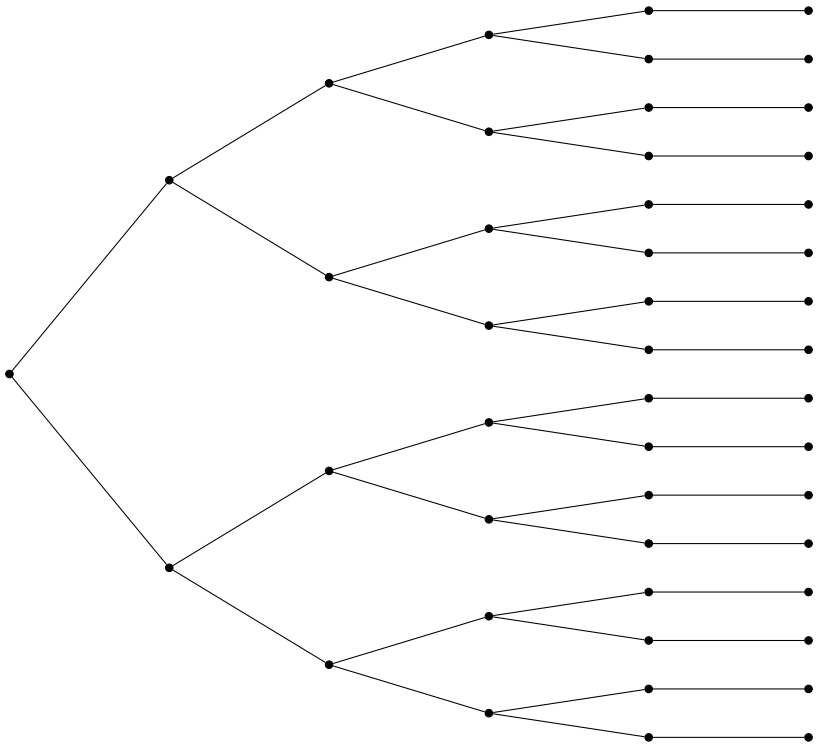
CHALMERS
UNIVERSITY OF TECHNOLOGY



Constrained Portfolio Optimization in Liability-Driven Investing

Master's Thesis in Computer Science: Algorithms, Languages and Logic

FILIP HALLQVIST

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019:NN

Constrained Portfolio Optimization in Liability-Driven Investing

FILIP HALLQVIST

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2019

Constrained Portfolio Optimization in Liability-Driven Investing
FILIP HALLQVIST

Supervisor: Tor Nordqvist, Captor AB
Supervisor: Martin Karrin, Captor AB
Examiner: Holger Rootzèn, Department of Mathematical Sciences

Master's Thesis 2019:NN
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A scenario tree for a multi-stage portfolio optimization problem with a total of 16 scenarios.

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2019

Constrained Portfolio Optimization in Liability-Driven Investing
FILIP HALLQVIST
Department of Mathematical Sciences Chalmers University of Technology

Abstract

In this thesis we formulate and implement a multi-stage portfolio optimization model, and solve it using a genetic algorithm. The goals of the thesis are, apart from formulating and implementing the problem, to estimate suitable parameters for the scenario generation, and to make sure that the problem is solved in a computationally efficient manner. Lastly, we investigate and discuss the performance of the complete system, including financial aspects of the produced solutions, the stability of the solutions, and the computational complexity of the model. We find that our problem formulation is useful, and that it it allows for great flexibility with regards to adding new constraints. We also find that our genetic algorithm solves the problem in reasonable time. Before the model can be used in practice however, results show that it needs to be improved with regards to stability in the solutions.

Keywords: portfolio, optimization, genetic, algorithm, asset, management, liability, driven, investing

Acknowledgements

I would like to thank Holger Rootzén for his constructive suggestions, insights, guidance, and enthusiastic encouragement throughout the project. I am also particularly grateful for the assistance given by Tor Nordqvist and Martin Karrin at Captor for their ideas and invaluable support. Furthermore, I would like to thank Josefine Bofeldt and Sara Joon for taking the time to discuss various aspects of the project. Special thanks to Filip Slottner Seholm and Herman Carlström for their continuous feedback during the thesis.

Filip Hallqvist, Gothenburg, June 2019

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background and motivation

Following the large financial crisis of 2009, the European Insurance and Occupational Pensions Authority released the Solvency II directive [24]. The directive included regulations for insurance companies on how to assess the value of their liabilities, instructions on minimum capital requirements, and guidance on what financial information to disclose and to whom. Even though it took nearly 10 years to implement the directive, being lawfully obliged to manage both their assets and liabilities was nothing new for insurance companies at the time. Back in 1982 however, it was [1]. With skyrocketing interest rates, insolvencies in the savings and loans industry became commonplace, and so in the ripples after the impasse had settled, it was clear to the whole industry that improved regulations were necessary [25]. This led to the birth of *asset-liability management* (ALM) as a practice.

ALM is a risk management methodology focusing on the liabilities in contrast to the assets of an insurance company or financial institution. Consider the case of a pension fund for an example. The liabilities of a pension fund mostly consist of the money that it owes to its beneficiaries as they enter retirement age. This money can be perceived as a loan with shorter or longer maturity, depending on the age of the beneficiaries. The value of this loan changes depending on current interest rates, and so by performing ALM studies the pension fund strives to manage the risks arising from changes in interest rates in the future.

In this sense, the historically low interest rates pose an interesting issue: How should insurance companies invest their money so they can repay liabilities, without taking too much risk? A reasonably safe hedge against its liabilities would be to invest all of the money into bonds with the same maturity as the time to retirement of its beneficiaries. Unfortunately, the liquidity of the bond market dries up for longer maturities, and even if this wasn't the case, investing only in interest rates would still not yield enough return to match the liabilities [23].

What further complicates the issue is that the liabilities of pension funds span over several decades. Evaluating the chances of a worldwide recession occurring in the near future is fairly easy (it is very unlikely that it will happen tomorrow), but when looking far into the future, the task becomes a lot harder (it is not too unlikely that

it will happen sometime in the next hundred years) [62]. This puts a lot of pressure on the financial models we use today. Not only should they be accurate enough for the short term, a challenge which is hard to conquer on its own, but they must also be robust enough to stand the test of time.

One solution, suggested in [75], [22], and [65], is to mathematically formulate the problem in such a way that one can evaluate several market scenarios over several time steps, so-called *multi-stage stochastic programming* as will be described in Section 2.2.1. By using this approach, we can reap the benefits of producing more robust results. But on the other hand, it gives rise to a new problem, namely: how do we solve the optimization problem in a computationally efficient manner? The computational complexity of the formulation grows exponentially with the number of scenarios, and therefore it is critical that the solver is fast enough to handle the jump in complexity [75]. Moreover, if we were to use off-the-shelf optimization software to solve the optimization problem, we might be limited by the way the software expects us to specify parameters and inputs.

An alternative to using off-the-shelf software is by implementing a genetic algorithm to solve the optimization problem. There is no true novelty in applying genetic algorithms to portfolio optimization problems in the general case. However, there is seemingly little research for the case where the end user of the portfolio optimization is a large institution such as a pension fund, which is shown in Section 3. This leads us to the main research question of this thesis, namely:

How can we define a multi-stage stochastic optimization model in an asset-liability setting, and solve in a computationally efficient manner using a genetic algorithm?

1.2 Goals and limitations

The primary research objectives are defined as follows:

- Define a multi-stage stochastic optimization model in an asset-liability management setting
- Implement a genetic algorithm which can solve the optimization problem in a computationally efficient manner
- Generate scenario trees based on historical data
- Investigate and discuss the performance of the complete system, including financial aspects of the suggested solution, stability of the solution, and computational complexity.

To finish the project within the given time limit, the model will be simplified to some extent.

First and foremost, we assume that liabilities move exactly like a bond with a

maturity of 20 years. In reality, the liabilities of a pension fund depends on several different parameters, such as interest rates, changes in salaries and tax rates, life expectancy of the population, and so on [52].

Furthermore, the value of assets and liabilities are only affected by changes in the underlying risk factors. This means that the model does not, for example, consider cases where the number of participants in the pension plan varies through time. The model also assumes that the participants have made a lump sum payment before the simulation starts, and do not perform further payments. Naturally, most defined benefit pension plans involve participants infusing money into the plan on a monthly or annual basis [8].

Our model includes a total of 12 instruments, 2 of which are bonds, and 3 of which are financial derivatives. A bond is merely a contract which allows one party to borrow money, just to pay all of the money back at a future date. Usually, the borrower is also obliged to pay an interest rate to its counterparty. Sometimes, this interest rate is paid in annual or semi-annual *coupons*. Bonds which involve the interest rate being paid in the form of coupons are called *coupon bonds*. The bonds in our model however, do not include coupons, but rather we use so called *zero-coupon bonds*. Zero-coupon bonds incorporate the interest rate by trading at a price *below* (or, if the interest rate is negative, above) their intrinsic value. This simplifies the implementation aspect, since we only have to keep track of prices and not both prices and coupons.

Moreover, we renew the batch of bonds at each time step. As an example, if we buy a zero-coupon bond with maturity T, we are allowed to hold it for one time period δt, at which point its tenor has decreased by δt. To compute its value after δt, we compare it with a new zero-coupon bond with maturity $T - \delta t$. We can then compute the profit or loss from holding the bond for one time period, but then we are forced to sell it and buy a new one. Of course, in reality, zero-coupon bonds can be held until they reach maturity. Refer to Section 5.1.1 for a more in-depth explanation on the matter.

1.3 Structure of the thesis

The structure of the thesis is outlined as follows. In Chapter 2 we give a theoretical background on relevant subjects. Chapter 3 contains descriptions of previous research on multi-stage stochastic programming, asset-liability management, and genetic algorithms. We then give a theoretical formulation of our optimization problem in Chapter 4. A description of how the theoretical formulation is implemented in practice, is given in Chapter 5. Finally, we present our findings in Chapter 6, and discuss the results in Chapter 7. We wrap up the thesis with a conclusion given in Chapter 8.

2

Theory

2.1 Modern portfolio theory

One of the most influential contributions to portfolio theory was made by Harry Markowitz in the early 1950's. Markowitz argued that not only should an investor strive to maximize the returns of an investment, but also consider the amount of risk coinciding with it. Markowitz formulated an optimization problem, parameterized on the so called *risk aversion* of the investor. The optimization problem maximizes the returns of a portfolio, while simultaneously minimizing risks. The risk aversion parameter describes the investors attitude towards risk. A low risk aversion value means that the investor is more willing to take on risk, whereas a high risk aversion value means that the investor is less willing to take on risk [45].

The mathematical formulation of the Markowitz model is as follows:

$$\begin{aligned} \max_{\mathbf{w}} \quad & \mu^\top \mathbf{w} - \lambda(\mathbf{w}^\top \Sigma \mathbf{w}), \\ \text{s.t.} \quad & \sum_{i}^{n} w_i = 1, \end{aligned}$$

where

μ	is the vector of expected returns for each asset,
$\mathbf{w}$	is the vector of asset weights,
Σ	is the covariance matrix of the assets,
λ	is a given parameter specifying the risk aversion of the investor,
n	is the total number of assets.

The definition given in this equation defines risk as variance, but one can also use other definitions, such as one of the following.

Value at risk (VaR)

$$\mathrm{VaR}_\alpha(X) = \min\{c : P(X \leq c) \geq \alpha\}$$

X is a random variable describing the return of some portfolio, and the α-*VaR* describes how much the investor stands to *at least* lose, with a probability of α, over some fixed time span, usually a year. Note that the metric does not give an upper bound on the potential losses, and as such is might be useful to combine the VaR with another risk metric.

Conditional value at risk (CVaR)

$$\mathrm{CVaR}_\alpha(X) = \mathbb{E}[X \mid X \geq \mathrm{VaR}_\alpha(X)]$$

An extension to the VaR metric is the *Conditional Value at Risk* (CVaR), also known as *Expected Shortfall* (ES). The CVaR describes the expected loss, given that it exceeds the α-VaR.

Target shortfall

$$B = \mathbb{E}[\max(T - R, 0)] \tag{2.1}$$

Like VaR and CVaR the target shortfall B is a downside risk metric. For a given target wealth, T, the target shortfall simply measures the negative difference between a target wealth and the total return of a portfolio, R. This is the risk metric that will be used in this thesis.

2.2 Stochastic programming

Many real-world optimization problems involve uncertainty in one way or another. For example, when predicting tomorrow's value of a stock price, or when predicting the number of cars sold during the next year. In these cases, ordinary optimization schemes might not ~~necessarily~~ suffice, and so we must make use of optimization techniques which are able to handle these uncertainties. One such technique is by formulating the optimization problem as a stochastic program, in which we minimize the *expectation* of the function.

As an example, borrowed from Shapiro and Philpott [55], let $G(x, D)$ be a goal function to be minimized, where D is a random variable. One can imagine that $G(x, D)$ is the production cost of some commodity, and the unknown variable D is the demand of the commodity one year from now. The decision variable x then denotes the amount to be produced.

Furthermore, assuming that we know the statistical distribution of D, we can formulate the problem as

$$\min_x \quad \mathbb{E}[G(x, D)]$$

where D is a discrete random variable, which takes the values $\{d_1, d_2, \ldots, d_k\}$, each one occurring with a fixed probability p_i. We call each value a *scenario*. For the production example, it follows that

$$\mathbb{E}[G(x, D)] = \sum_{i}^{k} p_i G(x, D), \qquad G(x, d) = cx - b[d - x]_+,$$

where

x	is the decision variable, denoting the amount to be produced,
c	is the cost of producing some commodity,
b	is the cost for storing over-producing the commodity,
d	is the demand of the commodity.

We can also extend the problem by adding constraints, and only in a few cases is it possible to find a closed form solution to the problem.

2.2.1 Multi-stage stochastic programming

A natural extension to stochastic programming involves optimizing over several so called stages. The idea is that we should only make our decisions based on data known at the present stage, and that optimal value for future stages should be considered uncertain. This is also known as the non-anticipativity constraint. Keeping the production example in mind, we can define a simple two-stage stochastic programming problem as

$$\min_x \quad ax + \mathbb{E}_\xi[Q(x, \xi)]$$

where cx is the cost of producing x items. $Q(x, \xi)$ is the optimal value of a new optimization problem, defined as

$$\begin{aligned} \min_y \quad & by, \\ \text{s.t.} \quad & Tx + Wy \leq h. \end{aligned}$$

Summarizing the variables in our formulations:

ax	is the cost of production in the first year,
by	is the cost of production in the second year,
$Tx + Wy \leq h$	handles inconsistencies in our system,
ξ	is the data available in the second year, $\xi = (b, T, W, h)$.

For the production example, $Tx + Wy \leq h$ might for example handle cases where, for some reason, producing x items makes it impossible to produce y items in the second year. Note that the constraints are linear. However, this is not to say that we cannot have more complex, non-linear constraints in a multi-stage optimization problem.

Furthermore, as the future is uncertain, we use ξ to represent a specific scenario. Recall that $Q(x, \xi)$ is the optimal value of the second-stage problem, and so by taking the expected value of all scenarios, we get $\mathbb{E}_\xi[Q(x, \xi)]$ as a simple way of capturing the uncertainty of the future.

The uncertainty gives rise to a stochastic data process

$$I = \{\omega_1, \omega_2 \ldots, \omega_n\},$$

where each ω_i is an event, and I is a path in the probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Simply put, I can be thought of as a *scenario*, or a sequence of observations. Again considering the production example, any I will contain exactly two events, since we model exactly two stages.

Note that the space of stochastic data processes $\{I^{(1)}, I^{(2)}, \ldots, I^{(m)}\}$ is infinitely large by definition, as it represents everything that can possibly happen. This is not an issue if the optimization problem can be solved analytically. However, more complex optimization problems must be solved numerically, and in these cases we are faced with the task of *discretizing* this infinitely large space into a countable set of scenarios.

The process of discretizing the probability space into a countable set of scenarios is called *scenario generation*, and is discussed in Section 2.2.2.

2.2.2 Scenario generation

As explained in Section 2.2.1, one of the key components of a multi-stage stochastic programming problem is *scenario generation*. Scenario generation is a broad research area, and an extensive list of surveys and technical articles on the subject is given in [75]. In a nutshell, scenario generation is the process of generating *scenario trees*, which are directed acyclic graphs describing the various sequences I of events that can occur in a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Each I is a scenario in our scenario tree, that is, a path of events from the root to a leaf node. An example of a scenario tree is depicted in Figure 2.1.

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2.2: A scenario tree with 7 states, stored as an adjacency matrix.

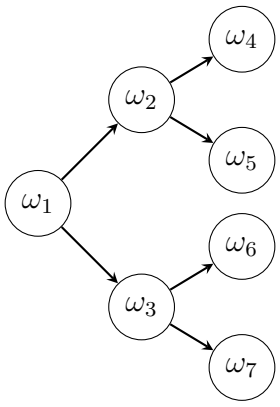


Figure 2.1: A scenario tree with 7 events.

A reasonably simple technique for generating scenarios is *sample average approximation* (SAA). The technique is closely related to monte-carlo methods, and is extensively described in [68]. The idea is to exploit the law of large numbers by naively sampling a large number of scenarios, and constructing trees out of them. The optimization part then involves evaluating all trees, and scoring each solution by taking the average of the scores resulting from evaluating the objective function over each individual tree [68].

In order to use the generated scenario trees in our model, we must make sure that they are stored in an appropriate structure. One of the simplest ways of organizing the scenario tree is by organizing scenarios as an *adjacency matrix*. An adjacency matrix is a matrix composed of 1's or 0's. The value at row i, column j, is 1 if there is an edge from node i to node j, and 0 otherwise. For example, consider Figure 2.1, where we have a binary scenario tree with a total of 7 states. The corresponding adjacency matrix is depicted in Figure 2.2.

A benefit with using adjacency matrices is that they allow modelling more complex types of trees. For example, there might be cases where we want two states in the same step to share a common successor. We do not have to change the structure of the adjacency matrix, but rather it is only a matter of flipping bits in the matrix.

A drawback with this approach, however, is that in most cases a lot of space is left

unused. This is due to the fact that during scenario generation, we produce directed acyclic graphs, which by definition means that at least half the matrix will consist of 0's. The reason is that the scenario tree is acyclic, and so for an adjacency matrix A, if we have $A_{i,j} = 1$ then clearly we must have $A_{j,i} = 0$. For our specific case, where the scenario trees are complete binary trees, the cons of using an adjacency matrix outweighs the pros.

An alternative to storing the scenario trees as matrices, is to store them as vectors. Consider Equation (2.2). At the first index of the vector we find the root of the tree, whereas at the last index we find the last leaf node.

$$B = [1, 2, 3, 4, 5, 6, 7] \tag{2.2}$$

By simple vector arithmetic, the parent of a node i can thus be found at $\lfloor i/2 \rfloor$. The left and right children of a node i can be found at $2i$ and $2i + 1$, respectively. Of course, these simple formulae can only be used if the branching is consistent throughout the tree, meaning that each non-leaf node in fact has exactly two children. Other implementations, [9] for example, mentions a 1-8-4-4-2-1-tree, meaning the trees have 8 nodes on the second level, 4 nodes on the third level, and so on. The simple vector arithmetic becomes a lot more involved, since the equations for finding parents and children of nodes depend on their level in the tree.

2.3 Quasi-random number generation

At the very base of simulation lies a random number generator. If we for example want to sample random numbers from a normal distribution, we can first generate a set of uniformly distributed variables in the range $[0, 1]$, and then transform the numbers so that they are normally distributed instead of uniformly distributed.

Several techniques exist for transforming uniformly distributed variables to normally distributed ones. For example, we can supply the numbers as inputs to the inverse normal cumulative distribution function (also known as the normal quantile function). There are also other methods such as the Ziggurat transform [46] or the Box-Muller transform [56].

For most intents and purposes, a *pseudo-random* number generator will be the right choice of generator. By pseudo-random, we mean that the generator is actually a deterministic algorithm which produces seemingly random numbers. However, for some applications, as in optimization, we would rather aim to explore the search space as fast as possible. Then it makes sense to use a random number generator producing numbers which are *evenly* spread out throughout the search space. These types of random number generators are denoted as *quasi-random*. Figure 2.3 shows the difference between using a quasi-random number generator and a pseudo-random number generator.

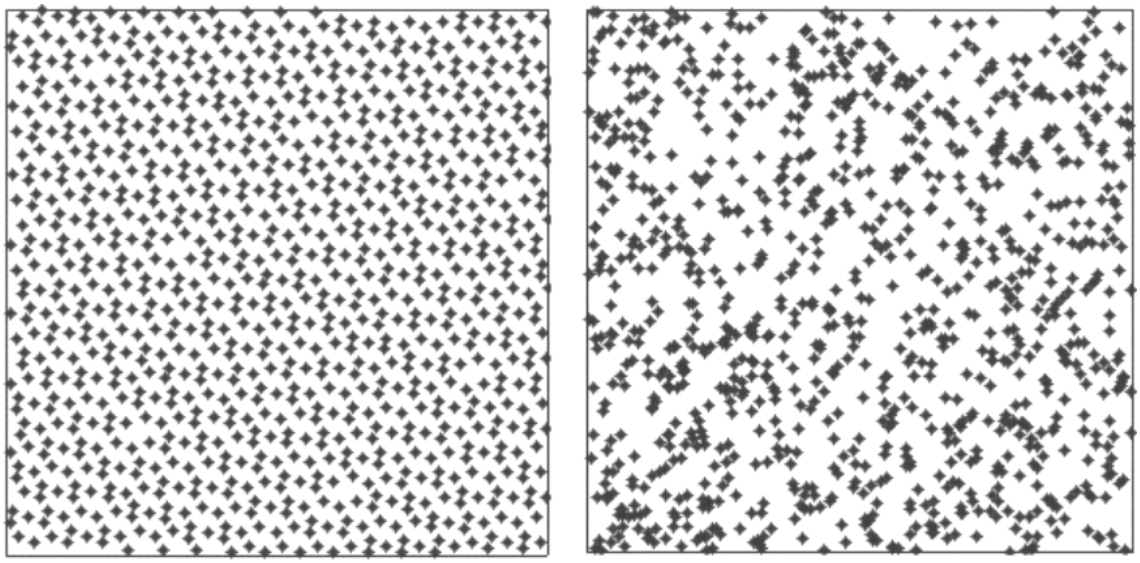


Figure 2.3: The left plot was produced using a quasi-random number generator, and the right plot was produced using a pseudo-random number generator.

To measure exactly how evenly spread out numbers are along multiple dimensions, we use the term *discrepancy*. Its mathematical formulation, originally defined in [51], can be seen in Equation (2.3). Let

$$D(J, N) = \mid \frac{n}{N} - V \mid \tag{2.3}$$

where

$\mathbb{I}^m$	is an m-dimensional half-open unit cube in the range $[0, 1)^m$,
J	is a set of points $\{x_1, x_2, \ldots, x_n\} \in \mathbb{I}^m$,
n	is the number of points in J,
N	is the total number of points in $\mathbb{I}^m$,
V	is the volume of the set J.

We also define here the *star discrepancy* $D^*(N) = \max_J \mid D(J, N) \mid$, as formulated in [51]. By minimizing the star discrepancy, one can produce a *low-discrepancy sequence*. Multiple low-discrepancy sequences have been defined throughout the years, one of the earlier examples being the Van Der Corput sequence [67], a more modern example being the Sobol sequence [59], which is the one we will use in this thesis.

The intricacies of the definition of the Sobol sequence makes it quite complex. A comparatively simple description of how to produce a Sobol sequence is given in [40], and we now briefly outline the process.

First, we generate a set $V = \{v_1, v_2, \ldots, v_n\}$ of so called *direction numbers*, $v_i = m_i/2^i$, where m_i is an odd integer, and $0 \geq m_i < 2^i$. Let d be some integer such that $d \geq 0$.

Now, for $i \leq d$, each m_i can be chosen freely. For $i > d$ however, we construct an arbitrary d-dimensional primitive polynomial P, as

$$P = x^d + a_1 x^{d-1} + \cdots + a_{d-1}x + 1.$$

Then, m_i and v_i for $i > d$ are computed using the coefficients of the *primitive polynomial* P,

$$\begin{aligned} v_i &= a_1 v_{i-1} \oplus a_2 v_{i-2} \oplus \cdots \oplus v_{i-d} \oplus 2^{-d} v_{i-d}, \\ m_i &= 2a_1 m_{i-1} \oplus 2^2 a_2 m_{i-2} \oplus \cdots \oplus 2^{d-1} a_{d-1} m_{i-d+1} \oplus 2^d m_{i-d} \oplus 2^{-d} m_{i-d}, \end{aligned}$$

where $\oplus$ is the bitwise XOR operation. Finally, the n:th number x_n of the one dimensional Sobol sequence is given by

$$x_n = b_1 v_1 \oplus b_2 v_2 \oplus \ldots$$

where each b_i is the i:th bit in the binary representation of n [40].

As for s-dimensional Sobol sequences, we simply construct s primitive polynomials $P_1, P_2, \ldots, P_s$ by defining $d_1, d_2, \ldots, d_s$ to be a set of integers such that $d_i > 0$. The polynomials are then used to generate new sets of direction numbers $V_1, V_2, \ldots, V_s$, one set for each dimension [40].

The interested reader can find more examples of low-discrepancy sequences in [44] and [17]. The source code for an implementation of the Sobol sequence can be found in [7].

2.4 Geometric Brownian Motion

There are numerous ways of modelling assets, ranging in complexity from simpler ones, such as by using a *geometric Brownian motion* [39], to more intricate ones, such as by using jump-diffusion processes [48]. The geometric Brownian motion is defined as

$$dS_t = \mu S_t dt + \sigma S_t dW_t. \tag{2.4}$$

The stochastic differential equation given in Equation (2.4) can be solved by application of Itô's Lemma [57].

First, divide both sides by S_t, like so:

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW_t.$$

Applying Itô's Lemma with with $f(t, x) = \ln x$ yields

$$\begin{aligned} d(\ln S_t) &= (\ln S_t)'\mu S_t dt + (\ln S_t)'\sigma S_t dW_t + \frac{1}{2}(\ln S_t)''\sigma^2 S_t^2 dt \\ &= \mu dt + \sigma dW_t - \frac{1}{2}\sigma^2 dt \\ &= \Big(\mu - \frac{\sigma^2}{2}\Big)dt + \sigma dW_t. \end{aligned}$$

Recall that the previous expression is really just an integral written using shorthand notation. Thus, we can rewrite the expression as

$$\ln S_T - \ln S_t = \Big(\mu - \frac{\sigma^2}{2}\Big)\tau + \sigma W_T,$$

where $\tau = T - t$. Now, by applying the exponential to each side we get

$$S_T = S_t e^{(\mu - \frac{\sigma^2}{2})\tau + \sigma W_T}.$$

The equation is a *continuous stochastic process*, although it can also be discretized, leading to the *discrete stochastic process*:

$$S_T = S_t e^{(\mu - \frac{\sigma^2}{2})\tau + \sigma\sqrt{\tau}Z}, \tag{2.5}$$

where $\tau = T - t$ and $Z \sim \mathcal{N}(0, 1)$.

2.5 LIBOR market model

The *LIBOR Market Model* (LMM) was originally developed by Brace, Gatarek, and Musiela in 1997 [6]. The model is used for pricing interest rate derivatives such as

interest rate swaps, but also more exotic derivatives such as Bermudan swaptions. Unlike other pricing models, such as the Hull-White Model [38] or the Cox-Ingersoll-Ross Model [13], it is not based on instantaneous rates, which cannot be observed in the market. Instead, it is based on LIBOR forward rates [39]. We now give a formal definition of the model.

Consider a set of dates $T_0, T_1, \ldots, T_N$, which are approximately equally spaced. A forward interest rate between dates T_{i-1} and T_i is denoted $L^{(i)}$ where $i = 1, 2, \ldots, N-1$. In the LIBOR market model, each forward rate $L^{(i)}$ is modelled as a stochastic process, and the movements of each forward rate depends on its correlation with the other forward rates.

The dynamics of each forward rate is a martingale under a forward measure $Q^{(k)}$ [43], meaning that forward rates have no drift under this measure. Thus, if we define $dW^{(k)}(t)$, $i \leq k \leq j$, under the forward measure $Q^{(k)}$, the dynamics of $L^{(i)}(t)$ are defined as

$$dL^{(i)}(t) = L^{(i)}(t)\sigma^{(i)}(t)dW^{(i)}(t), \qquad i = 1, 2, \ldots, j. \tag{2.6}$$

In Equation (2.6), $\sigma(t)$ is a deterministic function of time, and $L^{(i)}(0)$ must be given for all $i = 1, 2, \ldots, j$ [43]. It is common practice to define initial values on the forward rates $L^{(i)}(0)$ by looking at current market data. For example, the forward rates could be *bootstrapped* from current par rates [30]. An example of bootstrapping is given in Section 5.1.4.

Note that $dW^{(k)}$ forms a *multi-dimensional stochastic process*. Informally, we can think of each $dW^{(k)}$ as a stochastic process in one dimension, which is correlated with another stochastic process $dW^{(j)}$ moving in another dimension. Specifying all of the correlations is a lot of work, and so to bypass this hurdle one can apply principal component analysis.

Principal component analysis (PCA) is a powerful technique for reducing the dimensionality of data. An excellent description of the technique is given in [58], and some specific applications in finance is described in [41] and [30]. We will devote the end of this chapter to describe the method briefly.

Consider the set of historical forward rates $L^{(i)}(t_k)$ at time t_k. We begin by computing the mean of each forward rate,

$$\mu^{(i)} = \frac{1}{t_N} \sum_{k=1}^{N} \frac{L^{(i)}(t_k) - L^{(i)}(t_{k-1})}{L^{(k)}(t_{k-1})}.$$

where N is the total number of samples.

We can then compute the set of centered annual returns as

$$\Delta^{(i)}(t_k) = \frac{1}{t_{k+1} - t_k} \frac{L^{(i)}(t_{k+1}) - L^{(i)}(t_k)}{L^{(i)}(t_k)}.$$

Next, we compute the covariance matrix C using the centered annual returns. Each entry $C_{i,j}$ in the matrix is given by

$$C_{i,j} = \frac{1}{N} \sum_{k=1}^{N} \Delta^{(i)}(t_k) \Delta^{(j)}(t_k).$$

Finally we can now compute the principal components. By eigendecomposition, $C = U^\top \Lambda U$, where Λ contains the eigenvalues of C, and U a matrix which columns are the corresponding eigenvectors. The eigenvalues and eigenvectors are ordered according to $\lambda_1 > \lambda_2 > \cdots > \lambda_n$.

The i:th principal component is then given by

$$f_i = \sum_{j}^{k} \sqrt{\lambda_i^{-1}} u_k^{(i)} \Delta^{(k)}.$$

Each $u_k^{(i)}$ denotes the k:th element of the i:th eigenvector. It can then be shown that by solving for $\Delta^{(k)}$, we get a new, approximated, expression for the dynamics of the model [30]. The expression is formulated as

$$dL^{(i)}(t) = \sum_{k}^{n} \sqrt{\lambda_i} u_i^{(k)} d\widetilde{W}_k(t),$$

where n denotes the number of principal components, and $\widetilde{W}_k(t) := f_k$.

The number n of principal components can be set freely, but when modelling forward rates, using three or four principal components appears to be a reasonable choice. For example, as for the data used in [30], the first three principal components explained more than 90% of the market volatility.

2.6 Normal-log-normal mixture distribution

A thorough description of the *normal-log-normal* (NLN) mixture distribution is given in [71] and [11], so in this section we will only present it briefly. The benefits

of using an NLN distribution to model asset returns is made clear in [11], where we see that Kolmogorov-Smirnov tests yield better results for the NLN distribution compared to the normal distribution when applied to asset returns.

A normal-log-normally distributed random variable u is given by

$$u = \eta_1 e^{\frac{1}{2}\eta_2}, \tag{2.7}$$

where

$$\begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & \rho\psi \\ \rho\psi & \psi^2 \end{bmatrix} \right).$$

In other words, η_1 and η_2 are two normally distributed variables, with variances 1 and correlation ρ. Its first two moments are

$$\mathbb{E}(u) = \frac{1}{2}\rho\psi e^{\frac{1}{8}\psi^2}$$

and

$$\mathbb{E}(u - \mathbb{E}(u))^2 = \mathrm{Var}(u) = e^{\frac{1}{2}\psi^2}(1 + \rho^2\psi^2(1 - \frac{1}{4}e^{-\frac{1}{4}\psi^2})).$$

Furthermore, both the third and fourth moments exist, but their definitions are quite unwieldy (see [71]). However, for small ρ, we have

$$\mathbb{E}(u - \mathbb{E}(u))^3 \approx \frac{1}{2}\rho\psi e^{\frac{3}{8}\psi^2}(9 - 3e^{-\frac{1}{2}\psi^2}),$$

and

$$\mathbb{E}(u - \mathbb{E}(u))^4 \approx 3e^{\psi^2}.$$

Both equations can be found in [71], and specifically the kurtosis equation originates from [11] for $\rho = 0$.

A comparison between the normal-log-normal mixture distribution and the standard normal distribution is depicted in Figure 2.4.

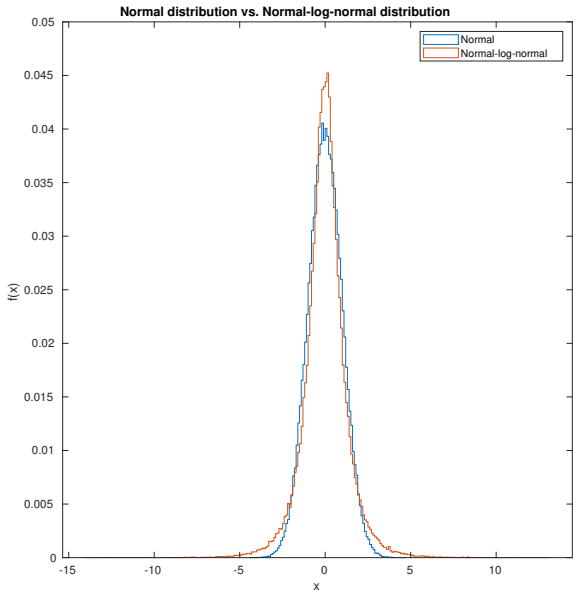


Figure 2.4: A plot showing the approximated probability density function of a normal-log-normal mixture distribution and the standard normal distribution.

As can be seen in [71], a standardized normal-log-normally distributed random variable $v = \frac{u-\mathbb{E}(u)}{\sqrt{\mathrm{Var}(u)}}$ possesses some interesting properties. In particular

- $m(v;\psi,\rho)$ is leptokurtic and symmetric for $\psi > 0$ and $\rho = 0$,
- $m(v;\psi,\rho)$ is skewed to the left for $\psi > 0$ and $\rho < 0$,
- $m(v;\psi,\rho)$ is skewed to the right for $\psi > 0$ and $\rho > 0$.

One inconvenient feature with the NLN distribution is that the analytical form of its density function is unknown, hence it can only be evaluated by numerical means [71].

We can construct normal-log-normally distributed variables with parameters ψ and ρ by using two normally distributed variables $\omega_1 \sim \mathcal{N}(0,1)$ and $\omega_2 \sim \mathcal{N}(0,1)$, and setting

$$\eta_1 = \omega_1 \tag{2.8a}$$

$$\eta_2 = \psi(\rho\omega_1 + (\sqrt{1-\rho^2})\omega_2) \tag{2.8b}$$

and then inserting η_1 and η_2 into (2.7).

2.7 Simulating correlated normal-log-normal processes

Several methods exist for simulating non-normal correlated random variables in the general case, as can be seen in [29], [33], or [66]. These methods tend to be quite hard to use, or might in some cases yield inaccurate results [53]. Had the variables which we wanted to simulate been normal instead, one could Cholesky decomposition instead, which is much simpler by comparison [32]. Since NLN distributed random variables can be constructed using two normally distributed variables (see Equation (2.8), we can actually make use of the Cholesky decomposition method.

Our approach to simulating correlated NLN distributed variables involves correlating their two underlying normally distributed variables. More specifically, consider the two *normally* distributed variables $\eta_1^{(x)}$ and $\eta_2^{(x)}$, given by Equation (2.8), and the corresponding *normal-log-normally* distributed variable x given by Equation (2.7). Also consider another normal-log-normally distributed variable y. We have

$$x = \eta_1^{(x)} e^{\frac{1}{2}\eta_2^{(x)}},$$
$$y = \eta_1^{(y)} e^{\frac{1}{2}\eta_2^{(y)}}.$$

We now would like to find a correlation $\hat{\rho}_{\eta^{(x)},\eta^{(y)}}$ between the underlying normally distributed variables, given by

$$\begin{bmatrix} \eta_1^{(x)} \\ \eta_2^{(x)} \end{bmatrix} \longleftarrow \hat{\rho}_{\eta^{(x)},\eta^{(y)}} \longrightarrow \begin{bmatrix} \eta_1^{(y)} \\ \eta_2^{(y)} \end{bmatrix}$$

so that the correlation between the corresponding normal-log-normally distributed variables x and y match their historical correlation $\rho_{x,y}$. This estimated correlation $\hat{\rho}_{\eta^{(x)},\eta^{(y)}}$ can be computed numerically by minimizing the following expression:

$$\sqrt{(\hat{\rho}_{\eta^{(x)},\eta^{(y)}} - \rho_{x,y})^2} \tag{2.9}$$

for $\hat{\rho}_{\eta^{(x)},\eta^{(y)}} \in [-1, 1]$. Having estimated the correlations, we can now generate correlated normal-log-normally distributed variables by applying Cholesky decomposition to their underlying normally distributed numbers, as described in [32].

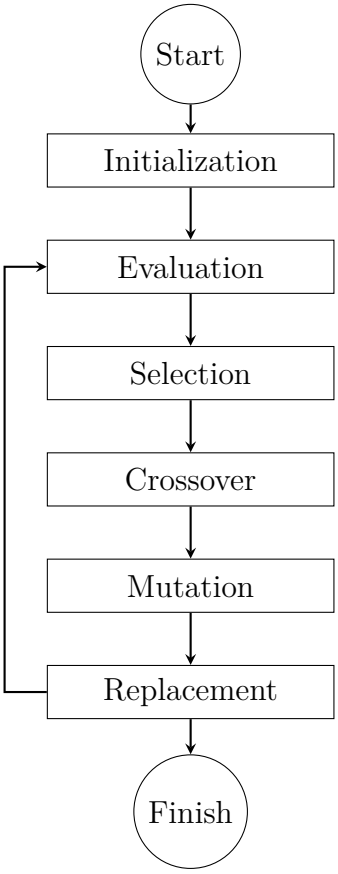


Figure 2.5: An example structure of a simple genetic algorithm.

2.8 Genetic algorithms

The class of *genetic algorithms* (GA) are algorithms inspired by the evolution of organisms in biology. Genetic algorithms form a subset of so called *evolutionary algorithms*. Other examples of evolutionary algorithms include *particle swarm algorithms* or *ant colony algorithms* [69]. In this thesis however, we only concern ourselves with genetic algorithms.

The main actor in a genetic algorithm is a *chromosome*, which in practice is just a vector. Each value c_i in a chromosome $\mathbf{c} = [c_1, c_2, \ldots, c_n]$ is called a *gene*, and each gene maps to a value of a decision variable x_i in our optimization problem. Hence, each chromosome is a potential solution to our optimization problem. The idea of a genetic algorithm is to maintain and evolve a pool of chromosomes until a satisfying solution is found [69]. An outline of the flow of a simple genetic algorithm is given in Figure 2.5. The components are described in the next few sections.

Encoding Schemes

We want to transform the values of our chromosomes into values which makes sense for our problem. Specifically, we want to transform the chromosome genes c_i to a decision variable x_i in our optimization problem, but how to do this depends on what *encoding scheme* is employed. For readability, we refer to decoded chromosomes as *individuals*.

The end goal of the genetic algorithm is to produce individuals with a high *fitness*. During the evaluation step, we pass each individual to a fitness function, defined as $f(\mathbf{c}) \longrightarrow F$, where $F \in [0, 1]$ is the fitness score of the individual $\mathbf{c}$. The purpose of the fitness function is to measure the quality of the produced solutions. However, as will be seen in the next section, it is also used to compute the probability of a specific individual being used when producing offspring.

Evolutionary Operators

Following the evaluation of each individual, the main loop of a genetic algorithm usually involves three steps: *selection*, *crossover*, and *mutation*. Sometimes, we also employ so called *elitism* as a final step before entering the next iteration of the loop. These techniques are commonly known as evolutionary operators [34].

The key mechanism of a genetic algorithm is that it produces offspring using the best individuals of the current generation. However, to maintain the stochastic aspect of the optimization, the best individuals are selected only with a fixed probability. This probability depends on the fitness of the individual, as well as the selection method used. A very basic selection method is called *roulette wheel* selection. As the name suggests, the probability of selecting any one individual is directly proportional to its fitness in relation to the total fitness of all individuals [69].

Having selected individuals, we then proceed by mixing individuals together, for example by splitting individuals in two and swapping the halves with another individuals. An extension to the just described *two point* crossover, is *k point* crossover, in which we do not use halves, but rather sequences of some other size.

The crossed individuals are then mutated according to our choice of mutation method. For chromosomes using real value encoding, a frequently employed approach is *creep* mutation [69]. In creep mutation, we slightly adjust the value of a gene, using for example a uniform distribution. Formally, we do this as

$$c_i' \leftarrow c_i - \frac{C_r}{2} + C_r \cdot U, \tag{2.10}$$

where c_i denotes the current value of a gene, c_i' denotes the next value of the gene, C_r is a constant describing the magnitude of the mutation, and U is a uniformly distributed random variable in the range $[0, 1]$.

Finally, some genetic algorithms also include elitism in the main loop. Elitism means that the next generation of individuals always contains at least k identical copies of the individual from the previous generation, which had the highest fitness score. By employing elitism, we can make sure that the best solution is never lost, allowing the algorithm to keep searching near the individual with the highest fitness. Both crossover and mutation can damage perfectly fine solutions, and elitism functions as an insurance against it.

The perceptive reader will note that elitism is missing in Figure 2.5. The simple reason is that elitism is considered to be a part of the replacement step.

Constraint Handling

Constraint handling for genetic algorithms can be divided into four main areas (see [2]):

1. Reducing the search space
2. Repairing infeasible individuals
3. Separating individuals and constraints
4. Penalization

The categories are ordered by preference, meaning that one should use penalization only if no other method is applicable.

An example of a search space reducing technique is the so called *death penalty* method. In this method, we simply remove infeasible individuals and generate new ones, hoping that the new ones will be feasible. If not, we iterate until we end up with a set of feasible individuals [49]. Instead of completely removing faulty individuals, one could also try to repair them, by making small or large adjustments to their genes.

If we cannot reduce the search space, and are unable to repair the individuals, [2] suggests to separate individuals and constraints. In this case we compute both a fitness score and an *unfitness* score. The latter described the degree to which an individual violates the constraints.

Finally then, if none of the previous alternatives are applicable, we might look into penalization. Applying penalization to optimization problems is not unique for the cases where genetic algorithms are employed [69]. Consider the equation

$$\begin{aligned} \max \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0 \\ & h_i(\mathbf{x}) = 0. \end{aligned}$$

By using penalization, the problem can be reformulated as an unconstrained, penalized, problem, we get

$$\max \qquad p(\mathbf{x};\lambda) = f(\mathbf{x}) - \lambda\left(\sum_{i}^{m}(\max(g_i(\mathbf{x}),0))^2 + \sum_{i}^{k}(h_i(\mathbf{x})^2\right)$$

where λ is a parameter controlling the magnitude of the penalty. Setting $\lambda = 0$ yields a fully unconstrained problem, and on the opposite end, setting $\lambda = \infty$ takes us back to the constrained version of the problem.

Finally, we note that there are many other ways of handling constraints in a genetic algorithm. In-depth discussions on this subject can be found in [15], [12], and [10]

3

Related Work

This thesis pertains to three main areas: asset-liability management, multi-stage stochastic programming, and evolutionary algorithms. A lot of research has been done in each of these areas individually, but comparably little in which ideas from all of the mentioned areas are combined.

One paper, which applies genetic algorithms to a portfolio optimization problem in an asset-liability setting is [26]. However, the optimization problem formulated in the paper is a single-stage problem, whereas in this thesis, we formulate a multi-stage problem instead.

As for multi-stage stochastic programming in an asset-liability management setting, [73] contains several examples, some specifically tailored for pension funds which is the main focus area of this thesis. One of the most famous examples of an asset-liability model for pension funds is the *InnoALM model* by Geyer et al. [31], which was originally developed at Innovest for the pension fund of the company Siemens in Austria. Other notable examples include [5] for the pension fund industry in the United Kingdom, [21] for Swiss pension funds, [74] for American pension funds, and lastly [18] for the pension fund industry in Brazil. The differences between each of these papers mostly lie in the way that they formulate their optimization problem, and less attention is spent on the solution procedure. All of the examples use proprietary optimization libraries to solve multi-stage stochastic programss. In fact, one of the benefits with formulating a portfolio optimization problem as a multi-stage stochastic program is that one can then utilize an off-the-shelf solver, and thus spend more time on the modelling part. However, as the main research question of this thesis covers both the modelling and the solving part, we implement our own solver from scratch.

We finally now relate to some papers which describe evolutionary algorithms applied to portfolio optimization problem. Note that none of the of the following papers define their portfolio optimization in an asset-liability setting, and only a few of the papers define their optimization problems as multi-stage stochastic programs.

First, [61] solves the Markowitz optimization problem described in Section 2.1. The problem involves maximizing the return of a portfolio while simultaneously minimizing the risk, and in the paper the problem is solved using an evolutionary algorithm. In [70], a multi-stage portfolio optimization problem is defined, and then solved using

a particle swarm algorithm. In [60], a constrained single-stage portfolio optimization problem is formulated. The paper implements a genetic algorithm to solve the problem. More research papers regarding the subset of evolutionary algorithms denoted as genetic algorithms, applied to portfolio optimization, include [34], [72], [54], [26], and [10].

To summarize, there are several papers which touches on one or two of the main areas mentioned in the beginning of this section. However, to the extent of our knowledge, there are very few, if any, which combine all three of asset-liability management, multi-stage stochastic programming, and genetic algorithms.

4

Theoretical model description

The model presented is mainly based on ideas from [31], [9], and [65]. The general idea is to evaluate a series of purchases and sales of N instruments over T time steps, and the goal is to maximize the expected wealth in the final time step, conditioned on a set of constraints. We use a complete binary scenario tree, meaning that each time step t contains $2^{(t-1)}$ events, except for the last one, which contains $2^{(T-2)}$ events. Thus, the scenario tree models $S = 2^{(T-1)}$ scenarios in total. See Figure 2.1 for an example of a scenario tree.

Parameters

We define the following parameters for the model.

N	is the set of non-derivative instruments.
M	is the set of derivative instruments.
S	is the set of scenarios in our scenario tree.
$r_i^{(s)}(t)$	is the percentage return of asset i, in time step t, under scenario s.
$L^{(s)}(t)$	is the value of the liabilities in time step t, under scenario s.
$p^{(s)}$	is the probability that scenario s occurs ($\sum_s^S p^{(s)} = 1$).
V_{init}	is the initial total wealth.
m_i	is the percentage transaction cost for asset i.
c_i	is the collateral requirement for asset i.
A	is a matrix of allocation constraints.
$a_j^{(l)}$	is the minimum allocation for constraint j.
$a_j^{(u)}$	is the maximum allocation for constraint j.
λ	is the risk aversion parameter.

Decision variables

The decision variables in our system are portfolio weights, defined as below.

$w_i^{(s)}(t)$	is the proportion of the portfolio allocated to asset i, in time step t, under scenario s.
$w_{\text{cash}}^{(s)}(t)$	is the portion of the wealth held in cash, in time step t, under scenario s.
$d_i^{(s)}(t)$	is the amount purchased or sold of asset i, between time step $t-1$ and t, under scenario s.

Auxiliary variables

Apart from the parameters and decision variables, we also define the following auxiliary variables to improve readability.

$B^{(s)}(t)$	is the target shortfall at time t, under scenario s, as given in Equation (4.1j).
$D^{(s)}(t)$	is the total transaction cost paid, in time step t, under scenario s.
$W^{(s)}(t)$	is the total wealth after rebalancing, in time step t, under scenario s.

Complete model

Having defined the parameters, decision variables, and auxiliary variables, a formal definition of the complete model is then given in (4.1).

$$\max \quad Z = \sum_{s \in S} p^{(s)} \left(W^{(s)}(T) - \lambda \sum_{t \in T} B^{(s)}(t) \right) \tag{4.1a}$$

$$\text{s.t.} \quad W^{(s)}(1) = W_{\text{init}} \cdot \sum_{i \in N \cup M} w_i^{(s)}(t) \cdot r_i^{(s)}(t) \tag{4.1b}$$

$$W^{(s)}(t) = (W^{(s)}(t-1) - D^{(s)}(t)) \cdot \sum_{i \in N \cup M} w_i^{(s)}(t) \cdot r_i^{(s)}(t) \tag{4.1c}$$

$$D^{(s)}(t) = W^{(s)}(t-1) \sum_{i \in N \cup M} m_i \mid d_i^{(s)}(t) \mid \tag{4.1d}$$

$$\sum_{i \in N} w^{(s)} = 1 \tag{4.1e}$$

$$w_i^{(s)}(1) = w_{i,\,\text{init}} \tag{4.1f}$$

$$w_i^{(s)}(t) = w_i^{(s)}(t-1) + d_i^{(s)}(t) \tag{4.1g}$$

$$\mathbf{a}^{(l)} \leq A\mathbf{w}^{(s)}(t) \leq \mathbf{a}^{(u)} \tag{4.1h}$$

$$w_{\text{cash}}^{(s)}(t) \leq \sum_{i \in M} w_i^{(s)}(t) \cdot c_i \tag{4.1i}$$

$$B^{(s)}(t) = \max(W^{(s)}(t) - L^{(s)}(t), 0) \tag{4.1j}$$

$$w_i^{(s)}(t) = w_i^{(s')}(t), \text{ for } s \text{ and } s' \text{ on the same path before time step } t \tag{4.1k}$$

We first give the model objective in (4.1a). As in [54] and [31], the objective maximizes expected terminal wealth, while penalizing accumulated target shortfalls.

Constraints (4.1b, 4.1d) describe the relationship between the total portfolio wealth at various time steps. As for derivative instruments, we do not invest in the underlying directly, but rather, a counter party takes the position for us, and the financial security is just a contract between us and the counter party. Hence, constraint (4.1e) maintains that the portfolio wealth can only be used to invest directly into assets. Constraint (4.1f) describes the changes in allocation between time steps. As per Equation (4.1d) we pay a transaction cost for each allocation change.

At any point in time we must make sure that the optimizer does not borrow money to invest in non-derivative instruments. This is done using constraint (4.1e). For derivative instruments however, the optimizer can invest however much it likes, as long as the *margin constraints* in Equation (4.1i) are fulfilled.

Furthermore, our model also supports so called *allocation constraints*, as given in Equation (4.1h). This allows us to set limits on linear combinations of weights in our model. For example, we can enforce that a minimum of 60% of the portfolio always consists of interest rates, or that the optimizer invests at most 40% of the portfolio in equities and real estate.

Lastly, we have a non-anticipativity constraint. In practice, it only means that decisions made at any time step t can only be made using information known at that time step. Of course, looking into the future is not possible.

The percentage return $r_i^{(s)}(t)$ of an instrument i, in time step t, under scenario s, can for example be sampled from historical time series, or be produced by simulation. As will be described in Section 5.1.1. The same holds true for the target in the target shortfall $B^{(s)}(t)$. Our definition of the percentage return of each instrument $r_i^{(s)}(t)$, as well as our definition of the targets in the target shortfall $B^{(s)}(t)$, is described in detail in chapter 5.

5

Implementation

In this chapter we describe our approach to implementing and solving the multi-stage portfolio optimization problem defined in chapter 4. The chapter is divided into three main parts. First, we describe how scenarios are generated in Section 5.1. Then, Section 5.2 contains a description of our genetic algorithm, including how it can be used to solve the portfolio optimization problem. Lastly, we discuss some of the practical aspects of implementing the complete application in Section 5.3.

5.1 Scenario generation

The end goal of the scenario generation step is to construct several scenario trees, as described in Section 2.2.2. Each node in a scenario tree contains a set of price movements of all instruments. In this thesis, we employ the sample average approximation method for constructing scenario trees. By using sample average approximation, we assign an equal probability to each scenario occurring, and so the complete scenario tree can be constructed by generating a large set of individual events, and then connecting the events so that they form a tree. The process is repeated until we have a satisfying number of scenario trees.

We use a binary tree structure for our scenario trees, meaning that each event at a time step t branches into two new events at time step $t+1$. Such a tree is illustrated in Figure 2.1. A complete binary scenario tree contains $2^t - 1$ events where t is the number of time steps. Note that we could have used any other type of structure for our tree, but using a binary tree allows us to traverse the tree using simple vector arithmetic, as described in Section 2.2.2. Refer to [54] or [31] for examples of other, more complex tree structures.

While constructing scenario trees might appear to be an easy task, there is a lot of edge cases which make the whole process quite complex. The following few subsections will outline the complete scenario generation process, starting with an overview of the relationship between risk types, risk factors, and instruments.

5.1.1 Risk types, risk factors, and instruments

Each node in the scenario tree is an event, and for each event we need to know the price movements of the *instruments*. In our model, the price movements of instruments are based on the movements of their underlying *risk factors*, and each risk factor then belongs to a specific *risk type*. A brief description of each of these components is given below.

Risk type
: The mathematical formulation of a stochastic process. For example a geometric Brownian motion or a jump-diffusion process.

Risk factor
: The result of manifesting the parameters of a certain risk type. In other words, when fitting the parameters of a specific risk type to historical data, we get a risk factor.

Instrument
: An arbitrary function of zero or more risk factors.

To make the relationship between the three components even clearer, consider Figure 5.1. In this figure we have depicted each risk type, risk factor, and instrument used in this thesis.

Note specifically the 1 year forward rate risk factor, the three dots below it, and the n year forward rate risk factor. The three dots represent all of the forward rates between the 1 year forward rate and the n year forward rate. Recall that in order to compute the value of a zero coupon bond with a maturity of 5 years, we first need to compute the forward rate for all years up until year 5. As will be described in Section 5.1.7, the liabilities in our model move like a bond with a maturity of 20 years, which means that we need to model forward rates up to year 20.

Note also, that the cash instrument in our model has no dependency to any risk factor. The reason is that the value of cash does not change in our model.

For our rather small and simple set of investable assets, it suffices to model two risk types. The generic risk type is loosely based on the Black-76 Model [3], and is used as a generic, one-size-fits-all type of risk. The forward rate risk type is based on the LIBOR Market Model [6], and is used for modelling forward rates. The two risk types are thoroughly described in Section 5.1.3 and 5.1.4 respectively.

We define a total of 17 risk factors, five of which are based on the generic risk type, and 12 of which are based on the forward rate risk type. In the next section we describe the data used for constructing the risk factors.

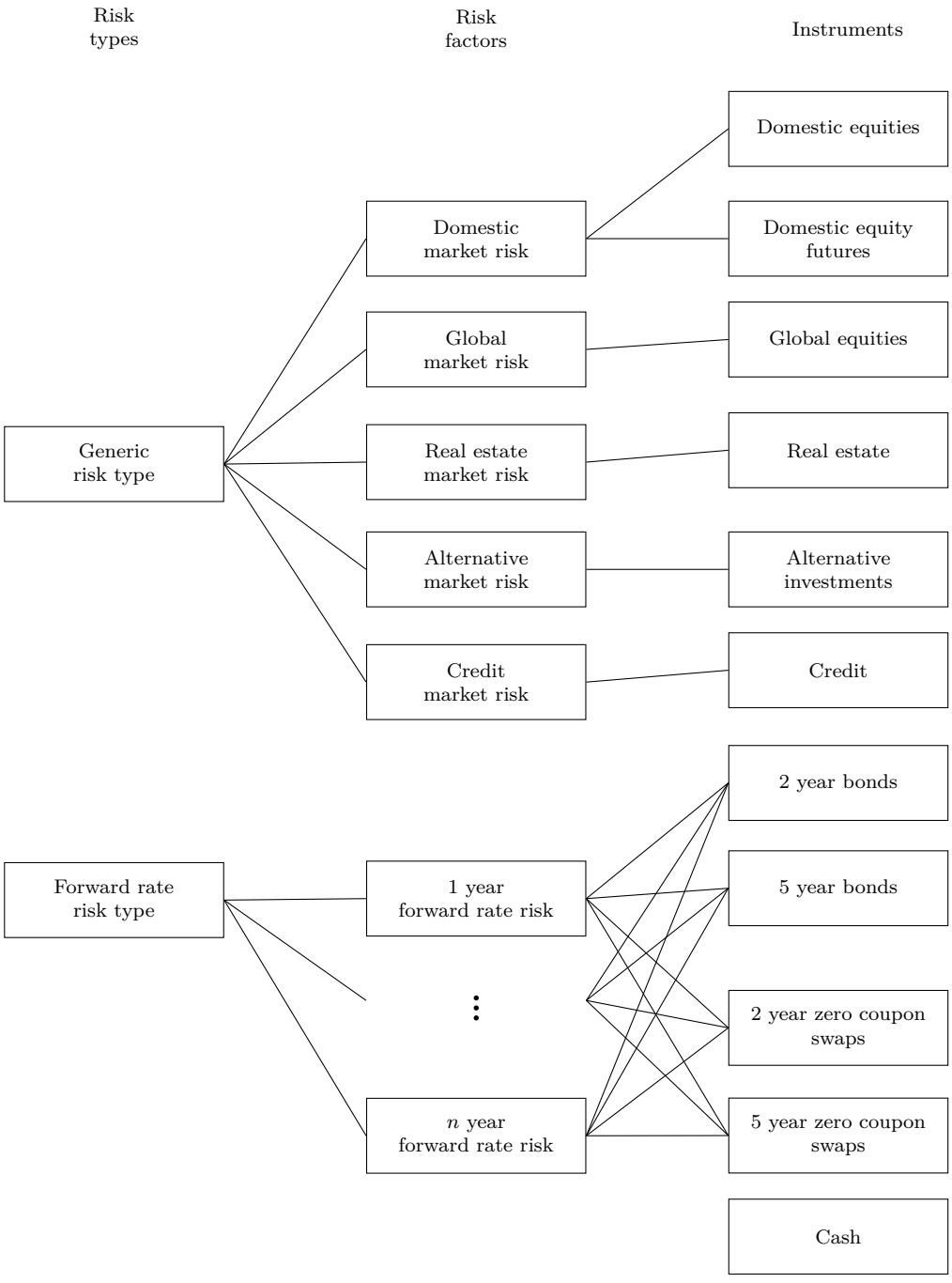


Figure 5.1: A relationship graph describing the relationship between risk types, risk factors, and instruments.

5.1.2 Data acquisition

Risk factors are constructed by fitting a certain risk type to historical data. In this section we give a short description of each time series data used in our model. All time series data range from 2009-01-05 to 2019-03-08.

NASDAQ Swap Fixing
An interest rate swap involves two parties swapping interest rates. One party pays a fixed interest rate, known as the fixed leg, and the other party pays a floating interest rate, known as the floating leg. The NASDAQ Swap Fixing data set contains daily fixings for the fixed rate leg in an interest rate swap. The data is used to construct the forward rate risk factors.

OMXS30 GI
Tracking the 30 most traded stocks on the Stockholm stock exchange, and includes reinvested dividends. The data set contains daily observations of the index values, and is used for constructing the domestic market risk factor.

DB CDX High Yield Excess Return A
This index reflects the excess returns of a credit position with a maturity of 5 years, over a bond with a maturity of 5 years. The data consists of daily observations, and we use it to construct the credit risk factor.

MSCI World Index
A global equity index, covering large and mid-cap equities from 23 countries. The index includes reinvested equity dividends, and the data consists of daily observations of the index value. The data is used for constructing the global market risk factor.

NHX Composite
An equal weighted index tracking the performance of Nordic hedge funds. The data contains monthly observations, and is used to construct the alternative market risk factor.

HOX Sweden
A price index tracking the prices of houses and apartments in Sweden. The data consists of monthly observations, and we use it to construct the real estate market risk factor.

5.1.3 Generic risk type

While the geometric brownian motion is extensively used in financial modelling, for example in the Black-Scholes option pricing model [4], it is primarily useful for demonstrational purposes. The reason is that the shocks in the geometric brownian motion are normal, whereas historically, the shocks in asset returns generally admit to distributions with heavier tails [62]. In this sense, more complex models, such as the jump-diffusion model, might be a more realistic candidate.

For this thesis, we will use a model quite similar to the geometric brownian motion, based on ideas from [3], but modify it to use non-normal shocks. Equation (2.5) gives a discrete formulation of the model, as our main area of application is only concerned with discrete time intervals. Our model is given by

$$S(T) = S_t e^{\mu\tau} e^{-\gamma+\sigma\sqrt{\tau}Y} \tag{5.1}$$

where

μ is the drift of the process,
γ is a convexity correction term,
σ is the volatility of the process,
Y is a random variable following a normal-log-normal mixture distribution,
τ is the difference $T - t$ in time.

Note that unlike a geometric brownian motion, which uses normally distributed shocks, our definition makes use of normal-log-normal shocks instead. This is also the reason as to why we need the convexity correction term γ. The $-\sigma^2/2$ term in the ordinary geometric brownian motion formulation (Equation (2.5)) is a convexity correction term, originating as a result of Itô's lemma[57]. When we exchange the normally distributed random variable Z for the normal-log-normally distributed random variable Y, the convexity correction term has to be adjusted as well. However, since an analytical formulation of the density function of the normal-log-normal distribution does not exist [71], the convexity correction term is computed numerically. The specific details regarding the convexity correction term is given in Section 5.1.5.

Finally, constructing risk factors from the generic risk type is just a matter of estimating the drift μ, the volatility σ, the convexity correction term γ, and the parameters ψ and ρ for the normal-log-normal random numbers, as described in Section 5.1.5.

5.1.4 Forward rate risk type

The forward rate risk type is based on the LIBOR market model [6], with the difference that we use normal-log-normally distributed shocks instead of normally distributed shocks (as in the original LIBOR market model). Note that the primary purpose of the forward rate risk type is to use it when constructing risk factors of forward rates of various maturities.

We will continue this section by giving the mathematical formulation of the forward rate risk type. We will then describe the specific details of how we can construct forward rate risk factors using the forward rate risk type.

The definition of the forward rate risk type is given in Equation (5.2).

$$f_i(T) = f_i(t)e^{\mu\tau}e^{-\gamma+\sqrt{\tau}\sqrt{\lambda_i}\sum_{j=1}^{k} Y_j v_{j,i}}) \tag{5.2}$$

where

μ	is the drift of the process.
γ	is a convexity correction term,
k	is the number of principal components used,
Y_j	is a normal-log-normally distributed random variable
$v_{j,i}$	is the j:th factor loading for the i:th forward rate
λ_i	is the variance of the i:th forward rate
τ	is the difference $T - t$ in time

Having defined the risk type, we now describe how to construct risk factors based on the forward rate risk type. Apart from estimating a few statistical parameters, a process which is described in Section 5.1.5, and which has to be done for the generic risk type as well, there are some additional steps involved when using the forward rate risk type.

Firstly, the initial value $f_i(0)$ of a forward rate risk factor is computed from current par rates using a technique called bootstrapping. The bootstrap method involves looking at par rates, and then using forward substitution to compute discount factors. The discount factors can then be used to derive forward rates.

Moreover, since this thesis is primarily concerned with portfolio optimization for pension funds, we must also take into account some extra precautionary steps added to the bootstrap method by Finansinspektionen (FI). The extra steps added by FI can be found in [28] and [27], and in the section below we will describe how to incorporate these extra steps into our model.

Bootstrapping

We will now describe how the bootstrap method is applied. To improve readability the following few paragraphs will use another notation for the forward rates. Specifically, we denote $f^{(t)}(T_{j-1}, T_j)$ to be the forward rate between maturities T_{j-1} and T_j, at time t.

The first step in the bootstrap method is to collect par rates for all possible maturities $T_1, ..., T_n$. We then, as per the instructions in [28], deduct 0.35 basis points from the par rates to account for credit risk. Let $\widetilde{par}(T_j)$ be the par rate of a treasury note with maturities T_j. We define the adjusted par rate $par(T_j)$ as

$$par(T_j) = \widetilde{par}(T_j) - 0.0035$$

The next step is to recursively compute discount factors from the adjusted par rates. Let $DF(T_j)$ be the discount factor for cash flows at time T_j. We have

$$DF(T_j) = \frac{1 - par(T_j) \sum_{i=1}^{j-1} DF(T_i)}{1 + par(T_j)} \tag{5.3}$$

with $DF(T_0) = 1$.

Now, if we have $T_i - T_{i-1} > 1$ for any $i = 2, 3, ..., j-1$, the recursive formula above does not suffice since we do not know the intermediate discount factors. It is not uncommon that par rates exist for treasury notes with maturities up to 10 years, but the next par rate might be for a note with a 12 year maturity, the one after that for a note with a 15 year maturity, and so on.

To compute the intermediate discount factors we must resort to a numerical method, such as the Newton-Raphson algorithm [19]. We make the assumption that the forward rate is constant between T_{i-1} and T_i. It follows that

$$DF(T_i + k) = DF(T_{i-1})^{\frac{\Delta T - k}{\Delta T}} DF(T_i)^{\frac{k}{\Delta T}} \tag{5.4}$$

where $k = T_{i-1}, T_{i-1} + 1, ..., T_i$, and $\Delta T = T_i - T_{i-1}$.

To find the discount factor $DF(T_j)$ we solve the following expression using the Newton-Raphson algorithm

$$\min \quad \Big(par(j) \sum_{i=1}^{j} DF(T_i) - (1 - DF(T_j))\Big)^2 \tag{5.5}$$

Forward rates can now be computed using

$$\widetilde{f}(T_{j-1}, T_j) = \frac{DF(T_{j-1})}{DF(T_j)} - 1 \tag{5.6}$$

Our computed forward rates must also be shifted in accordance with the *ultimate forward rate* (UFR), currently set to 4.2%. The UFR is gradually incorporated into the forward rates. At $T_{UFR_1} := 10$ the UFR is just barely noticeable, whereas at $T_{UFR_2} := 20$ the forward rate is exactly the UFR.

We define the UFR adjusted forward rate $f(T_{j-1}, T_j)$ as

$$f_{UFR}(T_{j-1}, T_j) = (1 - w(T_j)) \cdot \widetilde{f}(T_{j-1}, T_j) - w(T_j) \cdot UFR,$$

where

$$
\begin{aligned}
w(T_j) &= 0 && \text{for } T_j < T_{UFR_1}, \\
w(T_j) &= \frac{T_j - T_{UFR_1}}{T_{UFR_2} - T_{UFR_1} + 1} && \text{for } T_{UFR_1} < T_j \leq T_{UFR_2}, \\
w(T_j) &= 1 && \text{for } T_{UFR_2} \leq T_j.
\end{aligned}
$$

Lastly, one final adjustment is made to the forward rate. Our assumption on the log-normality of the forward rate risk type prevents the forward rates from ever going below 0. While unusual, negative interest rates do exist in the markets today, and to account for this we simply shift the produced forward rate upwards by a factor of a_{neg}. For this thesis, we set $a_{neg} := 1\%$. The value of a_{neg} can be set freely, and the model would likely work just as well with an even lower value. At last we can compute our final forward rates $f(T_{j-1}, T_j)$ as

$$
f(T_{j-1}, T_j) = f_{UFR}(T_{j-1}, T_j) + 0.01 \tag{5.7}
$$

For an in-depth description of the bootstrap method, see [39].

Principal component analysis

Much like the bootstrapping method as described in this section, we initialize the method of PCA by collecting historical par rates, computing the corresponding discount factors using Equation (5.3). Discount factors can then be used to compute forward rates using equations (5.4), (5.5), and (5.6). We then compute the changes of the forward rates throughout time, and it is these numbers onto which we then apply PCA. The method is thoroughly described in [30] as well as [41].

For our purposes, using $k = 3$ principal components is enough. In fact, it can be shown that the first principal component affects a level shift of the complete curve. The second component drives the slope or tilt of the yield curve, and the third component alters the curvature [30].

5.1.5 Parameters, correlations, and convexity correction

In this section we describe the process of estimating parameters, correlations and the convexity correction term for all risk factors.

Risk type parameters

Each risk factor must be defined in terms of a few parameters, namely its drift μ and the two parameters ψ and ρ, effectively controlling the distributions of the shocks. Furthermore, risk factors based on the generic risk type also uses a parameter σ controlling the magnitude of the volatility. Risk factors based on the forward rates however, employ another method to control the magnitude of the volatility. More specifically, the shocks of risk factors based on the forward rate risk type, are defined using values retrieved from the PCA.

All-in-all, this results in the following steps:

- Estimate drifts μ for all risk factors.
- Estimate parameters for the shocks ψ and ρ for all risk factors.
- Estimate volatilities σ for all risk factors based on the generic type risk.
- Perform PCA to retrieve values for the shocks of all risk factors which are based on the forward rate risk type.
- Estimate correlations between all risk factors.

The estimated drift $\hat{\mu}$ of each process is computed as

$$\hat{\mu} = \frac{\bar{\mu}}{\Delta_t},$$

where

$\bar{\mu}$ is the sample mean logarithmized return,
Δ_t is the annualization factor.

Note that the drift really just represents the annualized logarithmized returns. The logarithmized returns are given by $\bar{\mu} = \sum_i^N \ln(\mu_{i+1}/\mu_i)$. As for the annualization factor, using daily data would for example result in $\Delta_t = 1/253$, assuming there are 253 trading days in a year.

Likewise, estimation of volatilities $\hat{\sigma}$ are done according to

$$\hat{\sigma} = \sqrt{\frac{(N-1)^{-1}\sum_i^N(\mu_i - \bar{\mu})^2}{\Delta_t}}$$

Estimating parameters ψ and ρ for the normal-log-normal variables can be done using a two-sample Kolmogorov-Smirnov test [41]. Essentially, we're guessing values for ψ and ρ, and select the pair which yields the highest p-value. A fuller explanation is given in [20].

Correlations

It is clear that in reality, some risk factors are correlated with others. We now describe how we capture this behavior in our model.

In our case, we have two risk types: the generic risk type and the forward rate risk type. Both risk types employ non-normal shocks, and a method for simulating correlated stochastic processes with normal-log-normal shocks was given in Section 2.7.

For correlations between risk factors based on the generic risk type, it is merely a matter of following the steps outlined in Section 2.7. Correlations between risk factors based on the forward rate risk types on the other hand, have already been implicitly defined. This follows from the fact that their shocks are defined in terms of principal components. What also follows from this fact, is that we cannot define correlations between a generic risk factor and a *single* forward rate risk factor, but can only model the correlation between a generic risk factor and *all* forward rate risk factors.

We now proceed by describing the process of estimating the correlation between a generic risk factor and all forward rate risk factors.

Recall that the forward rate risk type uses three principal components. By definition, the first principal component explains most of the variance. Hence we settle with correlating only the normal-log-normally distributed variable corresponding to the first principal component of the forward rate risk type. The remaining two random variables in the shocks of the forward rate risk type are left uncorrelated.

By way of demonstration, consider two normal-log-normally distributed random variables $Y^{(g)}$ and $Y^{(f)}$. $Y^{(g)}$ corresponds to the shocks of a generic risk factor, and $Y^{(f)}$ corresponds to the first principal component in the shocks of the forward rate risk factors. As described in Section 2.7, in order to simulate correlated normal-log-normally distributed variables, we might instead correlate their underlying normally distributed variables. In our case, we want to estimate a correlation $\hat{\rho}_{\eta^{Y^{(g)}},\eta^{Y^{(f)}}}$, given by

$$\begin{bmatrix} \eta_1^{Y^{(g)}} \\ \eta_2^{Y^{(g)}} \end{bmatrix} \longleftarrow \hat{\rho}_{\eta^{Y^{(g)}},\eta^{Y^{(f)}}} \longrightarrow \begin{bmatrix} \eta_1^{Y^{(f)}} \\ \eta_2^{Y^{(f)}} \end{bmatrix}.$$

However, instead of minimizing the expression given in Equation (2.9), we minimize the following expression:

$$\sqrt{\frac{\sum_{i=1}^{N}(\hat{\rho}_{\eta^{Y^{(g)}},\eta^{Y^{(f)}}} - \rho_{Y^{(g)},Y^{(f_i)}})^2}{N}}$$

where f_i corresponds to the i:th forward rate risk factor, and $N := 12$ is the total number of forward rate risk factors that we use in our model. Hence we minimize over the root mean squared error between the estimated correlation parameter, and the historical correlation parameter for all the forward rates risk factors. We now simulate correlated risk factors, both those based on the generic risk type, and the ones based on the forward rate risk type, by using Cholesky decomposition as described in [32].

Convexity correction

Both the model for the generic risk types, and the model for the forward rate risk types, contain a convexity correction term, denoted as γ in the models. A description as for why it exists is given in Section 5.1.3, but in this section we will describe the process of computing it. A more in-depth explanation is given in [41].

Let $\epsilon = \sigma\sqrt{\tau}Y$ for risk factors based on the generic risk type, and $\epsilon_i = \sqrt{\tau}\sqrt{\lambda_i}\sum_{j=1}^{k} Y_j v_{j,i}$ for forward rate risk factor i, be the shocks of each risk factor. The end purpose of the convexity correction term is to nullify the drift coming form the shocks. Hence we have

$$\gamma = \log(\mathbb{E}[e^{\epsilon}]) \approx \log\Big(\frac{1}{N}\sum_{j}^{N} e^{\epsilon}\Big) \tag{5.8}$$

The last approximate equality in Equation (5.8) comes as a result of the law of large numbers, assuming $N >> 1$.

5.1.6 Generating price movements

Having described all of the prerequisites for the scenario generation, we can now finally generate price movements of instruments. As mentioned in the beginning of Section 5.1, we use sample average approximation to produce scenario trees. With all the mathematical equations in place, the next step is to sample random numbers for our risk factors.

In our case, we use a Sobol sequence as defined in Section 2.3, to sample quasi-uniformly distributed numbers. The quasi-uniform numbers are used as input to the quantile function of a normal distribution, which yields quasi-normally distributed numbers. Finally, the normally distributed numbers are then used to generate random numbers following a normal-log-normal mixture distribution. Some technical details regarding quasi-randomness is given in 2.3.

The quasi-uniformly random numbers are adapted to the normal distribution using the normal quantile function

$$\omega_i = \Phi^{(-1)}(r_i)$$

Having constructed our correlated normal-log-normally distributed numbers, we then generate new generic risk factor values using (5.1), and forward rates using (5.2).

The forward rates are used for computing discount factors, which then can be used to valuate bonds and interest rate swaps. Before we compute the discount factors however, we must reset the adjustment we made in order to allow for negative interest rates to occur (see Equation (5.7)).

Let $\hat{f}^{(t)}(T_{j-1}, T)$ be the value of a generated forward rate between T_{j-1} and T_j in time step t. We then have

$$\begin{aligned} f^{(t)}(T_{j-1}, T_j) &= \hat{f}^{(t)}(T_{j-1}, T) - a_{neg} \\ &= \hat{f}^{(t)}(T_{j-1}, T) - 0.01. \end{aligned}$$

Now, by rearranging (5.6), and using the regulated forward rates $f(T_{j-1}, T_j)$ instead of $\widetilde{f}(T_{j-1}, T_j)$ we can compute the discount factors in time step t

$$DF^{(t)}(T_j) = \frac{DF^{(t)}(T_{j-1})}{f^{(t)}(T_{j-1}, T_j) + 1},$$

with $DF^{(t)}_{T_0} = 1$.

Finally, as per (5.9) we can now derive the zero coupon rates $z^{(t)}(T_j)$. For the sake of completion, the price $P^{(t)}(T_j)$ at time t of a zero coupon bond maturing at T_j is given by (5.10).

$$z^{(t)}(T_j) = \Big(\frac{1}{DF^{(t)}(T_j)}\Big)^{\frac{1}{T_j}} - 1 \tag{5.9}$$

$$P^{(t)}(T_j) = \frac{1}{(1 + z^{(t)}(T_j - t))^{T_j - t}} \tag{5.10}$$

5.1.7 Target wealths

Apart from generating asset price movements, we also generate changes in liability values. During our simulation, we want to make sure that the total wealth of the

portfolio at any point in time is always worth more than the current value of the liabilities. The starting value L_0 of the liabilities is indirectly controlled by the *funding ratio parameter* r_f supplied by the user. The funding ratio describes the relationship between assets and liabilities, and can be used as a metric for the solvency of a company or financial institution.

The starting value of the liabilities is given by

$$L_0 = \frac{W_{\text{init}}}{r_f}.$$

The target wealth movements are then the exact same as the price movements of a 20 year zero coupon bond. That is, they can be computed using Equation (5.10).

5.2 Genetic algorithm

In this section we will discuss our approach for solving the multi-stage portfolio optimization problem using a genetic algorithm. We first give an overview of the various steps of the genetic algorithm. We then proceed by describing how the theoretical model, with all its constraints, is incorporated into our genetic algorithm.

5.2.1 Algorithm overview

Genetic algorithms usually follow a fixed structure, such as the one given in Figure 2.5, and so does ours, albeit with some small tweaks.

We will begin this section by describing the chromosomes used in our algorithm. The section contains quite a few different variables, which are decribed continuously throughout the section.

The chromosomes used in our genetic algorithms employ real-value encoding, and each gene describes how large proportion of the portfolio is invested in a specific instrument, at a specific time step, under a specific scenario.

Assuming our scenario tree consists of n_E events, and we have n_I instruments in our model, the total length of the chromosome will be $n_E \cdot n_I$. For example, assuming $n_I = 2$ and $n_E = 3$, the total length of a chromosome $\mathbf{c}$ would be $2 \cdot 3 = 6$. At c_3, we find the proportion invested in instrument 1, in event 2. In other words, the first n_I genes of the chromosome describe the instrument weights in the first event. The next n_I genes describe instrument weights for the second event, and so on. An example chromosome is depicted in Figure 5.2.

As with most genetic algorithms, we begin by generating an initial set of individuals.

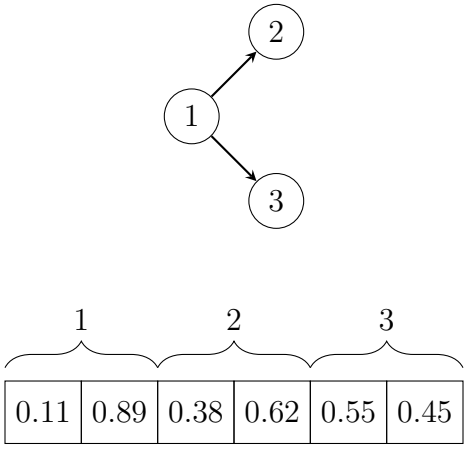


Figure 5.2: An illustration of a real-valued chromosome and a tree describing how each cell in our chromosome maps to a state in the scenario tree. Each gene in the chromosome corresponds to the portion of the portfolio which is invested in a particular instrument. The bracketed numbers above the chromosome show what genes correspond to which event.

Using a ~~plain~~ uniform distribution, we sample $|\,\mathbf{c}\,|$ random values between 0 and 1 for each individual, where $|\,\mathbf{c}\,|$ denotes the number of genes in chromosome $\mathbf{c}$.

Next, we make sure that all non-derivative instrument weights sum to 1 (constraint (4.1e)) by normalizing all indices which map to non-derivative instruments. While our optimization problem is indeed unconstrained by definition, applying penalties for breaking constraints should generally be avoided if possible [15], and by normalizing these specific indices, we reduce the search space instead. Refer to Section 5.2.2 for a further description on how constraints are handled in our model.

Having generated an initial population, we then enter the main loop of the algorithm. In each iteration we compute the fitness of all individuals, as will be described in Section 5.2.2. We then perform roulette wheel selection to select two individuals. We then apply k-point crossover to the selected individuals. In k-point crossover, one usually cuts the chromosome at k arbitrary positions, and then swap the resulting sequences of genes between the two chromosomes. In our case, swapping subsequences like this could potentially result in the new chromosomes violating the constraint that all non-derivative instrument weights sum up to 1 (constraint 4.1e). Thus, we make sure that chromosomes can only be cut so that each subsequence of genes contain a complete set of instrument weights for a single event, by only placing the cuts at n_I sized intervals, where n_I is the number of instruments. An example is given in Figure 5.3.

Following the crossover step, we then mutate individuals using uniform real-value creep. Formally, genes of selected chromosomes are mutated according to (2.10).

Mutation occurs with a probability p_{mut}, and we also truncate each gene of the chromosome so that the allocation constraints (see Equation (4.1h)) are fulfilled.

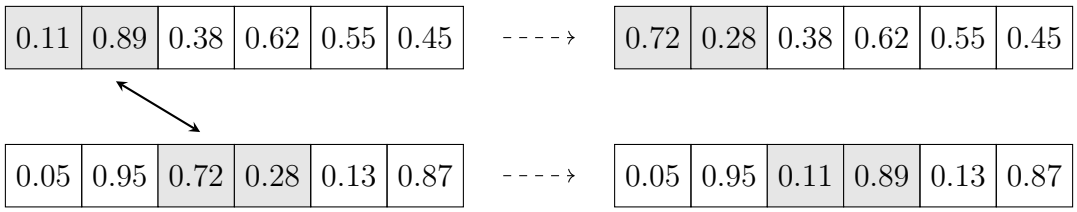


Figure 5.3: The k-point crossover process, in which a complete set of instrument weights are swapped. In this example there are 2 instruments, and 3 states. To the left we is a pair of chromosomes before the crossover, and to the right are the resulting chromosomes after the crossover.

The final step of the main loop of the algorithm involves replacing the old set of individuals with the new one. Our implementation uses generational replacement with elitism. This means that all individuals from the old generation are replaced with the offspring produced by applying evolutionary operators. However, by also incorporating elitism, we keep k_{eli} unmodified copies of the best individual of the old generation. All other individuals of the old generation are removed. In this way, we can guarantee that the best individual seen so far is never lost or destroyed by evolutionary operators.

We iterate in this loop for ~~exactly~~ S generations, after which the algorithm returns with the best individual of all generations.

5.2.2 Objective function

We now describe the objective function, also known as the fitness function, used in the genetic algorithm. As can be seen in Section 4, the optimization problem includes several constraints. Some constraints can be enforced by reducing the search space which, as mentioned in Section 5.2.1, is generally the preferred method. However, some constraints require other approaches, such as applying penalties [54]. Hence, in order to map the model as it is given in (4.1) to a genetic algorithm, we must give special attention to each part of the formulation.

First of all, we note that the constraint that the sum of all instrument weights for an event should be equal to 1 (see Equation (4.1e)), can be satisfied by just normalizing the relevant values in each individual. For each gene c_i pertaining to a specific event, we compute

$$c_i' = \frac{c_i}{\sum_{j \in N} = 1 c_j},$$

where N is the set of non-derivative instrument indices pertaining to this particular event. This is done after initialization as well as after mutation.

Individuals violating allocation constraints (Equation (4.1h)) or margin constraints

(Equation (4.1i)) on the other hand, cannot be repaired. For example, if an individual contains a gene crossing the upper bound of an instrument, ambiguity arises on how the exceeding part should be distributed onto other instruments. It leads to an under determined system of equations, and so we resort to using a penalty function instead.

Note that by defining a penalty function, we effectively separate the constraints into two categories: soft constraints and hard constraints. For example, the constraint that all non-derivative instrument weights should sum up to 1 (Equation (4.1e)) becomes a hard constraint, since we can make sure that no solution violates it by normalizing the values of the chromosome. On the other hand, the margin constraints (Equation (4.1i)) becomes a soft constraint, since the algorithm might produce solutions with a high fitness, despite the added penalty arising from violating the constraint. In practice however, margin constraints are seldom negotiable.

Adding the the penalties to (4.1a) results in a new objective function

$$\max \qquad Z' = Z - \sum_{s \in S} \sum_{t \in T} P(\mathbf{w}^{(s)}(t), \mathbf{a}^{(l)}, \mathbf{a}^{(u)}, A, w_{\text{cash}}^{(s)}(t), \mathbf{c}) \tag{5.11}$$

where

$$\begin{aligned} P(\mathbf{w}^{(s)}(t), \mathbf{a}^{(l)}, \mathbf{a}^{(u)}, A, w_{\text{cash}}^{(s)}(t), \mathbf{c}) = \\ \sum_{i=1}^{n} \sum_{j \in N \cup M} (\max(0, (A_{ij} w_j^{(s)}(t) - a_i^{(u)})^2 + \\ \sum_{i=1}^{n} \sum_{j \in N \cup M} (\max(0, (a_i^{(l)} - A_{ij} w_j^{(s)}(t))^2 + \\ (\max(0, \sum_{j \in M} w_j^{(s)}(t) \cdot c_j - w_{\text{cash}}^{(s)}(t))^2 \end{aligned}$$

and

Z	is the value given by (4.1a),
N	is the set of non-derivative instruments,
M	is the set of derivative instruments,
S	is the set of scenarios in our scenario tree,
c_j	is the collateral requirement for asset j,
A	is a n-by-m matrix of allocation constraints,
$a_i^{(l)}$	is the minimum allocation for constraint i,
$a_i^{(u)}$	is the maximum allocation for constraint i.

Thus in our case we apply quadratic penalty for both the allocation constraints and the margin constraints. Equation (5.11) describes the fitness function used in our genetic algorithm. That is, Z', describes the fitness of an individual.

5.3 Practical implementation aspects

To implement the model as described in sections 5.1 and 5.2, we use several different tools and programming languages. For parameter and correlation estimation we use MATLAB[64]. Both the scenario generation, including bootstrapping of forward rates, and the genetic algorithm, is implemented in raw C++[14]. The C++ code is then compiled to WebAssembly[50], which in turn can be called from a web browser using JavaScript. By compiling the C++ code to WebAssembly, allowing it to be called and run directly in a web browser, we can also develop a simple graphical interface for interacting with our model. Unlike the models developed in [65] and [31], this allows a user without programming knowledge to use our system.

6

Results

As we have now given a formal definition of our model in chapter 4, and a description of our implementation in chapter 5, it is time to evaluate it. We do so by using example settings, collected from a real world use case.

We introduce this section by describing the settings used, and then proceed by showing the results from the parameter and correlation estimation process. We run the model several times in order to produce and approximation of the efficient frontier. Finally, we conclude the section by describing the graphical interface developed for interacting with the model.

6.1 Example application

The constraints, margins, and transaction costs which we use to evaluate our model are based on the investment guidelines of a large Swedish defined benefit pension plan. The allocation constraints can be found in Table 6.1, the collateral margins for derivatives in Table 6.2, and lastly, the transaction costs can be found in Table 6.3.

As described in Section 5, our implementation uses binary scenario trees. We consider each time step in the binary tree to be one year in length, and simulate over a horizon of 3 years in total. As previously described, the optimizer evaluates a set of scenario trees, and for this example application a set consists of 1 000 scenario trees. The initial wealth is 1 unit. The target wealth moves, as mentioned in 5.1.7, as if it was a bond with a maturity of 20 years. We set the initial funding ratio to 1.2, and the target funding ratio to 1.3. Allocation constraints and margin requirements was set as per Table 6.3 and 6.2, respectively.

Parameters for the genetic algorithm is given as follows. The population size is set to 200, ~~which is~~ a good compromise between speed and stability. Note that for example [16] uses a population size of 500, and [34] uses 30. Thus our population size of 200 seems like an acceptable choice. For each generation we employ elitism for 10 individuals. The total number of generations is set to 1 000. The mutation rate is set to 0.02, and the crossover rate is set to 0.05. All the parameters up to this point are fixed throughout the evaluation process. One parameter which does

Instrument	Min (%)	Max (%)
Equities and equity futures)	0	60
Bonds and interest rate swaps	40	100
Real estate	0	15
Alternative investments	0	15
Any single instrument	0	100

Table 6.1: Table of the allocation constraints used for evaluating the optimization model.

Instrument	Margin (%)
Domestic equity futures	3
2-year zero coupon swap	3
5-year zero coupon swap	3

Table 6.2: Table of the sample collateral requirements for investing in derivatives.

not remain fixed throughout the evaluation process is the risk aversion parameter, which will be discussed further down this section.

The application is run on an 8:th generation Intel i7-8550U 1.80GHz processor. Our implementation uses no multithreading capabilities.

6.2 Results from parameter calibration

Parameters for the risk factors were computed as described in Section 5.1.5, using data as described in Section 5.1.2. The results are listed in Table 6.4 and Table 6.5.

To recap, μ is the annualized return, σ is the annualized volatility, and ψ and ρ are parameters for the normal-log-normal mixture distribution which controls the shocks of the risk factors.

Instrument	Transaction cost (%)
Domestic equity	0.1
Domestic equity futures	0.2
Global equity	0.1
Real estate	0.1
Alternative investments	0.1
2-year zero coupon bonds	0.1
5-year zero coupon bonds	0.1
2-year zero coupon swap	0.2
5-year zero coupon swap	0.2

Table 6.3: Table of the sample transaction costs used to evaluate the model.

	Risk factor					Correlations					
		μ	σ	ψ	ρ	1	2	3	4	5	6
1	Domestic market	0.11	0.24	1.00	-0.05	1.00	-0.27	-0.13	-0.15	-0.07	0.03
2	Global market	0.06	0.17	1.90	-0.15		1.00	0.41	-0.04	-0.09	-0.11
3	Real estate	0.06	0.06	1.40	0.10			1.00	-0.03	0.18	-0.34
4	Alternative	0.04	0.03	1.30	0.15				1.00	-0.17	-0.01
5	Credit	0.00	0.02	2.35	0.00					1.00	0.09
6	Forward rates	0.00	-	1.35	0.10						1.00

Table 6.4: Estimated parameters for all risk factors, and the correlation parameters between the risk factors. Note that the forward rate risk factor is missing a σ as it is defined in terms of principal components. The relevant eigenvectors and eigenvalues for the forward rate risk factors are given in Table 6.5.

As can be seen in Table 6.4, both the domestic market risk factor and the global market risk factor have $\rho \leq 0$, thus the distribution of their shocks is skewed to the left. The shocks of the real estate risk, alternative investments risk, and interest rate risks, are skewed to the right. Lastly, we have the shocks of the credit risk factor, which is symmetric in terms of skew, but shows a higher spread than all other risk factors.

We also find that the domestic market risk and global market risk show higher volatility than the other risk factors. Both the global market risk and real estate risk have got an annual return of 6%, but due to its greater volatility, investing in the global market would appear as a less attractive investment opportunity. Note that the forward rate risks do not have a σ parameter, as it is implicitly computed using PCA.

As for the correlations, we find that the global market shows reasonably high correlation to real estate. We also see that real estate and forward rates are inversely correlated. Except for these two cases, we find all risk factors to show relatively low correlation.

6.3 Results from optimization

One of the research objectives of the thesis is to investigate the performance of the complete system, taking multiple aspects into consideration, including the stability of the produced solutions and the computational complexity of the optimization. Furthermore, we also wanted to evaluate the usefulness of the model. To capture all of these aspects, several different experiments were run.

As a preliminary test, we evaluated the convergence of our genetic algorithm. We created a set of 1 000 scenario trees, and then ran the optimizer 10 times on these scenario trees, using the parameters as described in Section 6.2, and the risk aversion λ set to 1.0. Each iteration we stored the fitness of the current globally best individual, and this resulted in the convergence curve shown in Figure 6.1. As can

	1	2	3	4	5	6	7	8	9	10	11	12
λ	0.22	0.08	0.048	0.041	0.025	0.02	0.02	0.02	0.02	0.01	0.01	0.01
	0.18	0.34	0.35	0.39	0.35	0.32	0.26	0.25	0.26	0.25	0.22	0.21
	-0.60	-0.53	-0.25	-0.04	0.09	0.17	0.18	0.24	0.23	0.27	0.18	0.12
	0.49	0.03	-0.35	-0.31	-0.34	-0.12	-0.02	0.09	0.17	0.35	0.30	0.39
	0.56	-0.43	-0.25	0.11	0.19	0.26	0.12	0.18	0.05	0.06	-0.29	-0.44
	-0.19	0.32	0.12	-0.13	-0.36	0.08	-0.23	0.46	0.17	0.38	-0.43	-0.28
	-0.00	-0.19	0.02	0.37	0.07	-0.03	-0.33	-0.39	-0.20	0.55	-0.35	0.32
	0.14	-0.43	0.65	-0.23	-0.14	0.15	-0.25	0.18	-0.31	0.07	0.28	-0.01
	0.01	-0.07	0.32	-0.28	-0.21	0.11	0.45	-0.58	0.38	0.19	-0.13	-0.15
	-0.06	0.15	-0.13	0.10	0.10	-0.07	-0.27	-0.24	0.01	0.35	0.56	-0.61
	0.01	0.05	0.10	-0.35	0.47	-0.55	0.33	0.15	-0.29	0.33	-0.11	-0.06
	0.06	-0.10	0.13	-0.24	0.44	-0.18	-0.51	-0.03	0.62	-0.14	-0.09	0.08
	0.06	-0.25	0.20	0.50	-0.32	-0.64	0.13	0.16	0.27	-0.05	0.06	-0.10

Table 6.5: Eigenvectors and their corresponding eigenvalues (marked with λ) for the forward rate risk factors as a result of the principal component analysis.

be seen in the figure, the chromosomes ~~does~~ not seem improve too much after 1 000 generations.

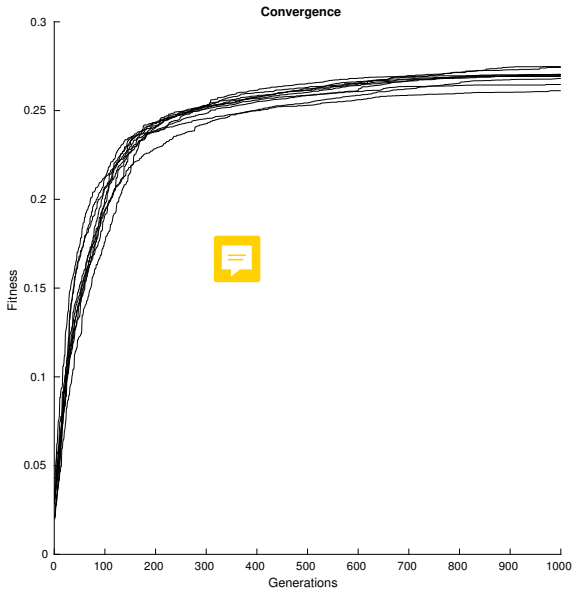
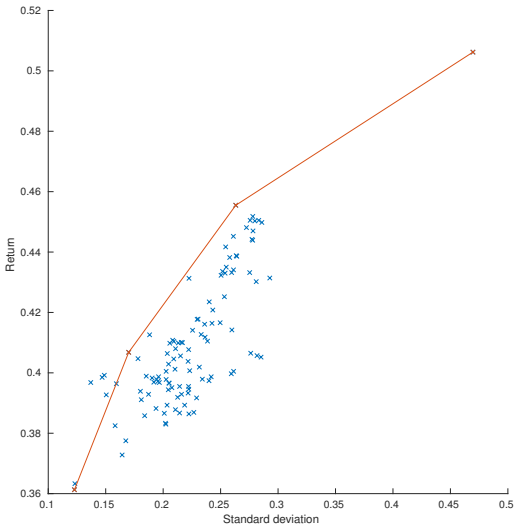
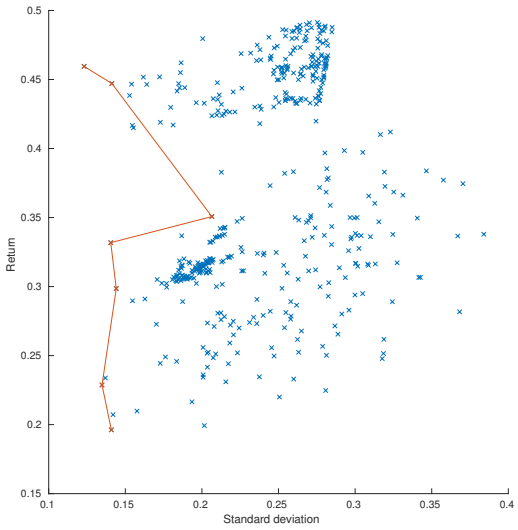


Figure 6.1: Plots showing the convergence curve of the genetic algorithm. Each curve shows a new run of the optimizer using the same set of scenario trees and the same parameters. Fitness is defined as per Equation (5.11).

Having evaluated the convergence rate of the optimizer, we then proceeded by investigating the stability of our model. One way of doing this is by running the optimize several times of a set of scenario trees, and plotting each solution on a two-dimensional space with expected return on the y-axis and expected standard deviation on the x-axis. The leftmost points in this scatter plot form a curve, which is called the *efficient frontier*. A robust model will result in most of the points on



(a) Solutions and the approximated efficient frontier for the first experiment.



(b) Solutions and the approximated efficient frontier for the second experiment.

Figure 6.2: Scatter plots of the solutions produced by running the algorithm several times over two sets of scenario trees. The solutions for the first set is shown in Figure 6.2a, and the solutions for the second set is shown in Figure 6.2b.

the efficient frontier, whereas on the opposite end, an unstable model will have a large spread between each point.

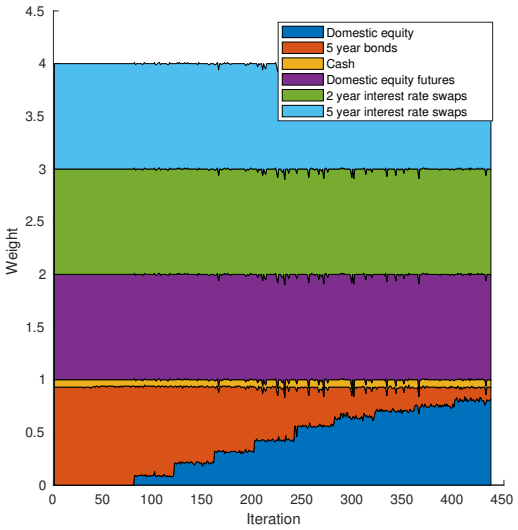
For these purposes we performed two experiments. Both experiments involved restarting the optimizer 200 times, each time producing one new solution. However, each one of the two experiments used its own set of scenario trees. For both experiments we ran the optimizer 240 times. The risk aversion parameter λ was set to 0 in both experiments.

For the first set of scenario trees (Figure 6.2a), we see that the efficient frontier is convex. However, there are many outliers in the produced solutions, and only a few are on or close to the efficient frontier.

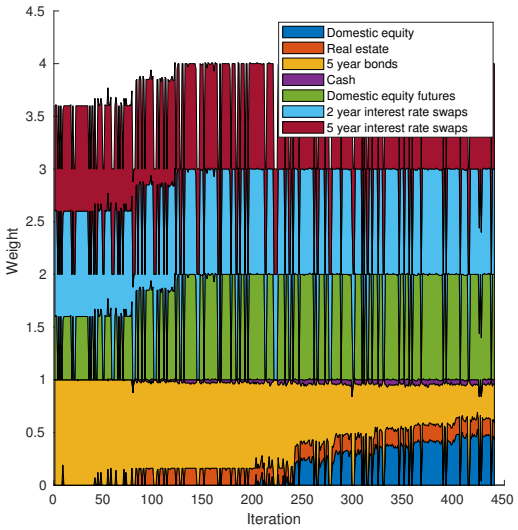
For the second set (Figure 6.2b), we also find that most of the solutions are quite far away from the efficient frontier. We also note that the optimizer seems to jump between two clusters of solutions, which might indicate that it gets stuck in local minima.

Having briefly looked into the stability of our model, we then wanted to evaluate the usefulness of the model, in particular with regards to the risk aversion. Ideally, a pension fund manager should be able to supply his or her own risk aversion as a parameter λ to the optimizer. The optimizer should then return a portfolio with sane instrument weights, matching the risk aversion of the pension fund manager.

Specifically, we ran two experiments for investigating how our model behaves for dif-



(a) Instrument weights from the first experiment.



(b) Instrument weights from the second experiment.

Figure 6.3: Area plots showing the resulting allocations for two experiments involving varying the risk aversion parameter λ. The allocations for the first set is shown in Figure 6.3a, and the allocations for the second set is shown in Figure 6.3b.

ferent values of the risk aversion parameter λ. Apart from using two different sets of scenario trees, both experiments were done exactly equal, using the same parameters as defined in Section 6.2. Each experiment involved running the optimizer 40 times for each risk aversion value λ that we tested, namely $\lambda \in \{0.0, 0.1, 0.2, \ldots, 1.0\}$. This resulted in a total of 440 runs.

Each run, we collected the instrument weights of the optimal solution that the optimizer came up with, and plotted it in an area graph. The results from both experiments can be found in Figure 6.3.

First ~~off~~, note that solutions do not sum up to 1, but rather sum up to 4. The reason is that our universe of investable assets also contains financial derivatives, which are bought using borrowed money. Our allocation constraints prevents us from investing more than 100% of the total portfolio value into any single instrument. This means that if the instrument weights sum up to 4, we invested all of our own money into ordinary financial assets (or cash), and also bought invested in financial derivatives for as much as the allocation constraints allow us to.

For the first experiment, we find that the solutions contain a broader set of financial assets the more we increase the risk aversion parameter λ. The same holds true for the second experiment. Also note that unlike the allocations in the first experiment, the allocations from the second experiment tend to jump between two solutions. In one solution, we make heavy use of domestic equity futures, whereas in another solution we barely have any domestic equity futures at all.

Having evaluated the stability of the solutions, as well as the impacts from varying

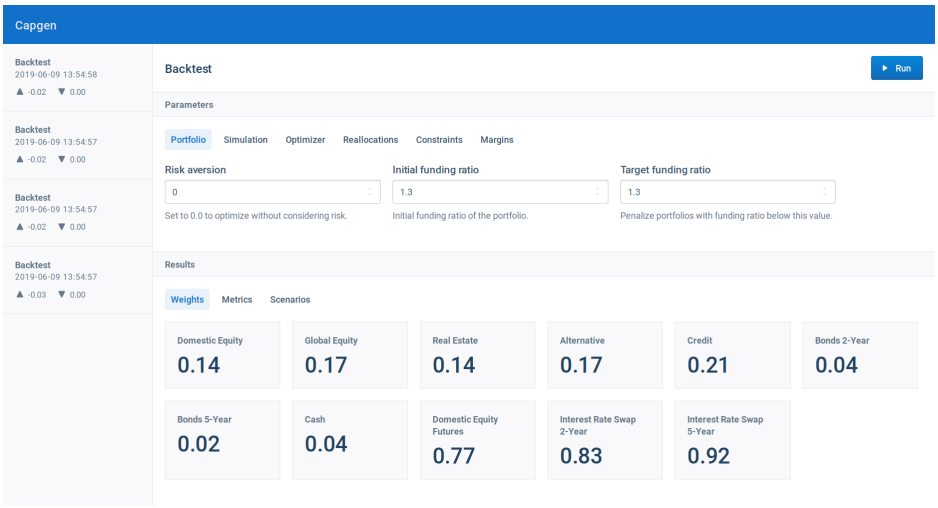


Figure 6.4: A screenshot of the graphical interface used for interacting with our model.

the risk aversion parameter, we also timed the optimization procedure to deduce whether or not the optimization problem is in fact solved in reasonable time. We found that running the optimization on the hardware described in Section 6.1 takes about 1 minute and 21 seconds on average. While this is quite a lot faster than the model described in [31], the results from the paper was produced using a much weaker hardware [31].

6.4 Graphical interface

In order to run the system, one can make use of a graphical interface. The optimization problem and genetic algorithm was implemented using WebAssembly, as described in 5.3. This allows us to interact with the model directly from a web browser. A screenshot of the interface is depicted in Figure 6.4. Note that the plots produced in the previous section are not part of the graphical interface. Moreover, the graphical interface only shows the optimal result after running the optimizer a single time, unlike the figures in the previous section which depict solutions produced from running the optimizer several times.

7

Discussion

7.1 Evaluation of the results

The research question of this thesis is as given in 1.2. The primary research objectives involves defining a multi-stage stochastic optimization model in an asset-liability management setting, and to implement a genetic algorithm to solve the optimization problem in a computationally efficient manner. A theoretical model description of a multi-stage stochastic programming problem was given in Section 4. A description of its implementation, as well as a description of the implementation details of the genetic algorithm was given in Section 5.

Feature-wise, the model supports evaluation of portfolios over several time periods, investing in financial derivatives, and maintaining the funding ratio of the portfolio above a pre-defined goal. These were identified as key features of a portfolio optimizer in an asset-liability management setting, and as such, the model structure is clearly useful for many practitioners in the insurance industry.

Our implementation is able to optimize over several thousand scenarios in just over a minute. Despite there being a lot more room for improvement (see Section 7.2), it does not suffer from computational issues of the magnitude described in [65].

However, while [65] does match our model in terms of functionality, current state-of-the-art models are a lot more complex. For example the Russell-Yasuda-Kasai model includes storing cash in multiple different accounts, allows interacting with the sponsor company of the pension fund, incorporates taxes, and so on [9]. Other examples include [31], which allows for greater flexibility in terms of scenario tree structure, [18] supports withdrawing and depositing money into the fund during the simulation, and [42] which also models the life-expectancy and retirement age of beneficiaries. Our own model supports none of these features.

On the other hand, and as mentioned in Section 3, there are relatively few models which describe both the mathematical formulation of the optimization problem, as well as a bespoke method for solving the problem. In our thesis, we have described both, which in practice should make the process of extending the model a lot more efficient.

Furthermore, by implementing both the optimization model and the solver, we improve the usability aspect of the system. Both [31], [18], and [42], among others, do require at least some familiarity with programming in order to use them, whereas our model comes with a graphical interface. Nonetheless, it should be noted that WebAssembly, described in Section 5.3, was not available at the time any of the previously mentioned models where developed, and so one could argue that such a comparison is one-sided.

Our project involves modelling both assets and liabilities. Both can be thought of as time series models, moving up and down throughout time, depending on a multitude of various factors. Whereas the price of assets might change based on things like company earnings or the production of oil, this is usually not the case for liabilities. Instead, the liabilities of a pension fund for example, depends on current interest rates, payment policies, global salary indices, and more. To prevent the scope of our thesis from growing out of hand, we simply modelled the liabilities as a 20-year bond.

7.2 Potential issues with the model

Naturally, there are several aspects of our model which could be improved. The way in which we model risk factors, using normal-log-normal distributions, might appear as a deficient model of reality. Still, several authors, including [37], [11], [63], and [71] suggest that the distribution is useful for modelling asset prices. But of course modelling asset prices is a huge research area, suggesting that there are many, and perhaps better, alternatives. The model described in [31] supports simulating scenarios using either a normal distribution or a t-distribution, none of which are considered in our work. Another approach, described in [36] and applied in [65], is the moment matching method. The general idea is to not concern oneself with what type of distribution should be used, but rather just try to make sure that the first k moments match the moments of the population.

Further, simulating correlated non-normal variables is a challenging task, and as was described in Section 5.1.6, our approach leads to a slight error in the correlation structure when generating new samples. Another approach to simulating correlated random variables with non-normal marginals is by using the Vale-Maurelli method [66] or the Headrick method [33]. However, as suggested in [53], these methods could potentially lead to biased values on skewness and kurtosis. Furthermore, the Headrick method is computationally expensive, and for our already complex problem, increasing the complexity further is undesirable. It is likely that a better approach would be to use copulas as described in [47].

Another issue is the stability of the model. While it does indeed produce feasible portfolio weights, they do tend to vary substantially between runs. To get around this issue, one might simply run the solver several times, although this is really a symptomatic treatment of the matter, curing the symptoms instead of the cause.

However, curing the problem would require quite a bit of research, as the non-stability might come from multiple sources. In [35] it is suggested that using plain real-value encoding for the chromosomes in our genetic algorithms seldom leads to stable solutions. This was not investigated further in this thesis.

7.3 Ethical considerations

While the optimization model introduced in this thesis does indeed pose an interesting mathematical problem on its own, it is also worth considering the ethical aspects connected with it.

As stated in the introduction of this thesis, strategic investment decisions made behind the closed doors of a pension fund have enormous implications on the lives of its beneficiaries. Even in a worst case scenario, where the performance of a fund deteriorates to a new all time low, the fund must still be able to pay current and future retirees, hence it is of utmost importance that the fund always considers its liabilities when developing its investment policies.

While this section indeed described the usefulness of our model, this is not to say that the model cannot be misused. Indeed, a model and the reality, are two separate units, and only in some cases are we able to develop realistic models of reality. For example, estimating parameters from historical data, might yield fantastic results in theory, but when applying the same practices in reality, might have disastrous consequences if the real world is changing.

Another important point, is that our implementation makes heavy use of numerical computation. While our problem setting is reasonably small, one could still use the model with a much greater universe of investable assets, larger scenario trees, and more complex methods of sampling scenarios. The computational complexity grows exponentially with the number of time steps simulated, and so clearly for very large problems the computational power required is quite substantial. Hence it is not only the time it takes to run such a system which might be problematic, but also the energy consumed by running it.

7.4 Further research topics

One of the benefits with implementing and solving the theoretical model defined in Section 4 using a genetic algorithm is that it is easy to extend. As pointed out in Section 7.2, our model does have its weaknesses, but it is not only these areas that could be explored as future research.

There are several aspects of the genetic algorithm which could be explored. For example, it could be interesting to compare the performance of our implementation

with that of others', such as [70] or [54]. Further, it would be interesting to investigate what parameters, such as population size or mutation rate, are the most optimal for our problem. The genetic algorithm developed in this thesis is quite bland, so it could be tailored even further for our specific optimization problem, which might improve the results.

While the model does solve the optimization problem in reasonable time, it still could be improved even further. Many parts of our model can be parallellised, which would lead to even better computation time.

8

Conclusion

In this thesis we formulated a multi-stage stochastic programming problem in an asset-liability management setting, and solved it using a genetic algorithm. The project is quite broad in scope, which required us to simplify some aspects of the thesis, mostly the parts related to financial modelling.

Our multi-stage stochastic programming formulation optimizes towards expected future wealth, and penalizes the solution depending on the specified risk aversion level. Risk is defined as the expectation of accumulated liability shortfalls throughout the stages. The liabilities were modelled to move as a 20-year bond, and the targets were adjusted according to initial funding ratio settings.

We defined two types of risk types: a generic risk type, and a forward rate risk type. Both types used non-normal shocks to capture the non-normal behaviour of market returns. Our stochastic process models did result in some difficulties concerning how to estimate the correlations between each process. We also used quasi-random numbers to improve the convergence rate of our system.

In order to solve the optimization problem we implemented a genetic algorithm. The algorithm operates on real valued chromosomes, and includes mutation, crossover, and elitism.

The results showed that our implementation indeed provides useful results, and that the optimizer runs in a computationally efficient manner. We also find that solutions have relatively low stability in terms of produced solutions. While the average target shortfalls are quite high, we can still produce the efficient frontier, which might suggest that one can change the parameters of the genetic algorithm to produce even better results.

~~Conclusively,~~ the findings of our thesis perhaps might require some further development in order to be implemented at a pension fund. Nonetheless, our thesis serves as a good ~~foundation~~ for extending or constructing new multi-stage stochastic programming systems in an asset-liability management setting.

Bibliography

[1] Alexandre Adam. *Handbook of Asset and Liability Management: From Models to Optimal Return Strategies*. Wiley, 2008.

[2] John Beasley. Population Heuristics, 2002.

[3] Fischer Black. The pricing of commodity contracts. *Journal of Financial Economics*, 3(1-2):167–179, 1976.

[4] Fischer Black and Myron Scholes. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81(3):637–654, 7 2002.

[5] John Board and Charles Sutcliffe. Joined-Up Pensions Policy in the UK: an Asset-Liability Model for Simultaneously Determining the Asset Allocation and Contribution Rate. In *Handbook of Asset and Liability Management - Set*, volume 2, pages 1029–1067. Elsevier, 2008.

[6] Alan Brace and Marek Musiela. THE MARKET MODEL OF INTEREST RATE DYNAMICS 1. Technical Report 2, University of South Wales, 1997.

[7] Pau Bratley and Bennett Fox. Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator. *ACM Transactions on Mathematical Software,*, 14:88–100, 1988.

[8] Giuseppina Cannas. *A quantitative model for the asset liability management of a Pension Fund*. PhD thesis, Università degli Studi di Cagliari, 2009.

[9] David R. Cariño and William T. Ziemba. Formulation of the Russell-Yasuda Kasai Financial Planning Model. *Operations Research*, 46(4):433–449, 11 2008.

[10] T. J. Chang, N. Meade, J. E. Beasley, and Y. M. Sharaiha. Heuristics for cardinality constrained portfolio optimisation. *Computers and Operations Research*, 2000.

[11] Peter K. Clark. A Subordinated Stochastic Process Model with Finite Variance for Speculative Prices. *Econometrica*, 41(1):135, 7 2006.

[12] Carlos A Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms. Technical report, Instituto Politécnico Nacional, 2000.

[13] John C. Cox, Jonathan E. Ingersoll, and Stephen A. Ross. A Theory of the Term Structure of Interest Rates. *Econometrica*, 53(2):385–407, 1985.

[14] cplusplus.com. C++, 2019.

[15] Bart Craenen, Agoston Eiben, and Elena Marchiori. How to handle constraints with evolutionary algorithms. In *Practical Handbook of Genetic Algorithms*, chapter 10, pages 341–361. Chapmann & Hall/CRC Press, 2001.

[16] Tianxiang Cui, Ruibin Bai, Andrew J. Parkes, Fang He, Rong Qu, and Jingpeng Li. A hybrid genetic algorithm for a two-stage stochastic portfolio optimization with uncertain asset prices. In *2015 IEEE Congress on Evolutionary Computation, CEC 2015 - Proceedings*, pages 2518–2525. Institute of Electrical and Electronics Engineers Inc., 9 2015.

[17] Ishaan L. Dalal, Deian Stefan, and Jared Harwayne-Gidansky. Low discrepancy sequences for monte carlo simulations on reconfigurable platforms. In *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, pages 108–113, 2008.

[18] Alan Delgado de Oliveira, Tiago Pascoal Filomena, Marcelo Scherer Perlin, Miguel Lejeune, and Guilherme Ribeiro de Macedo. A multistage stochastic programming asset-liability management model: an application to the Brazilian pension fund industry. *Optimization and Engineering*, 2017.

[19] Peter Deuflhard. A Short History of Newton's Method, 2012.

[20] D S Dimitrova, V K Kaishev, and S Tan. Computing the Kolmogorov. Technical report, City University of London, 2017.

[21] Gabriel Dondi, Florian Herzog, Lorenz M. Schumann, and Hans Geering. Dynamic Asset and Liability Management for Swiss Pension Funds. In *Handbook of Asset and Liability Management - Set*, volume 2, pages 963–1028. North-Holland/Elsevier, 5 2007.

[22] Jitka Dupacova, Stepan Josef, and J Hurt. *Stochastic Modeling in Economics and Finance*. Kluwer Academic Publishers, 12 2005.

[23] Fredrik Edlund. Målstyrd Förvaltning del 2: Indexering av försäkringstekniska åtaganden ("FTA"), 2017.

[24] European Parliament Council of the European Union. On the taking-up and pursuit of the business of Insurance and Reinsurance (Solvency II), 2009.

[25] Federal Deposit Insurance Corporation (FDIC). *History of the Eighties: Lessons for the Future. Vol. 1, An Examination of the Banking Crises of the 1980s and Early 1990s*. Federal Deposit Insurance Corporation (FDIC), 1 edition, 1997.

[26] João Filipe and Clérigo Ribeiro De Almeida. *Genetic Algorithms Applied to Asset and Liability Management*. PhD thesis, NOVA Information Management School, 2016.

[27] Finansinpektionen. Metod för att bestämma en diskonteringsräntekurva, 2013.

[28] Finansinspektionen. Diskonteringsräntekurvan, 2013.

[29] Allen I Fleishman. A method for simulating non-normal distributions. Technical Report 4, Stevens Institute of Technology, 1978.

[30] Gianluca Fusai and Andrea Roncoroni. *Implementing Models in Quantitative Finance: Methods and Cases*. Springer, 2008.

[31] Alois Geyer and William T. Ziemba. The Innovest Austrian Pension Fund Financial Planning Model InnoALM. *Operations Research*, 2008.

[32] Martin Haugh. Generating Random Variables and Stochastic Processes, 2017.

[33] Todd C. Headrick and Shlomo S. Sawilowsky. Simulating correlated multivariate nonnormal distributions: Extending the fleishman power method. *Psychometrika*, 1999.

[34] Ronald Hochreiter. Evolutionary stochastic portfolio optimization. *Studies in Computational Intelligence*, 2008.

[35] Ronald Hochreiter. Evolutionary optimization for decision making under uncertainty. In *Mendel*, 2011.

[36] Kjetil Høyland, Michal Kaut, and Stein W. Wallace. A heuristic for moment-matching scenario generation. *Computational Optimization and Applications*, 24(2-3):169–185, 2 2003.

[37] David A Hsieh. Modeling Heteroscedasticity in Daily Foreign-Exchange Rates. Technical Report 3, University of Chicago, 1989.

[38] John Hull and Alan White. The General Hull-White Model and Supercalibration. *Financial Analysts Journal*, 57(6):34–43, 2000.

[39] John Hull C. *Options, Futures, and Other Derivatives*. Pearson, 2014.

[40] Jennifer X.F. Jiang and John R. Birge. Comparisons of Alternative Quasi-Monte Carlo Sequences for American Option Pricing. Technical report, Northwestern University, The University of Chicago Graduate School of Business, 2004.

[41] Sara Joon and Josefin Bofeldt. *Pricing of a balance sheet option limited by a minimum solvency boundary*. PhD thesis, Kungliga Tekniska Högskolan, 2019.

[42] Agnieszka Karolina Konicz, Kourosh Marjani Rasmussen, David Pisinger, and Mogens Steffensen. A combined stochastic programming and optimal control approach to personal finance and pensions. Technical report, University of Denmark, University of Copenhagen, 2014.

[43] Andrew Lesniewski. LIBOR market model and its uses, 2008.

[44] Pierre L'Ecuyer and Christiane Lemieux. Recent Advances in Randomized Quasi-Monte Carlo Methods. In *Modeling Uncertainty*, pages 419–474. Kluwer Academic Publishers, 3 2006.

[45] Harry Markowitz. Portfolio Selection. Technical Report 1, The Rand Corporation, 1952.

[46] George Marsaglia and Wai Wan Tsang. The Ziggurat Method for Generating Random Variables. *Journal of Statistical Software*, 5(8), 9 2015.

[47] Alexander McNeil, Frey Rudiger, and Paul Embrechts. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

[48] Robert C Merton. Option pricing when underlying stock returns are discontinuous. Technical report, Massachusetts Institute of Technology, 1976.

[49] Zbigniew Michalewicz. Genetic Algorithms, Numerical Optimization, and Constraints. Technical report, University of North Carolina, 1995.

[50] Mozilla Corporation. WebAssembly, 2019.

[51] Harald Niederreiter and W Schmidt. Low-Discrepancy and Low-Dispersion Sequences. Technical report, Austrian Academy of Sciences, 1988.

[52] Erika Nyström and Viktoria Wirell. *Ett generationsneutralt avkastningsmål - Asset Liability Management analys för buffertfonderna i det svenska pensionssystemet*. PhD thesis, Linköpings Universitet, 2016.

[53] Oscar L. Olvera Astivia and Bruno D. Zumbo. A Cautionary Note on the Use of the Vale and Maurelli Method to Generate Multivariate, Nonnormal Data for Simulation Purposes. *Educational and Psychological Measurement*, 2015.

[54] Jonathan Rosmarin. *An Evolutionary Approach to Multistage Portfolio Optimization*. PhD thesis, Imperial College London, 2007.

[55] Alexander Shapiro and Andy Philpott. A Tutorial on Stochastic Programming, 2007.

[56] Takashi Shinzato. Box Muller Method. Technical report, Tokyo Institute of Technology, 2007.

[57] Steven E Shreve. *Stochastic Calculus for Finance II*. Springer Finance, 2004.

[58] Lindsay Smith. A tutorial on Principal Component Analysis, 2002.

[59] I. M. Sobol. On the distribution of points in a cube and theapproximate evaluation of integrals, 1967.

[60] F. Streichert, H. Ulmer, and A. Zell. Evaluating a hybrid encoding and three crossover operators on the constrained portfolio selection problem. Technical report, Centre for Bioinformatics Tubingen, 2004.

[61] Raj Subbu, Piero P Bonissone, Neil Eklund, Srinivas Bollapragada, and Kete Chalermkraivuth. Multiobjective Financial Portfolio Design: A Hybrid Evolutionary Approach, 2005.

[62] Nassim Nicholas Taleb. *The Black Swan: The Impact of the Highly Improbable*. Penguin, 1 2008.

[63] George E. Tauchen and Mark Pitts. The Price Variability-Volume Relationship on Speculative Markets. *Econometrica*, 51(2):485, 7 2006.

[64] The MathWorks Inc. MATLAB, 2019.

[65] Henri Tuovila. *Optimised Strategies for Dynamic Asset Allocation*. PhD thesis, Aalto University, 2016.

[66] C. David Vale and Vincent A. Maurelli. Simulating multivariate nonnormal distributions. *Psychometrika*, 1983.

[67] J. G. van der Corput. Verteilungsfunktionen I. *Akademie van Wetenschappen*, 38:813–821, 1935.

[68] Bram Verweij, Shabbir Ahmed, Anton J. Kleywegt, George Nemhauser, and Alexander Shapiro. The sample average approximation method applied to stochastic routing problems: A computational study. *Computational Optimization and Applications*, 24(2-3):289–333, 2 2003.

[69] Mattias Wahde. *Biologically inspired optimization methods*. WIT Press, 2008.

[70] Xiaojun Wu, Wei Fang, Wenbo Xu, Jun Sun, and Choi-Hong Lai. Solving the multi-stage portfolio optimization problem with a novel particle swarm optimization. *Expert Systems with Applications*, 2010.

[71] Minxian Yang. Normal Log-normal Mixture. Technical report, UNSW Sydney, 2008.

[72] Lean Yu, Shouyang Wang, and Kin Keung Lai. Portfolio Optimization Using Evolutionary Algorithms. In *Reflexing Interfaces*, pages 235–245. IGI Global, 5 2011.

[73] Stavros Zenios and William Ziemba, editors. *Handbook of Asset and Liability Management, Volume 2*. North Holland, 2007.

[74] Zhuojuan Zhang, Koray D. Simsek, William R. Pauling, John M. Mulvey, and Frank J. Fabozzi. OR PRACTICE—Assisting Defined-Benefit Pension Plans. *Operations Research*, 2008.

[75] William T. Ziemba. The Stochastic Programming Approach to Asset, Liability, and Wealth Management. *CFA Digest*, 2006.