=.2,nCol=2,horizLines=T,downsample=T
    ,ylim=y.lim.sub.03.sel4[,1]*transformation)
nicelines(lossOfUtility.sub.03.cumMean.lowerCL.sel4[,1]*transformation,
    downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.sub.03.cumMean.upperCL.sel4[,1]*transformation,
```

```
    downsample=T, col="darkgray", lty=3)
legendText = c(expression(paste("(e) ",lambda*"=.03")),"Transaction costs strategy : Subsequent","Rebalancing strategy : Hourly"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)

niceplot(lossOfUtility.sub.03.cumMean.sel4[,2]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,multiPlot=T,newDev=F,horizLines=T,downsample=T,ylim=y.lim.sub.03.sel4[,2]*transformation)
nicelines(lossOfUtility.sub.03.cumMean.lowerCL.sel4[,2]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.sub.03.cumMean.upperCL.sel4[,2]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(f) ",lambda*"=.03")),"Transaction costs strategy : Subsequent","Rebalancing strategy : Daily"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_03_sub_Hourly_Daily",type="eps")

niceplot(lossOfUtility.sub.03.cumMean.sel4[,3]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,nCol=2,horizLines=T,downsample=T,ylim=y.lim.sub.03.sel4[,3]*transformation)
nicelines(lossOfUtility.sub.03.cumMean.lowerCL.sel4[,3]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.sub.03.cumMean.upperCL.sel4[,3]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(g) ",lambda*"=.03")),"Transaction costs strategy : Subsequent","Rebalancing strategy : Monthly"))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)

niceplot(lossOfUtility.sub.03.cumMean.sel4[,4]*transformation,xLabels=x.labels,yTitle=y.title,figsPerPage=4,y.addCustom=.2,multiPlot=T,newDev=F,horizLines=T,downsample=T,ylim=y.lim.sub.03.sel4[,4]*transformation)
nicelines(lossOfUtility.sub.03.cumMean.lowerCL.sel4[,4]*transformation,downsample=T,col="darkgray",lty=3)
nicelines(lossOfUtility.sub.03.cumMean.upperCL.sel4[,4]*transformation,downsample=T,col="darkgray",lty=3)
legendText = c(expression(paste("(h) ",lambda*"=.03")),"Transaction costs strategy : Subsequent","Rebalancing strategy : Annually"))
legendObject = nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_03_sub_Monthly_Annually",type="eps")

# Setting up summarizing tables
tab1 = matrix(NA,36,5)
for (k in 1:9) {
  tab1[k*4-3,] = c(terminalWealth.th.03.mean[k],0,terminalUtility.th.03.mean[k],0,0)
  tab1[k*4-2,] = c(terminalWealth.none.03.mean[k],0,terminalUtility.none.03.mean[k],lossOfUtility.none.03.mean[k],lossOfUtility.none.03.sd[k])
  tab1[k*4-1,] = c(terminalWealth.pre.03.mean[k],totalTransCost.pre.03.mean[k],terminalUtility.pre.03.mean[k],lossOfUtility.pre.03.mean[k],lossOfUtility.pre.03.sd[k])
  tab1[k*4,]   = c(terminalWealth.sub.03.mean[k],totalTransCost.sub.03.mean[k],terminalUtility.sub.03.mean[k],lossOfUtility.sub.03.mean[k],lossOfUtility.sub.03.sd[k])
}

tab1[,2] = tab1[,2] * 1e2
tab1[,4] = tab1[,4] * 1e2
tab1[,5] = tab1[,5] * 1e3

tab1 = round(tab1,4)

for (k in 1:36) {
  tab1[k,2] = paste(tab1[k,2],"\\e{\\text-2}",sep="")
  tab1[k,4] = paste(tab1[k,4],"\\e{\\text-2}",sep="")
  tab1[k,5] = paste(tab1[k,5],"\\e{\\text-3}",sep="")
```

```
}

printex(tab1)

tab2 = matrix(NA,36,5)
for (k in 1:9) {
  tab2[k*4-3,] = c(terminalLogReturn.th.03.mean[k],annualizedSdLogReturn.th.03.mean[k],sharpeRatio.th.03.mean[k],volOfVol.th.03[k],correlation.th.03[k])
  tab2[k*4-2,] = c(terminalLogReturn.none.03.mean[k],annualizedSdLogReturn.none.03.mean[k],sharpeRatio.none.03.mean[k],volOfVol.none.03[k],correlation.none.03[k])
  tab2[k*4-1,] = c(terminalLogReturn.pre.03.mean[k],annualizedSdLogReturn.pre.03.mean[k],sharpeRatio.pre.03.mean[k],volOfVol.pre.03[k],correlation.pre.03[k])
  tab2[k*4,]   = c(terminalLogReturn.sub.03.mean[k],annualizedSdLogReturn.sub.03.mean[k],sharpeRatio.sub.03.mean[k],volOfVol.sub.03[k],correlation.sub.03[k])
}

tab2[,1] = tab2[,1] * 1e2
tab2[,4] = tab2[,4] * 1e3

tab2 = round(tab2,4)

for (k in 1:36) {
  tab2[k,1] = paste(tab2[k,1],"\\e{\\text-2}",sep="")
  tab2[k,4] = paste(tab2[k,4],"\\e{\\text-3}",sep="")
}

printex(tab2)

#
# Analysis of distributions of total transaction costs, lambda = .01
#

x.title = "Total transaction cost"
y.title = "Frequency"
breaksLength = 70

# Hourly rebalancings
dataSet = totalTransCost.pre.01[,1]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(a) ",lambda*"=.01")),"T.c. strategy : Preceding ","Reb. strategy : Hourly"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Daily rebalancings
dataSet = totalTransCost.pre.01[,3]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(b) ",lambda*"=.01")),"T.c. strategy : Preceding ","Reb. strategy : Daily"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Saving dual-plot
savePlot("images/hist_transCost_HourlyDaily",type="eps")

```

```
# Every 3rd day rebalancings
dataSet = totalTransCost.pre.01[,4]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(c) ",lambda*"=.01")),"T.c. strategy : Preceding ","Reb. strategy : Ev. 3rd day"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Every 12th day rebalancings
dataSet = totalTransCost.pre.01[,5]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(d) ",lambda*"=.01")),"T.c. strategy : Preceding ","Reb. strategy : Ev. 12th day"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Saving dual-plot
savePlot("images/hist_transCost_3rd12th",type="eps")

# Hourly rebalancings
dataSet = totalTransCost.pre.01[,6]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(e) ",lambda*"=.01")),"T.c. strategy : Preceding ","Reb. strategy : Monthly"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Daily rebalancings
dataSet = totalTransCost.pre.01[,7]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(f) ",lambda*"=.01")),"T.c. strategy : Preceding ","Reb. strategy : Bimonthly"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Saving dual-plot
savePlot("images/hist_transCost_MonthlyBi",type="eps")

# Hourly rebalancings
dataSet = totalTransCost.pre.01[,8]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(g) ",lambda*"=.01")),"T.c. strategy : Preceding ","Reb. strategy : Semiannually"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Daily rebalancings
dataSet = totalTransCost.pre.01[,9]
```

```
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(h) ",lambda*"=.01")),"T.c. strategy : Preceding","Reb. strategy : Annually"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Saving dual-plot
savePlot("images/hist_transCost_SemiAnnually",type="eps")

#
# Analysis of distributions of total transaction costs, lambda = .02
#

x.title = "Total transaction cost"
y.title = "Frequency"
breaksLength = 70

# Hourly rebalancings
dataSet = totalTransCost.pre.02[,1]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(a) ",lambda*"=.02")),"T.c. strategy : Preceding","Reb. strategy : Hourly"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Daily rebalancings
dataSet = totalTransCost.pre.02[,3]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(b) ",lambda*"=.02")),"T.c. strategy : Preceding","Reb. strategy : Daily"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Saving dual-plot
savePlot("images/hist_transCost02_HourlyDaily",type="eps")

# Every 3rd day rebalancings
dataSet = totalTransCost.pre.02[,4]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(c) ",lambda*"=.02")),"T.c. strategy : Preceding","Reb. strategy : Ev. 3rd day"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Every 12th day rebalancings
dataSet = totalTransCost.pre.02[,5]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,
```

```
    breaks=res)
legendText = c(expression(paste("(d) ",lambda*"=.02"),"T.c. strategy : Preceding
    ","Reb. strategy : Ev. 12th day"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Saving dual-plot
savePlot("images/hist_transCost02_3rd12th",type="eps")

# Hourly rebalancings
dataSet = totalTransCost.pre.02[,6]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(e) ",lambda*"=.02"),"T.c. strategy : Preceding
    ","Reb. strategy : Monthly"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Daily rebalancings
dataSet = totalTransCost.pre.02[,7]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,
    breaks=res)
legendText = c(expression(paste("(f) ",lambda*"=.02"),"T.c. strategy : Preceding
    ","Reb. strategy : Bimonthly"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Saving dual-plot
savePlot("images/hist_transCost02_MonthlyBi",type="eps")

# Hourly rebalancings
dataSet = totalTransCost.pre.02[,8]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(g) ",lambda*"=.02"),"T.c. strategy : Preceding
    ","Reb. strategy : Semiannually"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Daily rebalancings
dataSet = totalTransCost.pre.02[,9]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,
    breaks=res)
legendText = c(expression(paste("(h) ",lambda*"=.02"),"T.c. strategy : Preceding
    ","Reb. strategy : Annually"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Saving dual-plot
savePlot("images/hist_transCost02_SemiAnnually",type="eps")

#
# Analysis of distributions of total transaction costs, lambda = .03
#

x.title = "Total transaction cost"
```

```
y.title = "Frequency"
breaksLength = 70

# Hourly rebalancings
dataSet = totalTransCost.pre.03[,1]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(a) ",lambda*"=.03")),"T.c. strategy : Preceding
    ","Reb. strategy : Hourly"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Daily rebalancings
dataSet = totalTransCost.pre.03[,3]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,
    breaks=res)
legendText = c(expression(paste("(b) ",lambda*"=.03")),"T.c. strategy : Preceding
    ","Reb. strategy : Daily"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Saving dual-plot
savePlot("images/hist_transCost03_HourlyDaily",type="eps")

# Every 3rd day rebalancings
dataSet = totalTransCost.pre.03[,4]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(c) ",lambda*"=.03")),"T.c. strategy : Preceding
    ","Reb. strategy : Ev. 3rd day"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Every 12th day rebalancings
dataSet = totalTransCost.pre.03[,5]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,
    breaks=res)
legendText = c(expression(paste("(d) ",lambda*"=.03")),"T.c. strategy : Preceding
    ","Reb. strategy : Ev. 12th day"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Saving dual-plot
savePlot("images/hist_transCost03_3rd12th",type="eps")

# Hourly rebalancings
dataSet = totalTransCost.pre.03[,6]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(e) ",lambda*"=.03")),"T.c. strategy : Preceding
    ","Reb. strategy : Monthly"))
nicelegend("topleft",legendText,bty="n",cex=.7)
```

```
# Daily rebalancings
dataSet = totalTransCost.pre.03[,7]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,
    breaks=res)
legendText = c(expression(paste("(f) ",lambda*"=.03")),"T.c. strategy : Preceding
    ","Reb. strategy : Bimonthly"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Saving dual-plot
savePlot("images/hist_transCost03_MonthlyBi",type="eps")

# Hourly rebalancings
dataSet = totalTransCost.pre.03[,8]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(g) ",lambda*"=.03")),"T.c. strategy : Preceding
    ","Reb. strategy : Semiannually"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Daily rebalancings
dataSet = totalTransCost.pre.03[,9]
print(range(dataSet))
res = seq(min(dataSet),max(dataSet),length=breaksLength)
histObject = hist(dataSet,breaks=res,plot=F)
y.lim = range(histObject$counts) * 1.3
nicehist(dataSet,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,ylim=y.lim,
    breaks=res)
legendText = c(expression(paste("(h) ",lambda*"=.03")),"T.c. strategy : Preceding
    ","Reb. strategy : Annually"))
nicelegend("topleft",legendText,bty="n",cex=.7)

# Saving dual-plot
savePlot("images/hist_transCost03_SemiAnnually",type="eps")
```

B.6 Simulation model IV

B.6.1 Simulation machinery

```
##
# Master Thesis
# Simulation model IV
# Simulation algorithm
#

require(mnormt)

simPortfolio.stochVol = function(nSims,paramSet,dualBrownianFileName=NULL) {
  #
  # Simulates nSims portfolios following the 14 parameter values of paramSet
  # and returns terminal utilities of theoretical and simulated wealth and
```

```
# the loss of utility. Includes transaction costs and stochastic
# volatility!
#
require(mnormt)

logReturn = function(x) {
  #
  # Computes the log returns of a time series x.
      #
  n = length(x)
      xUp = x[2:n]
      xLow = x[1:(n-1)]
      logReturns = log(xUp/xLow)
      return(logReturns)
}

riskAversion = function(drift,volatility,rent,VaR,delta,alpha) {
  #
  # Computes the risk aversion parameter of a power-type utility function
  # through Value at Risk.
  #
  qAlpha = qnorm(alpha)
  solution = 1:2*NA
  a = drift - rent + qAlpha*volatility/sqrt(delta)
  b = 2*volatility^2*(VaR/delta+rent)
  solution[1] = 1 + (drift-rent)*(a+sqrt(a^2+b))/b
  solution[2] = 1 + (drift-rent)*(a-sqrt(a^2+b))/b
  return(solution)
}

optimalControl = function(drift,volatility,rent,riskAversion) {
  #
  # Computes the optimal control following a power-type utility function.
  #
  control = pmax(pmin((drift-rent)/((1-riskAversion)*volatility^2),1),0)
  return(control)
}

dualBrownianIncrements = function(n,delta,correlation) {
  #
  # Simulates random series of n brownian increments with variance delta.
  #
  varcov = matrix(c(1,correlation,correlation,1)*delta,2,2)
  meanVector = c(0,0)
  return(rmnorm(n,meanVector,varcov))
}

#
# Assigning variables.
#
varNames = c("initWealth","nTradingDays","nDailyIncrements","nDailyRebs","drift","rent","aversion","costProp","var.init","reversionRate","var.long","volOfVol","correlation")
nParams = length(paramSet)
nParams.required = length(varNames)
if (nParams != nParams.required) stop(paste("Number of input parameters equals ",nParams,". Must equal ",nParams.required,sep=""))
for (j in 1:nParams.required) { assign(varNames[j],paramSet[j]) }

#
# Initializing the simulation structure.
#

simIndex = 1:nSims
nTimePoints = nTradingDays * nDailyIncrements
```

```
lastIndex = nTimePoints
delta = 1 / nTimePoints
timePoints = seq(delta,1,delta)
nRebDelay = nDailyIncrements / nDailyRebs
rebIndex = seq(nRebDelay,nTimePoints,nRebDelay)
days = seq(delta*nTradingDays,nTradingDays,delta*nTradingDays)
rebDays = days[rebIndex]
ones = rep(1,nRebDelay)

# Start of simulation time
timeStart = proc.time()[3][[1]]

# Initializing simulation vectors
simWealth.none = NA
simWealth.transCost = NA

#
# Using full simulation scheme if nSims = 1
#

if (nSims == 1) {

  # Intializing other statistics
  return.risky = 1:nTimePoints * NA
  return.riskfree = 1:nTimePoints * NA

  # Intializing simulated wealth without transaction costs
  simWealth = NA
  simWealth.risky = NA
  simWealth.riskfree = NA
  transQuantity = 1:nTimePoints * 0
  propInRisky = NA
  propInRiskfree = NA

  # Intializing simulated wealth with preceding transaction costs
  simWealth.tc = NA
  simWealth.tc.risky = NA
  simWealth.tc.riskfree = NA
  transQuantity.tc = 1:nTimePoints * 0
  transCost.tc = 1:nTimePoints * 0
  propInRisky.tc = NA
  propInRiskfree.tc = NA

  # Generation of Brownian motions
  if (!is.null(dualBrownianFileName) && file.exists(dualBrownianFileName,sep="")) { cat("Loading brownian increments...\n"); load(dualBrownianFileName) }
  else { dualInc = dualBrownianIncrements(nTimePoints,delta,correlation) }
  if (!is.null(dualBrownianFileName) && !file.exists(dualBrownianFileName)) { cat("Saving brownian increments...\n"); save(dualInc,file=dualBrownianFileName) }
  inc.risky = dualInc[,1]
  inc.var = dualInc[,2]
  dualBM = colCumsums(dualInc)
  BM.risky = dualBM[,1]
  BM.vol = dualBM[,2]

  #
  # First part of the simulations
  #

  # Simulation of the stochastic volatility
  stochVar = 1:nTimePoints * NA
  stochVar[1] = var.init + reversionRate*(var.long-var.init)*delta + volOfVol*sqrt(var.init)*inc.var[1]
```

```
    for (k in 2:nTimePoints) { stochVar[k] = stochVar[k-1] + reversionRate*(var.
        long-stochVar[k-1])*delta + volOfVol*sqrt(stochVar[k-1])*inc.var[k] }
    stochVol = sqrt(stochVar)

    # Calculation of optimal strategy
    u.star.init = optimalControl(drift,sqrt(var.init),rent,aversion)
    u.star = optimalControl(drift,stochVol,rent,aversion)

    # Time points to be simulated (active time points)
    activeIndices = 1:nRebDelay
    rebPoint = tail(activeIndices,1)

    # Determining active variables
    inc.risky.active = inc.risky[activeIndices]
    stochVol.active = stochVol[activeIndices]
    u.star.rebPoint = u.star[rebPoint]
    return.risky[activeIndices] = cumprod(1+drift*delta+stochVol.active*inc.
        risky.active) - 1
    return.riskfree[activeIndices] = cumprod((1+rent*delta)*ones) - 1

    # Without transaction costs
    simWealth.risky[activeIndices] = u.star.init * initWealth * cumprod(1 +
        drift*delta + stochVol.active*inc.risky.active)
    simWealth.riskfree[activeIndices] = (1-u.star.init) * initWealth * cumprod
        ((1+rent*delta)*ones)
    simWealth[activeIndices] = simWealth.risky[activeIndices] + simWealth.
        riskfree[activeIndices]
    simWealth.risky.prime = simWealth.risky[rebPoint]
    simWealth.riskfree.prime = simWealth.riskfree[rebPoint]
    transQuantity[rebPoint] = (1-u.star.rebPoint)*simWealth.risky.prime - u.star
        .rebPoint*simWealth.riskfree.prime
    simWealth.risky[rebPoint] = simWealth.risky.prime - transQuantity[rebPoint]
    simWealth.riskfree[rebPoint] = simWealth.riskfree.prime + transQuantity[
        rebPoint]
    simWealth[rebPoint] = simWealth.risky[rebPoint] + simWealth.riskfree[
        rebPoint]
    propInRisky[activeIndices] = simWealth.risky[activeIndices] / simWealth[
        activeIndices]
    propInRiskfree[activeIndices] = simWealth.riskfree[activeIndices] /
        simWealth[activeIndices]

    # With transaction costs (preceding)
    simWealth.tc.risky[activeIndices] = u.star.init * initWealth * cumprod(1 +
        drift*delta + stochVol.active*inc.risky.active)
    simWealth.tc.riskfree[activeIndices] = (1-u.star.init) * initWealth *
        cumprod((1+rent*delta)*ones)
    simWealth.tc[activeIndices] = simWealth.tc.risky[activeIndices] + simWealth.
        tc.riskfree[activeIndices]
    simWealth.tc.risky.prime = simWealth.tc.risky[rebPoint]
    simWealth.tc.riskfree.prime = simWealth.tc.riskfree[rebPoint]
    signDiffReturn = sign((1-u.star.rebPoint)*u.star.init*prod(1+drift*delta+
        stochVol.active*inc.risky.active) - u.star.rebPoint*(1-u.star.init)*prod
        ((1+rent*delta)*ones))
    transQuantity.tc[rebPoint] = ((1-u.star.rebPoint)*simWealth.tc.risky.prime -
         u.star.rebPoint*simWealth.tc.riskfree.prime) / (1 - signDiffReturn*
        costProp*u.star.rebPoint)
    transCost.tc[rebPoint] = abs(costProp*transQuantity.tc[rebPoint])
    simWealth.tc.risky[rebPoint] = simWealth.tc.risky.prime - transQuantity.tc[
        rebPoint]
    simWealth.tc.riskfree[rebPoint] = simWealth.tc.riskfree.prime +
        transQuantity.tc[rebPoint] - transCost.tc[rebPoint]
    simWealth.tc[rebPoint] = simWealth.tc.risky[rebPoint] + simWealth.tc.
        riskfree[rebPoint]
    propInRisky.tc[activeIndices] = simWealth.tc.risky[activeIndices] /
        simWealth.tc[activeIndices]
```

```
    propInRiskfree.tc[activeIndices] = simWealth.tc.riskfree[activeIndices] / simWealth.tc[activeIndices]

    # Storing last rebalancing time point rebalancing strategy
    u.star.last = u.star.rebPoint

    for (j in rebIndex[-length(rebIndex)] + 1) {
                activeIndices = j:(j+nRebDelay-1)
                rebPoint = tail(activeIndices,1)

      # Determining active variables
      inc.risky.active = inc.risky[activeIndices]
      stochVol.active = stochVol[activeIndices]
      u.star.rebPoint = u.star[rebPoint]
      return.risky[activeIndices] = cumprod(1+drift*delta+stochVol.active*inc.risky.active) - 1
      return.riskfree[activeIndices] = cumprod((1+rent*delta)*ones) - 1

      # Without transaction costs
      simWealth.risky[activeIndices] = u.star.last * simWealth[j-1] * cumprod(1 + drift*delta + stochVol.active*inc.risky.active)
      simWealth.riskfree[activeIndices] = (1-u.star.last) * simWealth[j-1] * cumprod((1+rent*delta)*ones)
      simWealth[activeIndices] = simWealth.risky[activeIndices] + simWealth.riskfree[activeIndices]
      simWealth.risky.prime = simWealth.risky[rebPoint]
      simWealth.riskfree.prime = simWealth.riskfree[rebPoint]
      transQuantity[rebPoint] = (1-u.star.rebPoint)*simWealth.risky.prime - u.star.rebPoint*simWealth.riskfree.prime
      simWealth.risky[rebPoint] = simWealth.risky.prime - transQuantity[rebPoint]
      simWealth.riskfree[rebPoint] = simWealth.riskfree.prime + transQuantity[rebPoint]
      simWealth[rebPoint] = simWealth.risky[rebPoint] + simWealth.riskfree[rebPoint]
      propInRisky[activeIndices] = simWealth.risky[activeIndices] / simWealth[activeIndices]
      propInRiskfree[activeIndices] = simWealth.riskfree[activeIndices] / simWealth[activeIndices]

      # With transaction costs (preceding)
      simWealth.tc.risky[activeIndices] = u.star.last * simWealth.tc[j-1] * cumprod(1 + drift*delta + stochVol.active*inc.risky.active)
      simWealth.tc.riskfree[activeIndices] = (1-u.star.last) * simWealth.tc[j-1] * cumprod((1+rent*delta)*ones)
      simWealth.tc[activeIndices] = simWealth.tc.risky[activeIndices] + simWealth.tc.riskfree[activeIndices]
      simWealth.tc.risky.prime = simWealth.tc.risky[rebPoint]
      simWealth.tc.riskfree.prime = simWealth.tc.riskfree[rebPoint]
      signDiffReturn = sign((1-u.star.rebPoint)*u.star.last*prod(1+drift*delta+stochVol.active*inc.risky.active) - u.star.rebPoint*(1-u.star.last)*prod((1+rent*delta)*ones))
      transQuantity.tc[rebPoint] = ((1-u.star.rebPoint)*simWealth.tc.risky.prime - u.star.rebPoint*simWealth.tc.riskfree.prime) / (1 - signDiffReturn*costProp*u.star.rebPoint)
      transCost.tc[rebPoint] = abs(costProp*transQuantity.tc[rebPoint])
      simWealth.tc.risky[rebPoint] = simWealth.tc.risky.prime - transQuantity.tc[rebPoint]
      simWealth.tc.riskfree[rebPoint] = simWealth.tc.riskfree.prime + transQuantity.tc[rebPoint] - transCost.tc[rebPoint]
      simWealth.tc[rebPoint] = simWealth.tc.risky[rebPoint] + simWealth.tc.riskfree[rebPoint]
      propInRisky.tc[activeIndices] = simWealth.tc.risky[activeIndices] / simWealth.tc[activeIndices]
```

```r
        propInRiskfree.tc[activeIndices] = simWealth.tc.riskfree[activeIndices] / simWealth.tc[activeIndices]

        # Storing last rebalancing time point rebalancing strategy
        u.star.last = u.star.rebPoint
                  }
    }

    #
    # Using compact form of simulation scheme if nSims > 1
    #

    else {
      print("nSims > 1...")

      corrInc = simIndex * NA
      stochVol.mean = simIndex * NA
      stochVol.sd = simIndex * NA
      u.star.mean = simIndex*NA

      simWealth.sd = simIndex * NA
      simWealth.terminal = simIndex * NA
      simWealth.logReturn.sd = simIndex * NA

      simWealth.tc.sd = simIndex * NA
      simWealth.tc.terminal = simIndex * NA
      simWealth.tc.logReturn.sd = simIndex * NA
      totalTransCost = simIndex * 0

      for (k in 1:nSims) {

        # Generation of Brownian motion
        if (!is.null(dualBrownianFileName) && file.exists(dualBrownianFileName,sep="")) { cat("Loading brownian increments...\n"); load(dualBrownianFileName) }
        else { dualInc = dualBrownianIncrements(nTimePoints,delta,correlation) }
        if (!is.null(dualBrownianFileName) && !file.exists(dualBrownianFileName)) { cat("Saving brownian increments...\n"); save(dualInc,file=dualBrownianFileName) }
        inc.risky = dualInc[,1]
        inc.var = dualInc[,2]
        corrInc[k] = cor(inc.risky,inc.var)

        #
        # Simulated wealths until first rebalancing time point
        #

        # Simulation of the stochastic volatility
        stochVar = 1:nTimePoints * NA
        stochVar[1] = var.init + reversionRate*(var.long-var.init)*delta + volOfVol*sqrt(var.init)*inc.var[1]
        for (i in 2:nTimePoints) { stochVar[i] = stochVar[i-1] + reversionRate*(var.long-stochVar[i-1])*delta + volOfVol*sqrt(stochVar[i-1])*inc.var[i] }
        stochVol = sqrt(stochVar)
        stochVol.mean[k] = mean(stochVol)
        stochVol.sd[k] = sd(stochVol)

        # Calculation of optimal strategy
        u.star.init = optimalControl(drift,sqrt(var.init),rent,aversion)
        u.star = optimalControl(drift,stochVol,rent,aversion)
        u.star.mean[k] = mean(u.star)

        # Time points to be simulated (active time points)
        activeIndices = 1:nRebDelay
```

```
      rebPoint = tail(activeIndices,1)

      # Determining active variables
      inc.risky.active = inc.risky[activeIndices]
      stochVol.active = stochVol[activeIndices]
      u.star.rebPoint = u.star[rebPoint]
      return.risky.rebPoint = prod(1+drift*delta+stochVol.active*inc.risky.active)
      return.riskfree.rebPoint = prod((1+rent*delta)*ones)

      # No transaction costs
      simWealth = u.star.init*initWealth*cumprod(1+drift*delta+stochVol.active*inc.risky.active) + (1-u.star.init)*initWealth*cumprod((1+rent*delta)*ones)

      # With transaction costs
      simWealth.tc = u.star.init*initWealth*cumprod(1+drift*delta+stochVol.active*inc.risky.active) + (1-u.star.init)*initWealth*cumprod((1+rent*delta)*ones)
      signDiff.rebPoint = sign((1-u.star.rebPoint)*u.star.init*return.risky.rebPoint - u.star.rebPoint*(1-u.star.init)*return.riskfree.rebPoint)
      transCost = costProp * abs(((1-u.star.rebPoint)*u.star.init*initWealth*return.risky.rebPoint - u.star.rebPoint*(1-u.star.init)*initWealth*return.riskfree.rebPoint) / (1-signDiff.rebPoint*costProp*u.star.rebPoint))
      totalTransCost[k] = totalTransCost[k] + transCost
      simWealth.tc[rebPoint] = u.star.init*initWealth*return.risky.rebPoint + (1-u.star.init)*initWealth*return.riskfree.rebPoint - transCost

      # Storing last rebalancing time point rebalancing strategy
      u.star.last = u.star.rebPoint

      #
      # The rest of the simulated wealths
      #

      for (j in rebIndex[-length(rebIndex)] + 1) {

        # Time points to be simulated (active time points)
                activeIndices = j:(j+nRebDelay-1)
                rebPoint = tail(activeIndices,1)

        # Determining active variables
        inc.risky.active = inc.risky[activeIndices]
        stochVol.active = stochVol[activeIndices]
        u.star.rebPoint = u.star[rebPoint]
        return.risky.rebPoint = prod(1+drift*delta+stochVol.active*inc.risky.active)
        return.riskfree.rebPoint = prod((1+rent*delta)*ones)

        # No transaction costs
        simWealth[activeIndices] = u.star.last*simWealth[j-1]*cumprod(1+drift*delta+stochVol.active*inc.risky.active) + (1-u.star.last)*simWealth[j-1]*cumprod((1+rent*delta)*ones)

        # With transaction costs
        simWealth.tc[activeIndices] = u.star.last*simWealth.tc[j-1]*cumprod(1+drift*delta+stochVol.active*inc.risky.active) + (1-u.star.last)*simWealth.tc[j-1]*cumprod((1+rent*delta)*ones)
        signDiff.rebPoint = sign((1-u.star.rebPoint)*u.star.last*return.risky.rebPoint - u.star.rebPoint*(1-u.star.last)*return.riskfree.rebPoint)
        transCost = costProp * abs(((1-u.star.rebPoint)*u.star.last*simWealth.tc[j-1]*return.risky.rebPoint - u.star.rebPoint*(1-u.star.last)*simWealth.tc[j-1]*return.riskfree.rebPoint) / (1-signDiff.rebPoint*costProp*u.star.rebPoint))
```

```
        totalTransCost[k] = totalTransCost[k] + transCost
        simWealth.tc[rebPoint] = u.star.last*simWealth.tc[j-1]*return.risky.rebPoint + (1-u.star.last)*simWealth.tc[j-1]*return.riskfree.rebPoint - transCost

        # Storing last rebalancing time point rebalancing strategy
        u.star.last = u.star.rebPoint
              }

      simWealth.sd[k] = sd(simWealth)
      simWealth.terminal[k] = simWealth[lastIndex]
      simWealth.logReturn = logReturn(c(initWealth,simWealth))
      simWealth.logReturn.sd[k] = sd(simWealth.logReturn)

      simWealth.tc.sd[k] = sd(simWealth.tc)
      simWealth.tc.terminal[k] = simWealth.tc[lastIndex]
      simWealth.tc.logReturn = logReturn(c(initWealth,simWealth.tc))
      simWealth.tc.logReturn.sd[k] = sd(simWealth.tc.logReturn)
    }
  }

  # Calculation of total simulation time
  timeElapsed = proc.time()[3][[1]] - timeStart
  cat(nSims," simulation(s) completed in",timeElapsed," seconds.\n")
  flush.console()

  # Construction of the list of data to be returned from the function.
  if (nSims == 1) {
    stdNames = c("simWealth.risky","simWealth.riskfree","simWealth","transQuantity","transCost","propInRisky","propInRiskfree")
    returnList.none = list(simWealth.risky,simWealth.riskfree,simWealth,transQuantity,propInRisky,propInRiskfree)
    names(returnList.none) = stdNames[-5]
    returnList.tc = list(simWealth.tc.risky,simWealth.tc.riskfree,simWealth.tc,transQuantity.tc,transCost.tc,propInRisky.tc,propInRiskfree.tc)
    names(returnList.tc) = stdNames
    returnList = list(days,rebDays,rebIndex,inc.risky,inc.var,BM.risky,BM.vol,stochVol,u.star,return.risky,return.riskfree,returnList.none,returnList.tc)
    names(returnList) = c("days","rebDays","rebIndex","increments.risky","increments.vol","BM.risky","BM.vol","volatility","u.star","return.risky","return.riskfree","noTransCost","transCost")
  }
  else {
    paramSet = c(initWealth,nTradingDays,nDailyIncrements,nDailyRebs,drift,rent,aversion,costProp,var.init,reversionRate,var.long,volOfVol,correlation)
    stdNames = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")
    returnList.none = list(simWealth.terminal,simWealth.sd,simWealth.logReturn.sd)
    names(returnList.none) = stdNames[-4]
    returnList.tc = list(simWealth.tc.terminal,simWealth.tc.sd,simWealth.tc.logReturn.sd,totalTransCost)
    names(returnList.tc) = stdNames
    returnList = list(paramSet,corrInc,stochVol.mean,stochVol.sd,u.star.mean,returnList.none,returnList.tc)
    names(returnList) = c("parameters","correlation","stochVol.mean","stochVol.sd","u.star.mean","noTransCost","transCost")
  }

  return(returnList)
}
```

B.6.2 Execution

```
##
# Master thesis
# Simulation using stochastic volatility
#

setwd("M:/pc/dokumenter/Master")
if(getwd()=="M:/pc/dokumenter/Master") Sys.setenv(TMP="E:/work/joachiah")

require(doSMP)
source("R/supportFunctions.R")
source("R/machinery_general.R")
source("R/initParameters.R")
source("R/machinery_basic.R")
source("R/machinery_transCost.R")
source("R/machinery_stochVol.R")

alpha = .05
qAlpha.half = qnorm(1-alpha/2)

#
# One test run
#

nSims = 1
paramSet.stochVol[4] = 24
simObject.stochVol = simPortfolio.stochVol(nSims,paramSet.stochVol,"constVsStoch.RData")
paramSet.constVol[4] = 24
simObject.constVol = simPortfolio.transCost(nSims,paramSet.constVol,"constVsStoch.RData")

days = simObject.stochVol$days
stochVol = simObject.stochVol$volatility
x.ticks = seq(0,252,21)
x.title = "Trading days"
y.title = "Volatility"
niceplot(days,stochVol,x.ticks,xTitle=x.title,yTitle=y.title)
abline(h=sqrt(var.long),lty=3)
legendText = "(a)"
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/stochVol",type="eps")

uStar.stoch = simObject.stochVol$u.star
y.title = "u*"
niceplot(days,uStar.stoch,x.ticks,xTitle=x.title,yTitle=y.title)
abline(h=uStar.constVol,lty=3)
legendText = "(b)"
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/uStar_stoch",type="eps")

breaksLength = 70
scalar = 1e4
transCost.diff = simObject.constVol$precedingTransCost$transCost - simObject.stochVol$transCost$transCost
res = seq(min(scalar*transCost.diff),max(scalar*transCost.diff),length=breaksLength)
x.title = expression(paste("Transaction cost difference",phantom(0) %*% 10^4))
y.title = "Frequency"
nicehist(scalar*transCost.diff,xTitle=x.title,yTitle=y.title,breaks=res)
savePlot("images/transCost_diff",type="eps")
print(sum(simObject.constVol$precedingTransCost$transCost))
print(sum(simObject.stochVol$transCost$transCost))
```

```

nSims = 1
paramSet.stochVol[4] = 21/252
paramSet.constVol[4] = 21/252
rebIndex = seq(288,6048,288)
transCost.diff = 1:6000 * NA

for (i in 1:500) {
  simObject.stochVol = simPortfolio.stochVol(nSims,paramSet.stochVol)
  simObject.constVol = simPortfolio.transCost(nSims,paramSet.constVol)
  transCost.diff[(i*12-11):(i*12)] = simObject.constVol$precedingTransCost$transCost[rebIndex] - simObject.stochVol$transCost$transCost[rebIndex]
}

breaksLength = 70
scalar = 1e3
res = seq(min(scalar*transCost.diff),max(scalar*transCost.diff),length=breaksLength)
x.title = expression(paste("Transaction cost difference",phantom(0) %*% 10^3))
y.title = "Frequency"
nicehist(scalar*transCost.diff, xTitle=x.title, yTitle=y.title, breaks=res)
savePlot("images/transCost_diff_monthly",type="eps")
print(sum(simObject.constVol$precedingTransCost$transCost))
print(sum(simObject.stochVol$transCost$transCost))

#
# Multiple runs
#

# Performing reference simulations for comparison
nSims = 50000
nCores = 25
nDailyRebs = 24
nDailyRebs = c(24,6,1,1/2,1/12,1/21,1/42,1/126,1/252)
strategyNames = c("Hourly","Every 4th hour","Daily","Every 3rd day","Every 12th day","Monthly","Bimonthly","Semiannually","Annually")
volatility.const = sqrt(var.long)
u.star.const = optimalControl(drift, volatility.const, rent, riskAversion)
paramSets.basic = cbind(initWealth, nTradingDays, nDailyIncrements, nDailyRebs, drift, volatility.const, rent, riskAversion, u.star.const)
rebStrategy.benchmark.none = distribute(nSims, nCores, simPortfolio, paramSets.basic)
names(rebStrategy.benchmark.none) = strategyNames

paramSets.transCost.tc01 = cbind(initWealth, nTradingDays, nDailyIncrements, nDailyRebs, drift, volatility.const, rent, riskAversion, u.star.const, costProp=.01)
rebStrategy.benchmark.tc01 = distribute(nSims, nCores, simPortfolio.transCost, paramSets.transCost.tc01)
names(rebStrategy.benchmark.tc01) = strategyNames
n.entries = length(rebStrategy.benchmark.tc01)

for (k in 1:n.entries) {

  th = rebStrategy.benchmark.tc01[[k]]$theoretical
  rebStrategy.benchmark.tc01[[k]]$theoretical = list(merge.list(th[seq(1,3*nCores-2,3)]), merge.list(th[seq(2,3*nCores-1,3)]), merge.list(th[seq(3,3*nCores,3)]))
  names(rebStrategy.benchmark.tc01[[k]]$theoretical) = c("thWealth.terminal","thWealth.sd","thWealth.logReturn.sd")

  no = rebStrategy.benchmark.tc01[[k]]$noTransCost
  rebStrategy.benchmark.tc01[[k]]$noTransCost = list(merge.list(no[seq(1,3*nCores-2,3)]), merge.list(no[seq(2,3*nCores-1,3)]), merge.list(no[seq(3,3*
```

```
    nCores,3)]))
  names(rebStrategy.benchmark.tc01[[k]]$noTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd")

  pre = rebStrategy.benchmark.tc01[[k]]$precedingTransCost
  rebStrategy.benchmark.tc01[[k]]$precedingTransCost = list(merge.list(pre[seq(1,4*nCores-3,4)]), merge.list(pre[seq(2,4*nCores-2,4)]), merge.list(pre[seq(3,4*nCores-1,4)]), merge.list(pre[seq(4,4*nCores,4)]))
  names(rebStrategy.benchmark.tc01[[k]]$precedingTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")

  sub = rebStrategy.benchmark.tc01[[k]]$subsequentTransCost
  rebStrategy.benchmark.tc01[[k]]$subsequentTransCost = list(merge.list(sub[seq(1,4*nCores-3,4)]), merge.list(sub[seq(2,4*nCores-2,4)]), merge.list(sub[seq(3,4*nCores-1,4)]), merge.list(sub[seq(4,4*nCores,4)]))
  names(rebStrategy.benchmark.tc01[[k]]$subsequentTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")
}
save(rebStrategy.benchmark.tc01, file="Datasett/rebStrategy_stochVol_tc01_bench.RData")

paramSets.transCost.tc02 = cbind(initWealth, nTradingDays, nDailyIncrements, nDailyRebs, drift, volatility.const, rent, riskAversion, u.star.const, costProp=.02)
rebStrategy.benchmark.tc02 = distribute(nSims, nCores, simPortfolio.transCost, paramSets.transCost.tc02)
names(rebStrategy.benchmark.tc02) = strategyNames

for (k in 1:n.entries) {

  th = rebStrategy.benchmark.tc02[[k]]$theoretical
  rebStrategy.benchmark.tc02[[k]]$theoretical = list(merge.list(th[seq(1,3*nCores-2,3)]), merge.list(th[seq(2,3*nCores-1,3)]), merge.list(th[seq(3,3*nCores,3)]))
  names(rebStrategy.benchmark.tc02[[k]]$theoretical) = c("thWealth.terminal","thWealth.sd","thWealth.logReturn.sd")

  no = rebStrategy.benchmark.tc02[[k]]$noTransCost
  rebStrategy.benchmark.tc02[[k]]$noTransCost = list(merge.list(no[seq(1,3*nCores-2,3)]), merge.list(no[seq(2,3*nCores-1,3)]), merge.list(no[seq(3,3*nCores,3)]))
  names(rebStrategy.benchmark.tc02[[k]]$noTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd")

  pre = rebStrategy.benchmark.tc02[[k]]$precedingTransCost
  rebStrategy.benchmark.tc02[[k]]$precedingTransCost = list(merge.list(pre[seq(1,4*nCores-3,4)]), merge.list(pre[seq(2,4*nCores-2,4)]), merge.list(pre[seq(3,4*nCores-1,4)]), merge.list(pre[seq(4,4*nCores,4)]))
  names(rebStrategy.benchmark.tc02[[k]]$precedingTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")

  sub = rebStrategy.benchmark.tc02[[k]]$subsequentTransCost
  rebStrategy.benchmark.tc02[[k]]$subsequentTransCost = list(merge.list(sub[seq(1,4*nCores-3,4)]), merge.list(sub[seq(2,4*nCores-2,4)]), merge.list(sub[seq(3,4*nCores-1,4)]), merge.list(sub[seq(4,4*nCores,4)]))
  names(rebStrategy.benchmark.tc02[[k]]$subsequentTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")
}
save(rebStrategy.benchmark.tc02, file="Datasett/rebStrategy_stochVol_tc02_bench.RData")

paramSets.transCost.tc03 = cbind(initWealth, nTradingDays, nDailyIncrements, nDailyRebs, drift, volatility.const, rent, riskAversion, u.star.const, costProp=.03)
```

```
rebStrategy.benchmark.tc03 = distribute(nSims,nCores,simPortfolio.transCost,paramSets.transCost.tc03)
names(rebStrategy.benchmark.tc03) = strategyNames

for (k in 1:n.entries) {

  th = rebStrategy.benchmark.tc03[[k]]$theoretical
  rebStrategy.benchmark.tc03[[k]]$theoretical = list(merge.list(th[seq(1,3*nCores-2,3)]), merge.list(th[seq(2,3*nCores-1,3)]), merge.list(th[seq(3,3*nCores,3)]))
  names(rebStrategy.benchmark.tc03[[k]]$theoretical) = c("thWealth.terminal","thWealth.sd","thWealth.logReturn.sd")

  no = rebStrategy.benchmark.tc03[[k]]$noTransCost
  rebStrategy.benchmark.tc03[[k]]$noTransCost = list(merge.list(no[seq(1,3*nCores-2,3)]), merge.list(no[seq(2,3*nCores-1,3)]), merge.list(no[seq(3,3*nCores,3)]))
  names(rebStrategy.benchmark.tc03[[k]]$noTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd")

  pre = rebStrategy.benchmark.tc03[[k]]$precedingTransCost
  rebStrategy.benchmark.tc03[[k]]$precedingTransCost = list(merge.list(pre[seq(1,4*nCores-3,4)]), merge.list(pre[seq(2,4*nCores-2,4)]), merge.list(pre[seq(3,4*nCores-1,4)]), merge.list(pre[seq(4,4*nCores,4)]))
  names(rebStrategy.benchmark.tc03[[k]]$precedingTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")

  sub = rebStrategy.benchmark.tc03[[k]]$subsequentTransCost
  rebStrategy.benchmark.tc03[[k]]$subsequentTransCost = list(merge.list(sub[seq(1,4*nCores-3,4)]), merge.list(sub[seq(2,4*nCores-2,4)]), merge.list(sub[seq(3,4*nCores-1,4)]), merge.list(sub[seq(4,4*nCores,4)]))
  names(rebStrategy.benchmark.tc03[[k]]$subsequentTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")
}
save(rebStrategy.benchmark.tc03,file="Datasett/rebStrategy_stochVol_tc03_bench.RData")

#
# Performing simulations, transaction cost proportion = .01
#

nDailyRebs = c(24,6,1,1/2,1/12,1/21,1/42,1/126,1/252)
strategyNames = c("Hourly","Every 4th hour","Daily","Every 3rd day","Every 12th day","Monthly","Bimonthly","Semiannually","Annually")

costProp = .01
paramSets.stochVol = cbind(initWealth,nTradingDays,nDailyIncrements,nDailyRebs,drift,rent,riskAversion,costProp,var.init,reversionRate,var.long,volOfVol,correlation)
rebStrategy.stochVol.tc01 = distribute(nSims,nCores,simPortfolio.stochVol,paramSets.stochVol)
names(rebStrategy.stochVol.tc01) = strategyNames

 Organizing returned data
n.entries = length(rebStrategy.stochVol.tc01)
for (k in 1:n.entries) {
  rebStrategy.stochVol.tc01[[k]]$parameters = rebStrategy.stochVol.tc01[[k]]$parameters[1:nParam.stochVol]
  names(rebStrategy.stochVol.tc01[[k]]$parameters) = c("initWealth","nTradingDays","nDailyIncrements","nDailyRebs","drift","rent","riskAversion","costProp","var.init","reversionRate","var.long","volOfVol","correlation")

  none = rebStrategy.stochVol.tc01[[k]]$noTransCost
```

```
  rebStrategy.stochVol.tc01[[k]]$noTransCost = list(merge.list(none[seq(1,3*nCores-2,3)]), merge.list(none[seq(2,3*nCores-1,3)]), merge.list(none[seq(3,3*nCores,3)]))
  names(rebStrategy.stochVol.tc01[[k]]$noTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd")

  tc = rebStrategy.stochVol.tc01[[k]]$transCost
  rebStrategy.stochVol.tc01[[k]]$transCost = list(merge.list(tc[seq(1,4*nCores-3,4)]), merge.list(tc[seq(2,4*nCores-2,4)]), merge.list(tc[seq(3,4*nCores-1,4)]), merge.list(tc[seq(4,4*nCores,4)]))
  names(rebStrategy.stochVol.tc01[[k]]$transCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")
}
save(rebStrategy.stochVol.tc01, file="Datasett/rebStrategy_stochVol_tc01.RData")

#
# Performing simulations, transaction cost proportion = .02
#

costProp = .02
paramSets.stochVol = cbind(initWealth,nTradingDays,nDailyIncrements,nDailyRebs,drift,rent,riskAversion,costProp,var.init,reversionRate,var.long,volOfVol,correlation)
rebStrategy.stochVol.tc02 = distribute(nSims,nCores,simPortfolio.stochVol,paramSets.stochVol)
names(rebStrategy.stochVol.tc02) = strategyNames

 Organizing returned data
for (k in 1:n.entries) {
  rebStrategy.stochVol.tc02[[k]]$parameters = rebStrategy.stochVol.tc02[[k]]$parameters[1:nParam.stochVol]
  names(rebStrategy.stochVol.tc02[[k]]$parameters) = c("initWealth","nTradingDays","nDailyIncrements","nDailyRebs","drift","rent","riskAversion","costProp","var.init","reversionRate","var.long","volOfVol","correlation")

  none = rebStrategy.stochVol.tc02[[k]]$noTransCost
  rebStrategy.stochVol.tc02[[k]]$noTransCost = list(merge.list(none[seq(1,3*nCores-2,3)]), merge.list(none[seq(2,3*nCores-1,3)]), merge.list(none[seq(3,3*nCores,3)]))
  names(rebStrategy.stochVol.tc02[[k]]$noTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd")

  tc = rebStrategy.stochVol.tc02[[k]]$transCost
  rebStrategy.stochVol.tc02[[k]]$transCost = list(merge.list(tc[seq(1,4*nCores-3,4)]), merge.list(tc[seq(2,4*nCores-2,4)]), merge.list(tc[seq(3,4*nCores-1,4)]), merge.list(tc[seq(4,4*nCores,4)]))
  names(rebStrategy.stochVol.tc02[[k]]$transCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")
}
save(rebStrategy.stochVol.tc02, file="Datasett/rebStrategy_stochVol_tc02.RData")


 Performing simulations, transaction cost proportion = .03


costProp = .03
paramSets.stochVol = cbind(initWealth,nTradingDays,nDailyIncrements,nDailyRebs,drift,rent,riskAversion,costProp,var.init,reversionRate,var.long,volOfVol,correlation)
rebStrategy.stochVol.tc03 = distribute(nSims,nCores,simPortfolio.stochVol,paramSets.stochVol)
names(rebStrategy.stochVol.tc03) = strategyNames

 Organizing returned data
```

```r
for (k in 1:n.entries) {
  rebStrategy.stochVol.tc03[[k]]$parameters = rebStrategy.stochVol.tc03[[k]]$parameters[1:nParam.stochVol]
  names(rebStrategy.stochVol.tc03[[k]]$parameters) = c("initWealth","nTradingDays","nDailyIncrements","nDailyRebs","drift","rent","riskAversion","costProp","var.init","reversionRate","var.long","volOfVol","correlation")

  none = rebStrategy.stochVol.tc03[[k]]$noTransCost
  rebStrategy.stochVol.tc03[[k]]$noTransCost = list(merge.list(none[seq(1,3*nCores-2,3)]), merge.list(none[seq(2,3*nCores-1,3)]), merge.list(none[seq(3,3*nCores,3)]))
  names(rebStrategy.stochVol.tc03[[k]]$noTransCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd")

  tc = rebStrategy.stochVol.tc03[[k]]$transCost
  rebStrategy.stochVol.tc03[[k]]$transCost = list(merge.list(tc[seq(1,4*nCores-3,4)]), merge.list(tc[seq(2,4*nCores-2,4)]), merge.list(tc[seq(3,4*nCores-1,4)]), merge.list(tc[seq(4,4*nCores,4)]))
  names(rebStrategy.stochVol.tc03[[k]]$transCost) = c("simWealth.terminal","simWealth.sd","simWealth.logReturn.sd","totalTransCost")
}
save(rebStrategy.stochVol.tc03, file="Datasett/rebStrategy_stochVol_tc03.RData")

#
# Calculating relevant statistics and plotting
# Transaction cost proportion = 0
#

cat("\nTransaction cost proportion = 0\n\n")

x.labels = c(0,".5k","1.0k","1.5k","2.0k","2.5k","3.0k","3.5k","4.0k","4.5k","5.0k")
nn = 1:nSims

terminalWealth.tc01.none.bench = matrix(NA,nSims,n.entries)
sdWealth.tc01.none.bench = matrix(NA,nSims,n.entries)
sdLogReturn.tc01.none.bench = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.tc01.none.bench[,k] = rebStrategy.benchmark.tc01[[c(k,2)]]$simWealth.terminal
  sdWealth.tc01.none.bench[,k] = rebStrategy.benchmark.tc01[[c(k,2)]]$simWealth.sd
  sdLogReturn.tc01.none.bench[,k] = rebStrategy.benchmark.tc01[[c(k,2)]]$simWealth.logReturn.sd
}
terminalWealth.tc02.none.bench = matrix(NA,nSims,n.entries)
sdWealth.tc02.none.bench = matrix(NA,nSims,n.entries)
sdLogReturn.tc02.none.bench = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.tc02.none.bench[,k] = rebStrategy.benchmark.tc02[[c(k,2)]]$simWealth.terminal
  sdWealth.tc02.none.bench[,k] = rebStrategy.benchmark.tc02[[c(k,2)]]$simWealth.sd
  sdLogReturn.tc02.none.bench[,k] = rebStrategy.benchmark.tc02[[c(k,2)]]$simWealth.logReturn.sd
}
terminalWealth.tc03.none.bench = matrix(NA,nSims,n.entries)
sdWealth.tc03.none.bench = matrix(NA,nSims,n.entries)
sdLogReturn.tc03.none.bench = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.tc03.none.bench[,k] = rebStrategy.benchmark.tc03[[c(k,2)]]$simWealth.terminal
  sdWealth.tc03.none.bench[,k] = rebStrategy.benchmark.tc03[[c(k,2)]]$simWealth.sd
```

```
  sdLogReturn.tc03.none.bench[,k] = rebStrategy.benchmark.tc03[[c(k,2)]]$simWealth.logReturn.sd
}

terminalWealth.none.bench = rbind(terminalWealth.tc01.none.bench,terminalWealth.tc02.none.bench,terminalWealth.tc03.none.bench)
terminalWealth.none.mean.bench = colMeans(terminalWealth.none.bench)
terminalUtility.none.bench = utility(terminalWealth.none.bench,riskAversion)
terminalUtility.none.mean.bench = colMeans(terminalUtility.none.bench)
terminalUtility.none.sd.bench = colSds(terminalUtility.none.bench)

logReturn.none.bench = log(terminalWealth.none.bench)
logReturn.none.mean.bench = colMeans(logReturn.none.bench)
sdLogReturn.none.bench = rbind(sdLogReturn.tc01.none.bench,sdLogReturn.tc02.none.bench,sdLogReturn.tc03.none.bench)
volatility.none.bench = sdLogReturn.none.bench * sqrt(nTimePoints)
volatility.none.mean.bench = colMeans(volatility.none.bench)
excessReturn.none.bench = logReturn.none.bench - rent
sharpeRatio.none.bench = excessReturn.none.bench / volatility.none.bench
sharpeRatio.none.mean.bench = colMeans(sharpeRatio.none.bench)
volOfVol.none.bench = colSds(volatility.none.bench)
correlation.none.bench = colCorrs(logReturn.none.bench, volatility.none.bench)

terminalWealth.tc01.none = matrix(NA,nSims,n.entries)
sdWealth.tc01.none = matrix(NA,nSims,n.entries)
sdLogReturn.tc01.none = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.tc01.none[,k] = rebStrategy.stochVol.tc01[[c(k,6)]]$simWealth.terminal
  sdWealth.tc01.none[,k] = rebStrategy.stochVol.tc01[[c(k,6)]]$simWealth.sd
  sdLogReturn.tc01.none[,k] = rebStrategy.stochVol.tc01[[c(k,6)]]$simWealth.logReturn.sd
}
terminalWealth.tc02.none = matrix(NA,nSims,n.entries)
sdWealth.tc02.none = matrix(NA,nSims,n.entries)
sdLogReturn.tc02.none = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.tc02.none[,k] = rebStrategy.stochVol.tc02[[c(k,6)]]$simWealth.terminal
  sdWealth.tc02.none[,k] = rebStrategy.stochVol.tc02[[c(k,6)]]$simWealth.sd
  sdLogReturn.tc02.none[,k] = rebStrategy.stochVol.tc02[[c(k,6)]]$simWealth.logReturn.sd
}
terminalWealth.tc03.none = matrix(NA,nSims,n.entries)
sdWealth.tc03.none = matrix(NA,nSims,n.entries)
sdLogReturn.tc03.none = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.tc03.none[,k] = rebStrategy.stochVol.tc03[[c(k,6)]]$simWealth.terminal
  sdWealth.tc03.none[,k] = rebStrategy.stochVol.tc03[[c(k,6)]]$simWealth.sd
  sdLogReturn.tc03.none[,k] = rebStrategy.stochVol.tc03[[c(k,6)]]$simWealth.logReturn.sd
}

terminalWealth.none = rbind(terminalWealth.tc01.none,terminalWealth.tc02.none,terminalWealth.tc03.none)
nSims = nrow(terminalWealth.none)
terminalWealth.none.mean = colMeans(terminalWealth.none)
terminalUtility.none = utility(terminalWealth.none,riskAversion)
terminalUtility.none.mean = colMeans(terminalUtility.none)
terminalUtility.none.sd = colSds(terminalUtility.none)
lossOfUtility.none.mean = terminalUtility.none.mean.bench - terminalUtility.none.mean
lossOfUtility.none.mean.CL.lower = lossOfUtility.none.mean - qAlpha.half * (1/sqrt(nSims)) * sqrt(terminalUtility.none.sd.bench^2 + terminalUtility.none.
```

```
sd^2)
lossOfUtility.none.mean.CL.upper = lossOfUtility.none.mean + qAlpha.half * (1/sqrt(nSims)) * sqrt(terminalUtility.none.sd.bench^2 + terminalUtility.none.sd^2)

logReturn.none = log(terminalWealth.none)
logReturn.none.mean = colMeans(logReturn.none)
sdLogReturn.none = rbind(sdLogReturn.tc01.none,sdLogReturn.tc02.none,sdLogReturn.tc03.none)
volatility.none = sdLogReturn.none * sqrt(nTimePoints)
volatility.none.mean = colMeans(volatility.none)
excessReturn.none = logReturn.none - rent
sharpeRatio.none = excessReturn.none / volatility.none
sharpeRatio.none.mean = colMeans(sharpeRatio.none)
sharpeRatio.none.sd = colSds(sharpeRatio.none)
sharpeRatio.none.mean.CL.lower = sharpeRatio.none.mean - qAlpha.half * sharpeRatio.none.sd / sqrt(nSims)
sharpeRatio.none.mean.CL.upper = sharpeRatio.none.mean + qAlpha.half * sharpeRatio.none.sd / sqrt(nSims)
volOfVol.none = colSds(volatility.none)
correlation.none = colCorrs(logReturn.none,volatility.none)

tab1.none = matrix(NA,18,4)

for (i in 1:9) {
  tab1.none[2*i-1,] = c(terminalWealth.none.mean.bench[i],0,terminalUtility.none.mean.bench[i],0)
  tab1.none[2*i,]   = c(terminalWealth.none.mean[i],0,terminalUtility.none.mean[i],lossOfUtility.none.mean[i])
}
tab1.none[,4] = tab1.none[,4] * 1e4
tab1.none = round(tab1.none,4)

for (i in 1:18) { tab1.none[i,4] = paste(tab1.none[i,4],"\\e{\\text-4}",sep="") }
tab1.none[,2] = "-"
for (i in seq(1,17,2)) { tab1.none[i,4] = "-" }

printex(tab1.none)

tab2.none = matrix(NA,18,5)

for (i in 1:9) {
  tab2.none[2*i-1,] = c(logReturn.none.mean.bench[i],volatility.none.mean.bench[i],sharpeRatio.none.mean.bench[i],volOfVol.none.bench[i],correlation.none.bench[i])
  tab2.none[2*i,] = c(logReturn.none.mean[i],volatility.none.mean[i],sharpeRatio.none.mean[i],volOfVol.none[i],correlation.none[i])
}
tab2.none[,1] = tab2.none[,1] * 1e2
tab2.none[,3] = tab2.none[,3] * 1e2
tab2.none[,4] = tab2.none[,4] * 1e3
tab2.none = round(tab2.none,4)

for (i in 1:18) {
  tab2.none[i,1] = paste(tab2.none[i,1],"\\e{\\text-2}",sep="")
  tab2.none[i,3] = paste(tab2.none[i,3],"\\e{\\text-2}",sep="")
  tab2.none[i,4] = paste(tab2.none[i,4],"\\e{\\text-3}",sep="")
}

printex(tab2.none)

scalar = 1e4
x.labels = strategyNames
x.title = "Rebalancing strategy"
```

```
y.title = expression(paste("Mean loss of utility",phantom(0) %*% 10^4))
x.ticks = 1:9
y.range = range(c(lossOfUtility.none.mean.CL.lower*scalar,lossOfUtility.none.
    mean.CL.upper*scalar))
niceplot(lossOfUtility.none.mean*scalar,xLabels=x.labels,xTitle=x.title,yTitle=y
    .title,y.addCustom=.2,figsPerPage=4,ylim=y.range)
abline(h=0,col="darkgray",lty=3)
abline(v=x.ticks,col="darkgray",lty=3)
nicelines(lossOfUtility.none.mean.CL.lower*scalar,lty=2)
nicelines(lossOfUtility.none.mean.CL.upper*scalar,lty=2)
legendText = "(a) No transaction costs"
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_stochVol_none",type="eps")

y.title = "Mean Sharpe ratio"
y.range = range(c(sharpeRatio.none.mean.CL.lower,sharpeRatio.none.mean.CL.upper)
    )
niceplot(sharpeRatio.none.mean,xLabels=x.labels,xTitle=x.title,yTitle=y.title,
    figsPerPage=4,ylim=y.range)
abline(v=x.ticks,col="darkgray",lty=3)
nicelines(sharpeRatio.none.mean.CL.lower,lty=2)
nicelines(sharpeRatio.none.mean.CL.upper,lty=2)
legendText = "(a) No transaction costs"
nicelegend("left",legendText,bty="n",bg="white",cex=.7)
savePlot("images/sharpeRatio_stochVol_none",type="eps")

#
# Calculating relevant statistics and plotting
# Transaction cost proportion = .01
#

nSims = 50000

cat("Transaction cost proportion = .01\n")

terminalWealth.tc01.bench = matrix(NA,nSims,n.entries)
sdWealth.tc01.bench = matrix(NA,nSims,n.entries)
sdLogReturn.tc01.bench = matrix(NA,nSims,n.entries)
transCost.tc01.bench = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.tc01.bench[,k] = rebStrategy.benchmark.tc01[[c(k,3)]]$simWealth
      .terminal
  sdWealth.tc01.bench[,k] = rebStrategy.benchmark.tc01[[c(k,3)]]$simWealth.sd
  sdLogReturn.tc01.bench[,k] = rebStrategy.benchmark.tc01[[c(k,3)]]$simWealth.
      logReturn.sd
  transCost.tc01.bench[,k] = rebStrategy.benchmark.tc01[[c(k,3)]]$totalTransCost
}

terminalWealth.tc01.mean.bench = colMeans(terminalWealth.tc01.bench)
transCost.tc01.mean.bench = colMeans(transCost.tc01.bench)
terminalUtility.tc01.bench = utility(terminalWealth.tc01.bench,riskAversion)
terminalUtility.tc01.mean.bench = colMeans(terminalUtility.tc01.bench)
terminalUtility.tc01.sd.bench = colSds(terminalUtility.tc01.bench)

logReturn.tc01.bench = log(terminalWealth.tc01.bench)
logReturn.tc01.mean.bench = colMeans(logReturn.tc01.bench)
volatility.tc01.bench = sdLogReturn.tc01.bench * sqrt(nTimePoints)
volatility.tc01.mean.bench = colMeans(volatility.tc01.bench)
excessReturn.tc01.bench = logReturn.tc01.bench - rent
sharpeRatio.tc01.bench = excessReturn.tc01.bench / volatility.tc01.bench
sharpeRatio.tc01.mean.bench = colMeans(sharpeRatio.tc01.bench)
volOfVol.tc01.bench = colSds(volatility.tc01.bench)
correlation.tc01.bench = colCorrs(logReturn.tc01.bench,volatility.tc01.bench)

terminalWealth.tc01 = matrix(NA,nSims,n.entries)
```

```
sdWealth.tc01 = matrix(NA,nSims,n.entries)
sdLogReturn.tc01 = matrix(NA,nSims,n.entries)
transCost.tc01 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.tc01[,k] = rebStrategy.stochVol.tc01[[c(k,7)]]$simWealth.terminal
  sdWealth.tc01[,k] = rebStrategy.stochVol.tc01[[c(k,7)]]$simWealth.sd
  sdLogReturn.tc01[,k] = rebStrategy.stochVol.tc01[[c(k,7)]]$simWealth.logReturn.sd
  transCost.tc01[,k] = rebStrategy.stochVol.tc01[[c(k,7)]]$totalTransCost
}

terminalWealth.tc01.mean = colMeans(terminalWealth.tc01)
transCost.tc01.mean = colMeans(transCost.tc01)
terminalUtility.tc01 = utility(terminalWealth.tc01,riskAversion)
terminalUtility.tc01.mean = colMeans(terminalUtility.tc01)
terminalUtility.tc01.sd = colSds(terminalUtility.tc01)
lossOfUtility.tc01.mean = terminalUtility.tc01.mean.bench - terminalUtility.tc01.mean
lossOfUtility.tc01.mean.CL.lower = lossOfUtility.tc01.mean - qAlpha.half * (1/sqrt(nSims)) * sqrt(terminalUtility.tc01.sd.bench^2 + terminalUtility.tc01.sd^2)
lossOfUtility.tc01.mean.CL.upper = lossOfUtility.tc01.mean + qAlpha.half * (1/sqrt(nSims)) * sqrt(terminalUtility.tc01.sd.bench^2 + terminalUtility.tc01.sd^2)

lossOfUtility.tc01.mean.prime = terminalUtility.tc01.mean.bench - terminalUtility.tc01.mean

logReturn.tc01 = log(terminalWealth.tc01)
logReturn.tc01.mean = colMeans(logReturn.tc01)
volatility.tc01 = sdLogReturn.tc01 * sqrt(nTimePoints)
volatility.tc01.mean = colMeans(volatility.tc01)
excessReturn.tc01 = logReturn.tc01 - rent
sharpeRatio.tc01 = excessReturn.tc01 / volatility.tc01
sharpeRatio.tc01.mean = colMeans(sharpeRatio.tc01)
sharpeRatio.tc01.sd = colSds(sharpeRatio.tc01)
sharpeRatio.tc01.mean.CL.lower = sharpeRatio.tc01.mean - qAlpha.half * sharpeRatio.tc01.sd / sqrt(nSims)
sharpeRatio.tc01.mean.CL.upper = sharpeRatio.tc01.mean + qAlpha.half * sharpeRatio.tc01.sd / sqrt(nSims)
volOfVol.tc01 = colSds(volatility.tc01)
correlation.tc01 = colCorrs(logReturn.tc01, volatility.tc01)

tab1.tc01 = matrix(NA,18,4)

for (i in 1:9) {
  tab1.tc01[2*i-1,] = c(terminalWealth.tc01.mean.bench[i],transCost.tc01.mean.bench[i],terminalUtility.tc01.mean.bench[i],0)
  tab1.tc01[2*i,]   = c(terminalWealth.tc01.mean[i],transCost.tc01.mean[i],terminalUtility.tc01.mean[i],lossOfUtility.tc01.mean[i])
}

tab1.tc01[,2] = tab1.tc01[,2] * 1e2
tab1.tc01[,4] = tab1.tc01[,4] * 1e2
tab1.tc01 = round(tab1.tc01,4)

for (i in 1:18) {
  tab1.tc01[i,2] = paste(tab1.tc01[i,2],"\\e{\\text-2}",sep="")
  tab1.tc01[i,4] = paste(tab1.tc01[i,4],"\\e{\\text-2}",sep="")
}
for (i in seq(1,17,2)) { tab1.tc01[i,4] = "-" }

printex(tab1.tc01)
```

```
tab2.tc01 = matrix(NA,18,5)

for (i in 1:9) {
  tab2.tc01[2*i-1,] = c(logReturn.tc01.mean.bench[i],volatility.tc01.mean.bench[i],sharpeRatio.tc01.mean.bench[i],volOfVol.tc01.bench[i],correlation.tc01.bench[i])
  tab2.tc01[2*i,] = c(logReturn.tc01.mean[i],volatility.tc01.mean[i],sharpeRatio.tc01.mean[i],volOfVol.tc01[i],correlation.tc01[i])
}

tab2.tc01[,1] = tab2.tc01[,1] * 1e2
tab2.tc01[,4] = tab2.tc01[,4] * 1e3
tab2.tc01 = round(tab2.tc01,4)

for (i in 1:18) {
  tab2.tc01[i,1] = paste(tab2.tc01[i,1],"\\e{\\text-2}",sep="")
  tab2.tc01[i,4] = paste(tab2.tc01[i,4],"\\e{\\text-3}",sep="")
}

printex(tab2.tc01)

scalar = 1e2
x.labels = strategyNames
x.title = "Rebalancing strategy"
y.title = expression(paste("Mean loss of utility",phantom(0) %*% 10^2))
x.ticks = 1:9
y.range = range(c(lossOfUtility.tc01.mean.CL.lower*scalar,lossOfUtility.tc01.mean.CL.upper*scalar))
niceplot(lossOfUtility.tc01.mean*scalar,xLabels=x.labels,xTitle=x.title,yTitle=y.title,y.addCustom=.2,figsPerPage=4,ylim=y.range)
abline(h=0,col="darkgray",lty=3)
abline(v=x.ticks,col="darkgray",lty=3)
nicelines(lossOfUtility.tc01.mean.CL.lower*scalar,lty=2)
nicelines(lossOfUtility.tc01.mean.CL.upper*scalar,lty=2)
legendText = c(expression(paste("(b) ",lambda*"=.01")))
nicelegend("left",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_stochVol_tc01",type="eps")

y.title = "Mean Sharpe ratio"
y.range = range(c(sharpeRatio.tc01.mean.CL.lower,sharpeRatio.tc01.mean.CL.upper))
niceplot(sharpeRatio.tc01.mean,xLabels=x.labels,xTitle=x.title,yTitle=y.title,figsPerPage=4,ylim=y.range)
abline(v=x.ticks,col="darkgray",lty=3)
nicelines(sharpeRatio.tc01.mean.CL.lower,lty=2)
nicelines(sharpeRatio.tc01.mean.CL.upper,lty=2)
legendText = c(expression(paste("(b) ",lambda*"=.01")))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/sharpeRatio_stochVol_tc01",type="eps")

#
# Calculating relevant statistics and plotting
# Transaction cost proportion = .02
#

cat("Transaction cost proportion = .02\n")

terminalWealth.tc02.bench = matrix(NA,nSims,n.entries)
sdWealth.tc02.bench = matrix(NA,nSims,n.entries)
sdLogReturn.tc02.bench = matrix(NA,nSims,n.entries)
transCost.tc02.bench = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.tc02.bench[,k] = rebStrategy.benchmark.tc02[[c(k,3)]]$simWealth.terminal
  sdWealth.tc02.bench[,k] = rebStrategy.benchmark.tc02[[c(k,3)]]$simWealth.sd
```

```
  sdLogReturn.tc02.bench[,k] = rebStrategy.benchmark.tc02[[c(k,3)]]$simWealth.logReturn.sd
  transCost.tc02.bench[,k] = rebStrategy.benchmark.tc02[[c(k,3)]]$totalTransCost
}

terminalWealth.tc02.mean.bench = colMeans(terminalWealth.tc02.bench)
transCost.tc02.mean.bench = colMeans(transCost.tc02.bench)
terminalUtility.tc02.bench = utility(terminalWealth.tc02.bench, riskAversion)
terminalUtility.tc02.mean.bench = colMeans(terminalUtility.tc02.bench)
terminalUtility.tc02.sd.bench = colSds(terminalUtility.tc02.bench)

logReturn.tc02.bench = log(terminalWealth.tc02.bench)
logReturn.tc02.mean.bench = colMeans(logReturn.tc02.bench)
volatility.tc02.bench = sdLogReturn.tc02.bench * sqrt(nTimePoints)
volatility.tc02.mean.bench = colMeans(volatility.tc02.bench)
excessReturn.tc02.bench = logReturn.tc02.bench - rent
sharpeRatio.tc02.bench = excessReturn.tc02.bench / volatility.tc02.bench
sharpeRatio.tc02.mean.bench = colMeans(sharpeRatio.tc02.bench)
volOfVol.tc02.bench = colSds(volatility.tc02.bench)
correlation.tc02.bench = colCorrs(logReturn.tc02.bench, volatility.tc02.bench)

terminalWealth.tc02 = matrix(NA, nSims, n.entries)
sdWealth.tc02 = matrix(NA, nSims, n.entries)
sdLogReturn.tc02 = matrix(NA, nSims, n.entries)
transCost.tc02 = matrix(NA, nSims, n.entries)
for (k in 1:n.entries) {
  terminalWealth.tc02[,k] = rebStrategy.stochVol.tc02[[c(k,7)]]$simWealth.terminal
  sdWealth.tc02[,k] = rebStrategy.stochVol.tc02[[c(k,7)]]$simWealth.sd
  sdLogReturn.tc02[,k] = rebStrategy.stochVol.tc02[[c(k,7)]]$simWealth.logReturn.sd
  transCost.tc02[,k] = rebStrategy.stochVol.tc02[[c(k,7)]]$totalTransCost
}

terminalWealth.tc02.mean = colMeans(terminalWealth.tc02)
transCost.tc02.mean = colMeans(transCost.tc02)
terminalUtility.tc02 = utility(terminalWealth.tc02, riskAversion)
terminalUtility.tc02.mean = colMeans(terminalUtility.tc02)
terminalUtility.tc02.sd = colSds(terminalUtility.tc02)
lossOfUtility.tc02.mean = terminalUtility.tc02.mean.bench - terminalUtility.tc02.mean
lossOfUtility.tc02.mean.CL.lower = lossOfUtility.tc02.mean - qAlpha.half * (1/sqrt(nSims)) * sqrt(terminalUtility.tc02.sd.bench^2 + terminalUtility.tc02.sd^2)
lossOfUtility.tc02.mean.CL.upper = lossOfUtility.tc02.mean + qAlpha.half * (1/sqrt(nSims)) * sqrt(terminalUtility.tc02.sd.bench^2 + terminalUtility.tc02.sd^2)

logReturn.tc02 = log(terminalWealth.tc02)
logReturn.tc02.mean = colMeans(logReturn.tc02)
volatility.tc02 = sdLogReturn.tc02 * sqrt(nTimePoints)
volatility.tc02.mean = colMeans(volatility.tc02)
excessReturn.tc02 = logReturn.tc02 - rent
sharpeRatio.tc02 = excessReturn.tc02 / volatility.tc02
sharpeRatio.tc02.mean = colMeans(sharpeRatio.tc02)
sharpeRatio.tc02.sd = colSds(sharpeRatio.tc02)
sharpeRatio.tc02.mean.CL.lower = sharpeRatio.tc02.mean - qAlpha.half * sharpeRatio.tc02.sd / sqrt(nSims)
sharpeRatio.tc02.mean.CL.upper = sharpeRatio.tc02.mean + qAlpha.half * sharpeRatio.tc02.sd / sqrt(nSims)
volOfVol.tc02 = colSds(volatility.tc02)
correlation.tc02 = colCorrs(logReturn.tc02, volatility.tc02)

tab1.tc02 = matrix(NA,18,4)

```

```
for (i in 1:9) {
  tab1.tc02[2*i-1,] = c(terminalWealth.tc02.mean.bench[i],transCost.tc02.mean.bench[i],terminalUtility.tc02.mean.bench[i],0)
  tab1.tc02[2*i,]   = c(terminalWealth.tc02.mean[i],transCost.tc02.mean[i],terminalUtility.tc02.mean[i],lossOfUtility.tc02.mean[i])
}

tab1.tc02[,2] = tab1.tc02[,2] * 1e2
tab1.tc02[,4] = tab1.tc02[,4] * 1e2
tab1.tc02 = round(tab1.tc02,4)

for (i in 1:18) {
  tab1.tc02[i,2] = paste(tab1.tc02[i,2],"\\e{\\text-2}",sep="")
  tab1.tc02[i,4] = paste(tab1.tc02[i,4],"\\e{\\text-2}",sep="")
}
for (i in seq(1,17,2)) { tab1.tc02[i,4] = "-" }

printex(tab1.tc02)

tab2.tc02 = matrix(NA,18,5)

for (i in 1:9) {
  tab2.tc02[2*i-1,] = c(logReturn.tc02.mean.bench[i],volatility.tc02.mean.bench[i],sharpeRatio.tc02.mean.bench[i],volOfVol.tc02.bench[i],correlation.tc02.bench[i])
  tab2.tc02[2*i,] = c(logReturn.tc02.mean[i],volatility.tc02.mean[i],sharpeRatio.tc02.mean[i],volOfVol.tc02[i],correlation.tc02[i])
}

tab2.tc02[,1] = tab2.tc02[,1] * 1e2
tab2.tc02[,4] = tab2.tc02[,4] * 1e3
tab2.tc02 = round(tab2.tc02,4)

for (i in 1:18) {
  tab2.tc02[i,1] = paste(tab2.tc02[i,1],"\\e{\\text-2}",sep="")
  tab2.tc02[i,4] = paste(tab2.tc02[i,4],"\\e{\\text-3}",sep="")
}

printex(tab2.tc02)

scalar = 1e2
x.labels = strategyNames
x.title = "Rebalancing strategy"
y.title = expression(paste("Mean loss of utility",phantom(0) %*% 10^2))
x.ticks = 1:9
y.range = range(c(lossOfUtility.tc02.mean.CL.lower*scalar,lossOfUtility.tc02.mean.CL.upper*scalar))
niceplot(lossOfUtility.tc02.mean*scalar,xLabels=x.labels,xTitle=x.title,yTitle=y.title,y.addCustom=.2,figsPerPage=4,ylim=y.range)
abline(h=0,col="darkgray",lty=3)
abline(v=x.ticks,col="darkgray",lty=3)
nicelines(lossOfUtility.tc02.mean.CL.lower*scalar,lty=2)
nicelines(lossOfUtility.tc02.mean.CL.upper*scalar,lty=2)
legendText = c(expression(paste("(c) ",lambda*"=.02")))
nicelegend("left",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_stochVol_tc02",type="eps")

y.title = "Mean Sharpe ratio"
y.range = range(c(sharpeRatio.tc02.mean.CL.lower,sharpeRatio.tc02.mean.CL.upper))
niceplot(sharpeRatio.tc02.mean,xLabels=x.labels,xTitle=x.title,yTitle=y.title,figsPerPage=4,ylim=y.range)
abline(v=x.ticks,col="darkgray",lty=3)
nicelines(sharpeRatio.tc02.mean.CL.lower,lty=2)
nicelines(sharpeRatio.tc02.mean.CL.upper,lty=2)
```

```
legendText = c(expression(paste("(c) ",lambda*"=.02")))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/sharpeRatio_stochVol_tc02",type="eps")

#
# Calculating relevant statistics and plotting
# Transaction cost proportion = .03
#

cat("\nTransaction cost proportion = .03\n\n")

terminalWealth.tc03.bench = matrix(NA,nSims,n.entries)
sdWealth.tc03.bench = matrix(NA,nSims,n.entries)
sdLogReturn.tc03.bench = matrix(NA,nSims,n.entries)
transCost.tc03.bench = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.tc03.bench[,k] = rebStrategy.benchmark.tc03[[c(k,3)]]$simWealth
      .terminal
  sdWealth.tc03.bench[,k] = rebStrategy.benchmark.tc03[[c(k,3)]]$simWealth.sd
  sdLogReturn.tc03.bench[,k] = rebStrategy.benchmark.tc03[[c(k,3)]]$simWealth.
      logReturn.sd
  transCost.tc03.bench[,k] = rebStrategy.benchmark.tc03[[c(k,3)]]$totalTransCost
}

terminalWealth.tc03.mean.bench = colMeans(terminalWealth.tc03.bench)
transCost.tc03.mean.bench = colMeans(transCost.tc03.bench)
terminalUtility.tc03.bench = utility(terminalWealth.tc03.bench,riskAversion)
terminalUtility.tc03.mean.bench = colMeans(terminalUtility.tc03.bench)
terminalUtility.tc03.sd.bench = colSds(terminalUtility.tc03.bench)

logReturn.tc03.bench = log(terminalWealth.tc03.bench)
logReturn.tc03.mean.bench = colMeans(logReturn.tc03.bench)
volatility.tc03.bench = sdLogReturn.tc03.bench * sqrt(nTimePoints)
volatility.tc03.mean.bench = colMeans(volatility.tc03.bench)
excessReturn.tc03.bench = logReturn.tc03.bench - rent
sharpeRatio.tc03.bench = excessReturn.tc03.bench / volatility.tc03.bench
sharpeRatio.tc03.mean.bench = colMeans(sharpeRatio.tc03.bench)
volOfVol.tc03.bench = colSds(volatility.tc03.bench)
correlation.tc03.bench = colCorrs(logReturn.tc03.bench, volatility.tc03.bench)

terminalWealth.tc03 = matrix(NA,nSims,n.entries)
sdWealth.tc03 = matrix(NA,nSims,n.entries)
sdLogReturn.tc03 = matrix(NA,nSims,n.entries)
transCost.tc03 = matrix(NA,nSims,n.entries)
for (k in 1:n.entries) {
  terminalWealth.tc03[,k] = rebStrategy.stochVol.tc03[[c(k,7)]]$simWealth.
      terminal
  sdWealth.tc03[,k] = rebStrategy.stochVol.tc03[[c(k,7)]]$simWealth.sd
  sdLogReturn.tc03[,k] = rebStrategy.stochVol.tc03[[c(k,7)]]$simWealth.logReturn
      .sd
  transCost.tc03[,k] = rebStrategy.stochVol.tc03[[c(k,7)]]$totalTransCost
}

terminalWealth.tc03.mean = colMeans(terminalWealth.tc03)
transCost.tc03.mean = colMeans(transCost.tc03)
terminalUtility.tc03 = utility(terminalWealth.tc03,riskAversion)
terminalUtility.tc03.mean = colMeans(terminalUtility.tc03)
terminalUtility.tc03.sd = colSds(terminalUtility.tc02)
lossOfUtility.tc03.mean = terminalUtility.tc03.mean.bench - terminalUtility.tc03
    .mean
lossOfUtility.tc03.mean.CL.lower = lossOfUtility.tc03.mean - qAlpha.half * (1/
    sqrt(nSims)) * sqrt(terminalUtility.tc03.sd.bench^2 + terminalUtility.tc03.
    sd^2)
lossOfUtility.tc03.mean.CL.upper = lossOfUtility.tc03.mean + qAlpha.half * (1/
    sqrt(nSims)) * sqrt(terminalUtility.tc03.sd.bench^2 + terminalUtility.tc03.
```

```
sd^2)

logReturn.tc03 = log(terminalWealth.tc03)
logReturn.tc03.mean = colMeans(logReturn.tc03)
volatility.tc03 = sdLogReturn.tc03 * sqrt(nTimePoints)
volatility.tc03.mean = colMeans(volatility.tc03)
excessReturn.tc03 = logReturn.tc03 - rent
sharpeRatio.tc03 = excessReturn.tc03 / volatility.tc03
sharpeRatio.tc03.mean = colMeans(sharpeRatio.tc03)
sharpeRatio.tc03.sd = colSds(sharpeRatio.tc03)
sharpeRatio.tc03.mean.CL.lower = sharpeRatio.tc03.mean - qAlpha.half * sharpeRatio.tc03.sd / sqrt(nSims)
sharpeRatio.tc03.mean.CL.upper = sharpeRatio.tc03.mean + qAlpha.half * sharpeRatio.tc03.sd / sqrt(nSims)
volOfVol.tc03 = colSds(volatility.tc03)
correlation.tc03 = colCorrs(logReturn.tc03, volatility.tc03)

tab1.tc03 = matrix(NA,18,4)

for (i in 1:9) {
  tab1.tc03[2*i-1,] = c(terminalWealth.tc03.mean.bench[i],transCost.tc03.mean.bench[i],terminalUtility.tc03.mean.bench[i],0)
  tab1.tc03[2*i,]   = c(terminalWealth.tc03.mean[i],transCost.tc03.mean[i],terminalUtility.tc03.mean[i],lossOfUtility.tc03.mean[i])
}

tab1.tc03[,2] = tab1.tc03[,2] * 1e2
tab1.tc03[,4] = tab1.tc03[,4] * 1e2
tab1.tc03 = round(tab1.tc03,4)

for (i in 1:18) {
  tab1.tc03[i,2] = paste(tab1.tc03[i,2],"\\e{\\text-2}",sep="")
  tab1.tc03[i,4] = paste(tab1.tc03[i,4],"\\e{\\text-2}",sep="")
}
for (i in seq(1,17,2)) { tab1.tc03[i,4] = "-" }

printex(tab1.tc03)

tab2.tc03 = matrix(NA,18,5)

for (i in 1:9) {
  tab2.tc03[2*i-1,] = c(logReturn.tc03.mean.bench[i],volatility.tc03.mean.bench[i],sharpeRatio.tc03.mean.bench[i],volOfVol.tc03.bench[i],correlation.tc03.bench[i])
  tab2.tc03[2*i,] = c(logReturn.tc03.mean[i],volatility.tc03.mean[i],sharpeRatio.tc03.mean[i],volOfVol.tc03[i],correlation.tc03[i])
}

tab2.tc03[,1] = tab2.tc03[,1] * 1e2
tab2.tc03[,4] = tab2.tc03[,4] * 1e3
tab2.tc03 = round(tab2.tc03,4)

for (i in 1:18) {
  tab2.tc03[i,1] = paste(tab2.tc03[i,1],"\\e{\\text-2}",sep="")
  tab2.tc03[i,4] = paste(tab2.tc03[i,4],"\\e{\\text-3}",sep="")
}

printex(tab2.tc03)

scalar = 1e2
x.labels = strategyNames
x.title = "Rebalancing strategy"
y.title = expression(paste("Mean loss of utility",phantom(0) %*% 10^2))
x.ticks = 1:9
```

```
y.range = range(c(lossOfUtility.tc03.mean.CL.lower*scalar,lossOfUtility.tc03.mean.CL.upper*scalar))
niceplot(lossOfUtility.tc03.mean*scalar,xLabels=x.labels,xTitle=x.title,yTitle=y.title,y.addCustom=.2,figsPerPage=4,ylim=y.range)
abline(h=0,col="darkgray",lty=3)
abline(v=x.ticks,col="darkgray",lty=3)
nicelines(lossOfUtility.tc03.mean.CL.lower*scalar,lty=2)
nicelines(lossOfUtility.tc03.mean.CL.upper*scalar,lty=2)
legendText = c(expression(paste("(d) ",lambda*"=.03")))
nicelegend("left",legendText,bty="n",bg="white",cex=.7)
savePlot("images/lossOfUtility_stochVol_tc03",type="eps")

y.title = "Mean Sharpe ratio"
y.range = range(c(sharpeRatio.tc03.mean.CL.lower,sharpeRatio.tc03.mean.CL.upper))
niceplot(sharpeRatio.tc03.mean,xLabels=x.labels,xTitle=x.title,yTitle=y.title,figsPerPage=4,ylim=y.range)
abline(v=x.ticks,col="darkgray",lty=3)
nicelines(sharpeRatio.tc03.mean.CL.lower,lty=2)
nicelines(sharpeRatio.tc03.mean.CL.upper,lty=2)
legendText = c(expression(paste("(d) ",lambda*"=.03")))
nicelegend("topleft",legendText,bty="n",bg="white",cex=.7)
savePlot("images/sharpeRatio_stochVol_tc03",type="eps")

# Plotting transaction cost histograms, lambda = .01

graphics.off()

x.title = "Total transaction cost"
y.title = "Frequency"
breaksLength = 70

# Hourly rebalancings
dataSet1 = transCost.tc01[,1]
dataSet2 = transCost.tc01.bench[,1]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(a) ",lambda*"=.01"),"T.c. strategy : Preceding","Reb. strategy : Hourly"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

# Daily rebalancings
dataSet1 = transCost.tc01[,3]
dataSet2 = transCost.tc01.bench[,3]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(b) ",lambda*"=.01"),"T.c. strategy : Preceding","Reb. strategy : Daily"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

```

```
savePlot("images/hist_transCost01_stochVol_HourlyDaily",type="eps")

# 'Every 3rd day' rebalancings
dataSet1 = transCost.tc01[,4]
dataSet2 = transCost.tc01.bench[,4]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(c) ",lambda*"=.01")),"T.c. strategy : Preceding","Reb. strategy : Ev. 3rd day"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

# 'Every 12th day' rebalancings
dataSet1 = transCost.tc01[,5]
dataSet2 = transCost.tc01.bench[,5]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(d) ",lambda*"=.01")),"T.c. strategy : Preceding","Reb. strategy : Ev. 12th day"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

savePlot("images/hist_transCost01_stochVol_3rd12th",type="eps")

# Monthly rebalancings
dataSet1 = transCost.tc01[,6]
dataSet2 = transCost.tc01.bench[,6]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(e) ",lambda*"=.01")),"T.c. strategy : Preceding","Reb. strategy : Monthly"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

# Bimonthly rebalancings
dataSet1 = transCost.tc01[,7]
dataSet2 = transCost.tc01.bench[,7]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,nCol=2,ylim=y.lim,breaks=res)
```

```
legendText = c(expression(paste("(f) ",lambda*"=.01"),"T.c. strategy : Preceding
    ","Reb. strategy : Bimonthly"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

savePlot("images/hist_transCost01_stochVol_MonthlyBi",type="eps")

# Semiannual rebalancings
dataSet1 = transCost.tc01[,8]
dataSet2 = transCost.tc01.bench[,8]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(e) ",lambda*"=.01"),"T.c. strategy : Preceding
    ","Reb. strategy : Semiannual"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

# Annual rebalancings
dataSet1 = transCost.tc01[,9]
dataSet2 = transCost.tc01.bench[,9]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,2*x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,nCol=2,ylim
    =y.lim,breaks=res)
legendText = c(expression(paste("(f) ",lambda*"=.01"),"T.c. strategy : Preceding
    ","Reb. strategy : Annual"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addObj = addHist(dataSet2,breaks=res,density=30)

savePlot("images/hist_transCost01_stochVol_SemiAnnual",type="eps")

#
# Plotting transaction cost histograms, lambda = .02
#

# Hourly rebalancings
dataSet1 = transCost.tc02[,1]
dataSet2 = transCost.tc02.bench[,1]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(a) ",lambda*"=.02"),"T.c. strategy : Preceding
    ","Reb. strategy : Hourly"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

# Daily rebalancings
dataSet1 = transCost.tc02[,3]
dataSet2 = transCost.tc02.bench[,3]
```

```
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,nCol=2,ylim
    =y.lim,breaks=res)
legendText = c(expression(paste("(b) ",lambda*"=.02")),"T.c. strategy : Preceding
    ","Reb. strategy : Daily"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

savePlot("images/hist_transCost02_stochVol_HourlyDaily",type="eps")

# 'Every 3rd day' rebalancings
dataSet1 = transCost.tc02[,4]
dataSet2 = transCost.tc02.bench[,4]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(c) ",lambda*"=.02")),"T.c. strategy : Preceding
    ","Reb. strategy : Ev. 3rd day"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

# 'Every 12th day' rebalancings
dataSet1 = transCost.tc02[,5]
dataSet2 = transCost.tc02.bench[,5]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,nCol=2,ylim
    =y.lim,breaks=res)
legendText = c(expression(paste("(d) ",lambda*"=.02")),"T.c. strategy : Preceding
    ","Reb. strategy : Ev. 12th day"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

savePlot("images/hist_transCost02_stochVol_3rd12th",type="eps")

# Monthly rebalancings
dataSet1 = transCost.tc02[,6]
dataSet2 = transCost.tc02.bench[,6]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(e) ",lambda*"=.02")),"T.c. strategy : Preceding
    ","Reb. strategy : Monthly"))
nicelegend("topleft",legendText,bty="n",cex=.7)
```

```
addHist(dataSet2,breaks=res,density=30)

# Bimonthly rebalancings
dataSet1 = transCost.tc02[,7]
dataSet2 = transCost.tc02.bench[,7]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(f) ",lambda*"=.02"),"T.c. strategy : Preceding","Reb. strategy : Bimonthly"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

savePlot("images/hist_transCost02_stochVol_MonthlyBi",type="eps")

# Semiannual rebalancings
dataSet1 = transCost.tc02[,8]
dataSet2 = transCost.tc02.bench[,8]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(e) ",lambda*"=.02"),"T.c. strategy : Preceding","Reb. strategy : Semiannual"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

# Annual rebalancings
dataSet1 = transCost.tc02[,9]
dataSet2 = transCost.tc02.bench[,9]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,2*x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(f) ",lambda*"=.02"),"T.c. strategy : Preceding","Reb. strategy : Annual"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addObj = addHist(dataSet2,breaks=res,density=30)

savePlot("images/hist_transCost02_stochVol_SemiAnnual",type="eps")

#
# Plotting transaction cost histograms, lambda = .03
#

# Hourly rebalancings
dataSet1 = transCost.tc03[,1]
dataSet2 = transCost.tc03.bench[,1]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
```

```
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(a) ",lambda*"=.03")),"T.c. strategy : Preceding ","Reb. strategy : Hourly"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

# Daily rebalancings
dataSet1 = transCost.tc03[,3]
dataSet2 = transCost.tc03.bench[,3]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(b) ",lambda*"=.03")),"T.c. strategy : Preceding ","Reb. strategy : Daily"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

savePlot("images/hist_transCost03_stochVol_HourlyDaily",type="eps")

# 'Every 3rd day' rebalancings
dataSet1 = transCost.tc03[,4]
dataSet2 = transCost.tc03.bench[,4]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(c) ",lambda*"=.03")),"T.c. strategy : Preceding ","Reb. strategy : Ev. 3rd day"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

# 'Every 12th day' rebalancings
dataSet1 = transCost.tc03[,5]
dataSet2 = transCost.tc03.bench[,5]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(d) ",lambda*"=.03")),"T.c. strategy : Preceding ","Reb. strategy : Ev. 12th day"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

savePlot("images/hist_transCost03_stochVol_3rd12th",type="eps")
```

```
# Monthly rebalancings
dataSet1 = transCost.tc03[,6]
dataSet2 = transCost.tc03.bench[,6]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(e) ",lambda*"=.03")),"T.c. strategy : Preceding
    ","Reb. strategy : Monthly"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

# Bimonthly rebalancings
dataSet1 = transCost.tc03[,7]
dataSet2 = transCost.tc03.bench[,7]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,nCol=2,ylim
    =y.lim,breaks=res)
legendText = c(expression(paste("(f) ",lambda*"=.03")),"T.c. strategy : Preceding
    ","Reb. strategy : Bimonthly"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

savePlot("images/hist_transCost03_stochVol_MonthlyBi",type="eps")

# Semiannual rebalancings
dataSet1 = transCost.tc03[,8]
dataSet2 = transCost.tc03.bench[,8]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,nCol=2,ylim=y.lim,breaks=res)
legendText = c(expression(paste("(e) ",lambda*"=.03")),"T.c. strategy : Preceding
    ","Reb. strategy : Semiannual"))
nicelegend("topleft",legendText,bty="n",cex=.7)
addHist(dataSet2,breaks=res,density=30)

# Annual rebalancings
dataSet1 = transCost.tc03[,9]
dataSet2 = transCost.tc03.bench[,9]
x.range = range(c(dataSet1,dataSet2))
x.min = min(x.range)
x.max = max(x.range)
res = seq(x.min,2*x.max,length=breaksLength)
histObject1 = hist(dataSet1,breaks=res,plot=F)
histObject2 = hist(dataSet2,breaks=res,plot=F)
y.lim = range(c(histObject1$counts,histObject2$counts)) * 1.3
nicehist(dataSet1,xTitle=x.title,yTitle=y.title,multiPlot=T,newDev=F,nCol=2,ylim
    =y.lim,breaks=res)
legendText = c(expression(paste("(f) ",lambda*"=.03")),"T.c. strategy : Preceding
    ","Reb. strategy : Annual"))
```

```
nicelegend("topleft",legendText,bty="n",cex=.7)
addObj = addHist(dataSet2,breaks=res,density=30)

savePlot("images/hist_transCost03_stochVol_SemiAnnual",type="eps")
```

Bibliography

[1] Fred Espen Benth. *Matematisk finans*. 3rd ed. Universitetsforlaget, 2007.

[2] *CIR process*. URL: http://en.wikipedia.org/wiki/CIR_process.

[3] John C. Cox, Jonathan E. Ingersoll Jr., and Stephen A. Ross. "A theory of the term structure of interest rates". In: *Econometrica* 53 (Mar. 1985), pp. 385–408.

[4] M. H. A. Davies and A. R. Norman. "Portfolio selection with transaction costs". In: *Mathematics of Operations Research* 15 (Nov. 1990), pp. 676–713.

[5] *Discretization*. URL: http://en.wikipedia.org/wiki/Discretization.

[6] Jon Exley, Shyam Mehta, and Andrew Smith. "Mean reversion". In: (June 2004).

[7] *Geometric Brownian motion*. URL: http://en.wikipedia.org/wiki/Geometric_Brownian_motion.

[8] David Harper. *An introduction to value at risk*. URL: http://www.investopedia.com/articles/04/092904.asp.

[9] Steven Heston. "A closed-form solution for options with stochastic volatility with applications to bond and currency options". In: *The review of financial studies* 6 (Summer 1993), pp. 327–343.

[10] Alireza Javaheri. *Inside volatility arbitrage: the secrets of skewness*. 1st ed. John Wiley & sons, inc., 2005.

[11] Philippe Jorion. *Value at risk*. 3rd ed. The McGraw-Hill companies, 2007.

[12] Peter E. Kloeden and Eckhard Platen. *Numerical solution of stochastic differential equations*. 1st ed. Springer-Verlag, 1992.

[13] Athanasios Koulakiotis, Nicholas Papasyriopoulos, and Phil Molyneux. "More evidence on the relationship between stock price returns and volatility: A note". In: *International research journal of finance and economics* 1 (Jan. 2006), pp. 21–28.

[14] Bernt Øksendal. *Stochastic differential equations*. 6th ed. Springer-Verlag, 2007.

[15] Robert C. Merton. "Lifetime portfolio selection under uncertainty: The continuous-time case". In: *The Review of Economics and Statistics* 51 (Aug. 1969), pp. 247–257.

[16] Nimalin Moodley. "The Heston model: a practical approach with Matlab code". Honours project. University of the Witwatersrand, 2005.

[17] William F. Sharpe. "The Sharpe ratio". In: *The Journal of Portfolio Management* 21 (Fall 1994), pp. 49–58.

[18] *Sharpe ratio*. URL: `http://en.wikipedia.org/wiki/Sharpe_ratio`.

[19] *Stochastic volatility*. URL: `http://en.wikipedia.org/wiki/Stochastic_volatility`.