Portfolio Optimisation Using Value at Risk

Project Report

by

Vinay Kaura

A project report submitted as partial fulfilment of the requirements for the degree of

Computing (Computational Management) MEng

Imperial College London

Project Supervisor: Prof. Berç Rustem
Second Marker: Panayiotis Parpas

Project Website: http://www.doc.ic.ac.uk/~vk02/project

Abstract

Optimal portfolios are normally computed using the portfolio risk measured in terms of its variance. However, performance risk is a problem if the portfolio does not perform well. This project involves using linear programming techniques to define and handle the "Value-At-Risk" risk metric.

By evaluating historical prices to create future scenarios one can determine the "Value-At-Risk" of a specified portfolio. Using linear programming software to develop a returns model for the FTSE 100 one can, hence, calculate which stocks should be bought or sold in order to minimise the "Value-At-Risk" of a portfolio with an underlying required returns constraint. The developed tool will look at multi-period scenarios and seek to optimise the portfolio accordingly.

This report documents the analysis of current ways of measuring single period "Value-At-Risk" and the formulation of a unique method of calculating multi-period "Value-At-Risk". It goes on to describe an application which implements this model and highlights the results of exhaustive testing of the application.

Ultimately, using back testing, this report demonstrates how the developed model would have, hypothetically, been able to make profits of up to 40% over the course of the past year while the FTSE 100 benchmark rose by only 27%.

Great stocks are extremely hard to find. If they weren't, then everyone would own them. - Philip A. Fisher

Acknowledgements

Throughout the duration of this project Professor Berç Rustem has been a brilliant project supervisor and his vast array of knowledge has been a great help in times of need. I am also very grateful for the advice given by Panayiotis Parpas during my project.

For the assistance given in setting up of batch jobs and discussing technical implementation techniques I would like to thank Duncan White from CSG.

I would also like to thank Rezwan Ud-Din, from Goldman Sachs, who took time out of his busy schedule to test my final application.

Table of Contents

Chapter 1

Introduction

1.1 Introduction & Motivation

Before looking at "Value at Risk" (VaR), we need to firstly define what risk is and, secondly, why we require a method to measure it. With regards to this project, risk is a measure of how volatile an asset's returns are. Exposure to this volatility can lead to a loss in ones investments. For this reason tools are used not only to passively measure and report risk, but also to defensively control or actively manage it.

As stated in [6] and [7], there have been several large publicised losses in the 1990s, all of which have highlighted the need for accurate risk measure and control. These have included:

- In February 1993, Japan's Showa Shell Sekiyu oil company losing $1.58Billion from speculating on exchange rates.
- In December 1993, MG Refining and Marketing reporting a loss of $1.3Billion from failed hedging of long-dated oil supply commitments.
- In December 1994, California's Orange County announcing losses from repos and other transactions totalling $1.8Billion.
- In February 1995, Nick Leeson, a trader from Britain's Barings PLC, losing $1.33Billion from unauthorised Nikkei futures trading.

After such events regulators have sought to defensively control the risk institutions take. They aim to find a balance between setting out a comprehensive set of regulations to identify institutions who take on excessive risk, while not introducing too rigid a system.

VaR is a popular method which regulators use to assess risk. For example, the Basle Committee[1] has sanctioned institutions to use internal VaR models for capital requirements[2].

VaR is also used by institutions themselves. It is commonly used for self-regulation in the following manner:

- Benchmark Measure – To provide a company-wide yardstick to compare risks across different markets.
- Potential Loss Measure – To give a broad idea of the worst loss an institution can incur.
- Equity Capital – To set a capital cushion for the institution.

When managed properly, VaR can provide a controlled way of getting high returns on ones investments. In March 2006, David Viniar, Chief Financial Officer from the investment bank Goldman Sachs, stated that the firm's VaR rose from $80million in the previous quarter to $92million (+15%), which allowed the firm to increase its net income by 64%.

---

[1] Committee set up by the Bank of International Settlements and based in Basle. It drew up international capital adequacy standards for banks. As stated on http://www.finance-glossary.com.

[2] Rules to identify how much risk the bank is exposed to and then make sure if it loses money then it has enough in reserves to cover these losses. As stated in [18].

However, rather than just using VaR to measure the risk of institutions as a whole, it is also used by institutions in portfolio optimisation techniques to actively manage their risk. This is the focus of my project.

Modern Portfolio Theory models the return of an asset as a random variable and a portfolio as a weighted combination of these assets[1]. This implies that the return of a portfolio is thus also a random variable and consequently has an expected value and a variance. Risk in this model is normally identified with the variance of portfolio return. However, for the purposes of my project, VaR will be used to define the risk of portfolios.

The main problem with variance is that it does not take into consideration the direction of an investment's movement. An asset could be volatile because its price rises quickly, however investors are not distressed by gains!

For investors, risk is about the odds of losing money, and VaR is based on that common-sense fact. By assuming that investors care about the odds of big losses, VaR can be used to answer the questions, "What is my worst-case scenario?" or "How much could I lose in a really bad month?"

The VaR statistic has three components: a time period, a confidence level and a loss amount (or loss percentage). It can thus be used to answer question such as:

- What is the most I can (with a 95% or 99% level of confidence) expect to lose in pounds over the next month?
- What is the maximum percentage I can (with 95% or 99% confidence) expect to lose over the next year?

Bearing this risk measure in mind, I aim to develop an application that will recommend rational investment decisions. Rationality is modelled by supposing that an investor choosing between several portfolios with identical expected returns, will prefer that portfolio which minimizes risk.

By discretising the future into scenarios comprising of multiple time periods, investment decisions will need to be made at the beginning of each period. Using Linear Programming techniques, an objective function and set of constraints will be generated to represent the investment problem. Making use of optimisation software, the optimal parameters for the decisions will then be determined.

This is a risky project since there does not exist at present a similar multi-period portfolio optimisation model which takes into consideration VaR. Combining the representation of a multi-period portfolio with the calculation of VaR, this report defines a model which can optimise a portfolio to hopefully provide lucrative returns.

It's not whether you're right or wrong that's important, but how much money you make when you're right and how much you lose when you're wrong. - George Soros

[1] As stated on http://www.investopedia.com/articles/04/092904.asp

1.2 Project Objectives

The project has the following key objectives:

- Analyse the current methods of calculating single period VaR and formulate a new multi-period optimisation model
- Implement the developed model as an easy to use application
- Implement an automated back-testing facility to verify the accuracy of the developed model
- Demonstrate how the application could be incorporated into a professional Portfolio Manager's daily workflow

1.3 Key Contributions

This report is divided up into the following key chapters:

Background Research
The background research introduces various methods in which VaR is currently calculated and how it can be applied to portfolio optimisations. It also explains that VaR has adverse properties which makes it difficult to use in portfolio optimisations. "Conditional Value-At-Risk" (CVaR) is introduced as an alternative method of calculating VaR. We then go onto discuss the multi-period portfolio optimisation problem and finally combine the multi-period portfolio representation with the calculation of CVaR to define a new multi-period portfolio optimisation model using CVaR.

System Design
The formulated model will only be useful to a user if it is incorporated in a well designed application. This chapter describes a proposed application which seeks to demonstrate how a professional portfolio manager could use the model to make investment decisions and contains features which would allow the use of the application to fit in the portfolio manager's normal day-to-day work patterns.

System Implementation
This chapter describes how the various modules of the application were implemented, and highlights the problems encountered and how they were rectified.

Testing & Evaluation
The project's testing is split into testing of the model, through back-testing, and testing of the application, through exhaustive and user testing. This chapter describes the tests carried out, and their results and analysis.

Conclusion
This chapter analyses the overall project and comments upon its strengths and weaknesses. It also highlights suggestions for future work which could be carried out to further enhance the developed application and model.

Chapter 2

Background Research

Investing without research is like playing stud poker and never looking at the cards.
Peter Lynch

2.1 Value at Risk and Current Implementations

Formally put, "Value-at-Risk" (VaR) measures the worst expected loss over a given horizon under normal market conditions at a given confidence level, as stated in [12].

To put this into context, one might say that the VaR of their portfolio is £0.5 million at the 95% confidence level with the target horizon set to one week. This means that there is a 5 out of 100 chance that the portfolio will lose over £0.5 million within the target horizon under normal market conditions.

Figure 1, below, visually highlights this. It displays 520 observations of the return of a portfolio with the target horizon set to one week. The VaR of the portfolio is the loss that will not be exceeded in 95% of the cases, ie. the lower 5% of returns (26 largest losses), which is equal to £0.5 million.

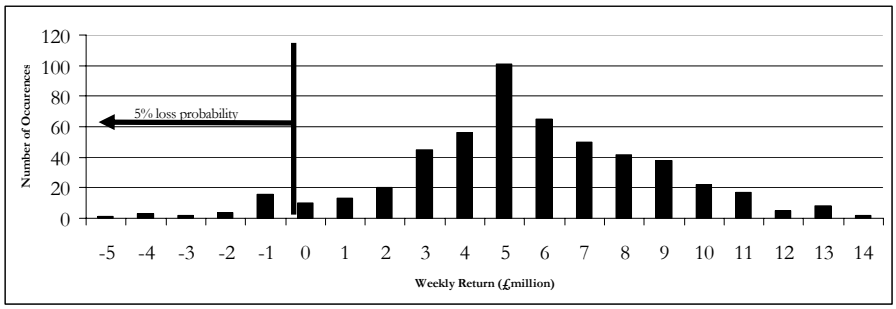


Figure 1

One method of estimating the VaR is to assume that the distribution of returns of the portfolio follows a Normal Distribution. Hence, we can multiply the standard deviation of the returns (3.19) by the 95 percentile of the standard normal distribution (1.645) and subtract this from the mean of the returns (5.55). In this example, this method would estimate the VaR of the portfolio to be £0.3 million.

VaR is a popular risk measure as it provides users with a summary measure of market risk using the same units as the portfolio's bottom line; eg. GBP, USD or EUR depending upon the base currency of the portfolio. This means that VaR can also be communicated to a relatively non-technical audience, such as managers or shareholders.

VaR is also favourable as it looks at downside risk. Hence, unlike the variance of a portfolio, it is not impacted by high returns.

In order to analyse VaR further, we can mathematically define it as:

$$VaR = \zeta_\alpha(\xi) = \inf\{\zeta \mid P(\xi \le \zeta) \ge \alpha\} \quad (1)$$

where α is the confidence level and ξ is a random variable

As mentioned earlier, we could assume the rate of returns of a portfolio could follow a normal distribution, in which case the VaR of the portfolio would be[1]:

$$VaR = \zeta_\alpha(\xi) = \Phi^{-1}(\alpha) = \mu + k(\alpha)\sigma \quad (2)$$

s.t.

$$k(\alpha) = \sqrt{2}erf^{-1}(2\alpha - 1) \quad (3)$$

$$erf(z) = \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2} dt \quad (4)$$

where ξ is a randomly distributed variable with mean μ and standard deviation σ

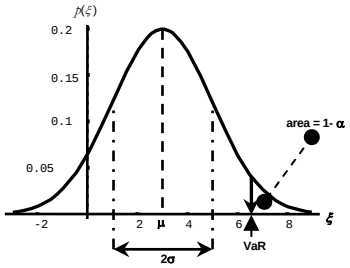


Figure 2

The problem with this assumption is that the normal distribution curve assumes each event is completely random. However, in reality, when a share price falls, people sell. This extra surge of selling will push the share price into the extremes a lot quicker than the curve suggests. A similar thing will happen when a share price rises. To overcome this problem, rather than using a parametric approach, implementations can use an empirical approach to calculate a portfolio's current risk.

As stated in [3], three common ways of calculating VaR are:

- Variance covariance
- Historical
- Stochastic or Monte Carlo simulation

The variance covariance method uses information on the volatility and correlation of stocks to compute the VaR of a portfolio. The volatility of an asset is a measure of the degree to which price fluctuations have occurred in the past and hence expected to occur in the future. Correlation is a measure of the degree to which the price of one asset is related to the price of another.

This method of calculating VaR has become popular after the risk management group at the investment bank J.P. Morgan released an open source system called RiskMetrics in 1994. RiskMetrics aimed to provide an industry wide standard of measuring financial risk. It began by releasing details of the model used at J.P. Morgan, and publishing the volatility and

[1] Taken from [11]

correlation information for stocks listed on the major markets in the world. As demands for enhancements and advice grew, RiskMetrics was spun off from J.P. Morgan in 1998. Since then RiskMetrics has produced a concrete implementation of its model and expanded its data set by providing information on foreign exchange, equity, fixed income, and commodities in 33 countries.[1]

Steps to calculate VaR using the variance covariance method as employed by RiskMetrics:

Step 1 – Specify the confidence level α and construct the volatility matrix V by multiplying a diagonal matrix of the standard deviations of the returns of the assets in your portfolio (available from RiskMetrics) by the confidence interval level η of a normal distribution.

$$\eta = \Phi^{-1}(\alpha) \tag{5}$$

$$V = \eta \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sigma_i \end{bmatrix} \tag{6}$$

Step 2 – Using a column vector ω representing the weighting of each asset in your portfolio and a matrix C representing the correlation of the return each asset (available from RiskMetrics) we can calculate the VaR of the portfolio.

$$VaR = \sqrt{\omega^T VCV\omega} \tag{7}$$

One of the reasons this method has become popular is because it is simple to calculate the VaR figure, as well as being easy to implement. In practice, institutions can hook in their back office systems to the RiskMetrics data source to obtain the volatility and correlation matrices. It, however, does have a few limitations. Since the confidence interval level is obtained by assuming the distribution of returns follows the normal distribution, as mentioned earlier, this may not be realistic.

Due to the volume of data, RiskMetrics only provides volatility information for certain periods (eg. 1 month, 6 months, 1 year) and if an institution wishes to calculate the VaR over a different period then it must use a mapping function on the available data to estimate the volatility. In addition, since all historical information about stocks is summarised as a single volatility value and series of correlation values a vast amount of information is lost.

The historical method was described using Figure 1. To briefly recap, it works by keeping a record of daily profit and loss of the portfolio. The VaR of the portfolio is the loss that will not be exceeded in α% of the cases, ie. the lower $(1-\alpha)\%$ of returns, where α is the confidence level. The benefit of this method is that the historical information is realistic, since, for example, if a major market event occurred in the past, this would be picked up accurately.

[1] As stated on the RiskMetrics website (www.riskmetrics.com)

Taking into account each historical return allows this method to more accurately calculate the VaR of the portfolio. A second advantage is that since the institution is keeping its own historical data it can choose the time horizon to capture the data for, so unlike the variance covariance method, mapping is not required.

The problem with this method is that it will not work if the portfolio composition changes over time. However, a historical simulation approach using the historical asset returns data can be used to overcome this problem. This simulation uses the current portfolio composition to calculate the VaR over each time period covered by the historical data using the historical observations of asset return values. The current VaR of the portfolio is hence the highest VaR of the lowest $(1-\alpha)\%$ of VaRs calculated from historical simulation method. The problem with this is that the historical simulation method is computationally intensive for large portfolios.

The stochastic method works in the same way as the historical simulation method, but instead of using historical rates of returns uses a computer generated series of prices for each asset. More complex manners of generating the prices will provide a more accurate VaR figure, but will obviously take longer to compute. The method has the advantage of allowing users to tailor ideas about future patterns that differ from historical patterns.

Rather than just calculating the VaR of a portfolio, we wish to use the VaR formulation as the objective function and aim to minimise it with respect to a portfolio of stocks. If we take a step back and look at the abstract mathematical definition of VaR we can specify a simple version of the optimisation problem as:

$$\min_{x}\{VaR_{\alpha}[f(x,\xi)]\} \tag{8}$$

$$s.t.$$

$$p^{T}x \geq R \tag{9}$$

where $f(x,\xi)$ *is the loss function for the portfolio,* p *is a vector of the mean predicted future prices,* x *is a vector of the weightings of the assets in the portfolio and* R *is the required return*

As explained in greater detail by Definition 3.3 in [1], VaR has some adverse properties. A disadvantageous property of VaR is that it is non-sub-additive and non-convex. This means that it is possible to construct two portfolios, X and Y, in way such that $VaR(X+Y) > VaR(X) + VaR(Y)$. This is counter-intuitive since portfolio diversification should reduce risk.

Furthermore, as a consequence of its non-convexity VaR has multiple local minima. This is illustrated in an explicit plot of VaR as stated by Uryasev in [11], reproduced in Figure 3. Multiple local minima make the optimisation problem hard to solve since we want to find the global minimum.

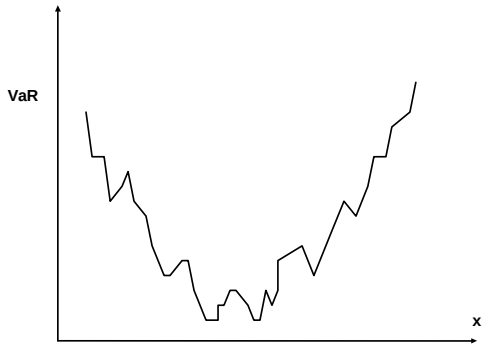


Figure 3

Another problem with VaR is that it does not give any measure of the losses which exceed the VaR. For example, if the distribution of losses has a stretched tail, the VaR figure would not provide any indication of this.

2.2 Conditional Value at Risk

Since VaR, as it has been defined so far, has some adverse properties we need to investigate alternative ways of calculating it. The main problems of the VaR we have defined are:

- it has adverse properties which makes it difficult to optimise a portfolio with a VaR objective function
- it does not give any indication of the losses which exceed the VaR

An alternative measure VaR is "Conditional Value at Risk" (CVaR). By definition, CVaR is the expected loss given that the loss is greater than the VaR at that level. Using formula (1) we can mathematically define CVaR as follows:

$$CVaR = E[\xi \mid \xi \geq \zeta_\alpha(\xi)] = E[\xi \mid \xi \geq VaR] \tag{10}$$

where ξ is a random variable

From this definition, it is clear that $CVaR \geq VaR$. Since CVaR is more conservative than VaR, it is guaranteed not to under-estimate it.

It has also been shown in [9] and [10] that, unlike VaR, CVaR is convex and coherent in the sense it is sub-additive and monotonic. This means that a CVaR function will be easier to optimise in an optimisation problem regarding wealth distribution in a portfolio.

Once again we can assume that the rate of returns follows a normal distribution and hence the CVaR of a portfolio would be[1]:

$$CVaR = \mu + k_1(\alpha)\sigma \tag{11}$$

s.t.

$$k_1(\alpha) = \frac{1}{\sqrt{2\pi}e^{(erf^{-1}(2\alpha-1))^2}(1-\alpha)} \tag{12}$$

[1] Taken from [11]

$$erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \tag{13}$$

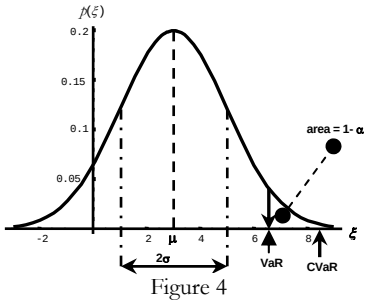


Figure 4

However, rather than use this parametric approach, CVaR can also be calculated using an empirical approach. As with VaR, using either historical or stochastic simulations, future prices can be generated and the CVaR calculated.

From Figure 4 it can be visually seen, that CVaR will lie in the tail of the distribution beyond the value of the VaR of the portfolio. Therefore, an α tail distribution can be used to formulate the CVaR regardless of the distribution of returns.

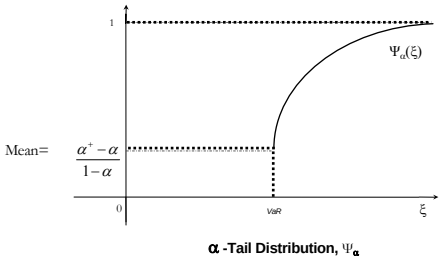


Figure 5

Since VaR is the mean of the losses greater than VaR, in [10] Rockafellar and Uryasev have used the tail distribution as the basis to formulate the following definition of CVaR:

$$CVaR_\alpha(x,\zeta) = \zeta + (1-\alpha)^{-1} \int_{\xi \in \Re^n} [f(x,\xi) - \zeta]^+ p(\xi) \; d\xi \tag{14}$$

where α *is the confidence level,* ζ *is the VaR of the portfolio,* x *is a vector of the weightings of the assets in the portfolio,* ξ *is a random variable,* $p(\xi)$ *is the probability of* ξ *occurring,* $f(x,\xi)$ *is the loss function for the portfolio, and* $z^+ = \max\{z,0\}$

In addition, it has been shown in [10] that the calculation of CVaR can be reduced to an LP problem in the following manner:

- The integral over the continuous distribution can be estimated using scenarios:-
  - $\xi \to \xi^s, \;\; p(\xi) \to p_s, \;\; \sum_{s=1}^{S} p_s = 1, \;\; f(x,\xi) \to f(x,\xi^s)$
  - Making the definition of CVaR:

$$CVaR_\alpha(x,\zeta) = \zeta + (1-\alpha)^{-1} \sum_{s=1}^{S} [f(x,\xi^s) - \zeta]^+ p_s \tag{15}$$

- This definition can now be reduced to LP by expanding out $[f(x,\xi^s) - \zeta]^+$:-

- $[f(x,\xi^s)-\zeta]^+ \rightarrow z_s \geq f(x,\xi^s)-\zeta; z_s \geq 0; s=1,..,S$

It is clear that this linear formulation is guaranteed to implement the z^+ function when, for all s, z_s is minimised. The meaning of CVaR is also made clearer with this formulation. As can be seen, z_s will be zero, in all cases when the loss of the portfolio is less than the VaR of the portfolio. If the loss is greater than the VaR of the portfolio, z_s will take the value of the difference between the loss and the VaR of the portfolio. Since the distribution of z_s represents the tail distribution of losses exceeding VaR, the mean can be found by taking the weighted sum of z_s and dividing it by $(1-\alpha)$. The CVaR of the portfolio is thus this mean value added to the VaR of the portfolio.

Since $CVaR \geq VaR$, if the weightings of assets is optimised in a way such that for all s, z_s equals zero, $CVaR = VaR$ will hold and the optimal ζ will equal the VaR of the portfolio.

This linear formulation is highly beneficial, as it means that large portfolios can be constructed and optimised with a CVaR constraint in a faster time.

Using this definition of CVaR and the optimisation problem stated in [8] we can set up two similar optimisation problems. Firstly we can seek to minimise loss with the CVaR of the portfolio as a constraint:

$$\min_{x,\zeta} \frac{1}{\sum_{i=1}^{n} q_i x_i^0} \sum_{i=1}^{n} -E[\xi_i]x_i \tag{16}$$

s.t.

$$\zeta + \frac{1}{(1-\alpha)S} \sum_{s=1}^{S} z_s \leq \eta \tag{17}$$

$$z_s \geq \sum_{i=1}^{n} (-\xi_i^s x_i + q_i x_i^0) - \zeta; \;\; z_s \geq 0 \tag{18}$$

$$\sum_{i=1}^{n} q_i x_i^0 = \sum_{i=1}^{n} c_i q_i (\underline{\delta}_i + \overline{\delta}_i) + \sum_{i=1}^{n} q_i x_i \tag{19}$$

$$x_i = x_i^0 - \underline{\delta}_i + \overline{\delta}_i \tag{20}$$

$$E[\xi_i]x_i \leq v_i \sum_{k=1}^{n} E[\xi_k]x_k \tag{21}$$

$$x_i \geq 0 \tag{22}$$

where q_i is the initial price of asset i, x_i^0 is the initial number of shares held in asset i, x_i is the number of shares held in asset i at the end of the time period, ξ_i^s is the scenario dependent price of asset i at the end of the time period, S is the total number of scenarios which exist, c_i is the transaction cost of asset i, α is the confidence level, ζ is the VaR of the portfolio, η is the upper limit for the CVaR of the portfolio, v_i is the maximum fraction of the portfolio, in terms of wealth, asset i can exist within the portfolio and x is a vector of the weightings of the assets in the portfolio

The objective function (16) seeks to minimise the loss incurred by the portfolio. Inequalities (17) and (18) are the CVaR constraints of the portfolio. (17) assumes that the probability of each scenario occurring is equal. Inequalities (19) and (20) ensure that transaction costs, proportional to the value of the shares traded, are taken into consideration when optimising the portfolio. This adds necessary friction to model to ensure that buying or selling assets is not without a small penalty. Inequality (21) is a value constraint to ensure a diversified portfolio. Constraint (22) ensures that short positions are not allowed.

Conversely, we can seek to minimise the CVaR of a portfolio given a returns constraint:

$$\min_{x,\zeta}\left\{\zeta + \frac{1}{(1-\alpha)S}\sum_{s=1}^{S} z_s\right\} \tag{23}$$

s.t.

$$\frac{1}{\sum_{i=1}^{n} q_i x_i^0}\sum_{i=1}^{n} -E[\xi_i]x_i \geq R \tag{24}$$

and constraints (18), (19), (20), (21), (22)

where R *is the required return*

The two minimisation problems which have been stated seek to optimise a portfolio by generating S scenarios for the next time period. By increasing the number of scenarios, one would hope to obtain a more accurate value of VaR. Rather than simply increasing S and generating a greater number of prices for the next time period for assets, we can look to define a multi-period optimisation problem.

2.3 Multi-period Scenario Generation & Portfolio Representation

In order to specify a multi-period optimisation returns model, we need to define a method of generating and future prices and representing a portfolio over multi-period scenarios. Multi-period scenarios will be used to represent the future prices. These scenarios can be represented in a tree structure whereby the next time period of the scenario is based upon all information available at the current time period.

The root node of the scenario tree represents 'today'. The prices it represents are observable. The tree structure seeks to partition the future into discrete scenarios with an associated probability. The nodes further down the tree represent conditional future time periods. The arcs represent realisations of the variable prices of assets. The probability of a future node is the product of the probabilities of the ancestors of the node.

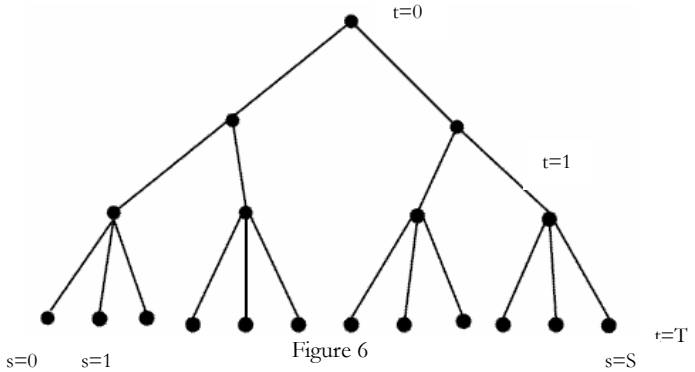


Figure 6

A scenario tree is effectively made up of nodes which contain a cluster of scenarios, one of which is assigned to be the centroid. The final tree consists of the centroids of each node and their branching probabilities.

As stated in [5], the following algorithm can be used to generate a scenario tree:
Step 1 – (*Initialisation*): Create a root node with N scenarios. Initialise all scenarios with the initial prices of the assets. Form a job queue consisting of the root node.
Step 2 – (*Simulation*): Remove a node from the job queue. Simulate one time period of growth in each scenario.
Step 3 – (*Randomised Seeds*): Randomly choose a number of distinct scenarios around which to cluster the rest: one per desired branch in the scenario tree.
Step 4 – (*Clustering*): Group scenario with the seed point to which it is the closest. If the resulting clustering is unacceptable, return to *Step 3*.
Step 5 – (*Centroid Selection*): For each cluster, find the scenario which is the closest to its centre, and designate it as the centroid.
Step 6 – (*Queuing*): Create a child scenario tree for each cluster (with probability proportional to the number of scenarios in the cluster), and install its scenarios and centroid. If the child nodes are not leaves, append to the job queue. Terminate the algorithm if the job queue is empty, otherwise go to *Step 2*.

Imperial College's Department of Computing, has implemented this algorithm as a C++ application called *Cluster*. Given a covariance matrix of historical prices of assets, growth rate of the assets and their current prices, it will output a scenario tree. The application uses sobol sequences[1] to cluster scenarios.

If *Cluster* is to be used to generate scenarios it needs to be provided with the following inputs:

- Exponential growth rate
- Covariance describing the deviation of the assets from the exponential growth curve defined the growth rate above

In order for the exponential growth rate to be calculated from the historical asset prices, the Least Squares method can be used. A general exponential curve can be defined by:

$$y = Ae^{Bx}$$

By taking logarithms of both sides a straight line can be plotted with the equation:

$$\log(y) = \log(A) + Bx$$

Linear regression can then be used to calculate the values of A and B by solving:

$$\begin{aligned} B &= \frac{\operatorname{cov}(x, y)}{\sigma_x^2} \\ a &= \bar{y} - B\bar{x} \\ A &= e^a \end{aligned} \tag{25}$$

[1] A particular type of quasi-random number sequence, as stated on http://www.montegodata.co.uk/Educate/OptionTerms.htm

Given that we have multi-period scenarios, we also need a way of defining a portfolio's weights and transactions spanning several time periods. This has been done in [4] as follows:

$$P + (1-c)\overline{\delta}_0 - (1+c)\underline{\delta}_0 = w_0 \tag{26}$$

$$1'\overline{\delta}_0 - 1'\underline{\delta}_0 = 1 - 1'P \tag{27}$$

$$r_t^s w_{t-1}^s + (1-c)\overline{\delta}_t^s - (1+c)\underline{\delta}_t^s = w_t^s \qquad t = 1,2,..,T \tag{28}$$

$$1'\overline{\delta}_t - 1'\underline{\delta}_t = 0 \qquad t = 1,2,..,T \tag{29}$$

where p_s *is the probability of scenario s occurring,* W_0 *is the initial wealth,* w_t^s *is a vector of asset balances at time t under scenario s,* r_t^s *is a vector of returns of assets under scenario s at time t,* $\underline{\delta}_t^s$ *is a vector of assets sold at time t under scenario s,* $\overline{\delta}_t^s$ *is a vector of assets bought at time t under scenario s,* c *is a vector of transaction costs for each asset,* P *is a vector representing the initial portfolio and* 1 *is the vector* $(1,1,1,...,1)'$

Constraints (26) and (27) enforce the initial capital allocation. Given an initial portfolio of constraints, the model allows money to be added to the portfolio in this initial time period. It also enforces the initial budget of unity, ensuring that the sum of the weights of each asset equals 1. Constraint (28) represents the decision of buying/selling assets at each time period for each scenario. It also incorporates the return of assets in the time period, and the transaction costs of buying and selling shares. Constraint (29) is a balance constraint, ensuring that the value of shares sold equals the value of shares bought in the same period.

2.4 Multi-period Optimisation

Now that a method of generating and representing future prices of assets over several periods has been defined, we can specify a multi-period returns model. Basing it on the method of representing a portfolio over multi-periods as stated in [4], this model will optimise a portfolio when a required returns constraint is submitted by the user. To make the below model easier to read, vector notation has been used.

$$\min_{w,\zeta}\left\{\zeta + \frac{1}{(1-\alpha)}\sum_{s=1}^{S} p_s z_s\right\} \tag{30}$$

s.t.

$$P + (1-c)\overline{\delta}_0 - (1+c)\underline{\delta}_0 = w_0 \tag{31}$$

$$1'\overline{\delta}_0 - 1'\underline{\delta}_0 = 1 - 1'P \tag{32}$$

$$r_t^s w_{t-1}^s + (1-c)\overline{\delta}_t^s - (1+c)\underline{\delta}_t^s = w_t^s \qquad t = 1,2,..,T \tag{33}$$

$$1'\overline{\delta}_t - 1'\underline{\delta}_t = 0 \qquad t = 1,2,..,T \tag{34}$$

$$\sum_{s=1}^{S} p_s w_T^s \geq R \tag{35}$$

$$\frac{\left|w_t^s\right|_\infty}{1'w_t^s} \leq \nu \tag{36}$$

$$z_s \geq W_0 - W_0(1'w_T^s) - \zeta \geq 0 \qquad s = 1,2,..,S \tag{37}$$

$$0 \le \overline{\delta}_t^s \le \overline{\Delta}_t^s \qquad s = 1,2,..,S \ \& \ t = 0,1,..,T \qquad (38)$$

$$0 \le \underline{\delta}_t^s \le \underline{\Delta}_t^s \qquad s = 1,2,..,S \ \& \ t = 0,1,..,T \qquad (39)$$

$$w_t^L \le w_t^s \le w_t^U \qquad s = 1,2,..,S \ \& \ t = 0,1,..,T \qquad (40)$$

where p_s *is the probability of scenario s occurring,* W_0 *is the initial wealth,* w_t^s *is a vector of asset balances at time t under scenario s,* r_t^s *is a vector of returns of assets under scenario s at time t,* $\underline{\delta}_t^s$ *is a vector of assets sold at time t under scenario s,* $\overline{\delta}_t^s$ *is a vector of assets bought at time t under scenario s,* c *is a vector of transaction costs for each asset,* P *is a vector representing the initial portfolio,* v *is a fraction representing the maximum proportion of the portfolio, in terms of wealth, an asset can be held and* 1 *is the vector* $(1,1,1,...,1)'$

Similar to the single period optimisation problems, constraints (30) and (37) seek to minimise the CVaR of the portfolio. However, in the multi-period case, instead of a scenario representing a single time period, a scenario represents several time periods. A limitation of the model I have constructed is that the CVaR of the portfolio is formulated by only looking at the returns of the portfolios in the final time period of each of the scenarios. I made this decision so as to slightly simplify the multi-period model and decrease the number of variables in the model, reducing the time it will take for the model to be optimised.

Constraint (35) ensures that the model produces a return greater than or equal to the minimum required return of the portfolio. Constraint (36) ensures that an asset can not constitute more than defined fraction of the portfolio. Constraints (39), (38) and (40) bound the vectors which represent the portfolio weightings, and the buying and selling of assets. Constraints (31), (32), (33) and (34) are as those stated in Section 2.3.

2.5 Efficient Frontier

Given the model I have developed in the previous section, every possible asset combination can be plotted in a risk-return space. The line along the upper edge of the region which defines the collection of the plotted points is known as the efficient frontier. Combinations along this line represent portfolios for which there is lowest risk for a given level of return. Conversely, for a given amount of risk, the portfolio lying on the efficient frontier represents the combination offering the best possible return. Mathematically the Efficient Frontier is the intersection of the Set of Portfolios with Minimum Risk and the Set of Portfolios with Maximum Return[1].

No portfolios can be constructed corresponding to points in the region above the efficient frontier. Points below the frontier are suboptimal so a rational investor will hold a portfolio only on the frontier. This means that my model should, given a returns constraint, only return points on the frontier. Figure 7 displays an efficient frontier for VaR, as stated in [12]

[1] As stated on http://en.wikipedia.org/wiki/Efficient_frontier

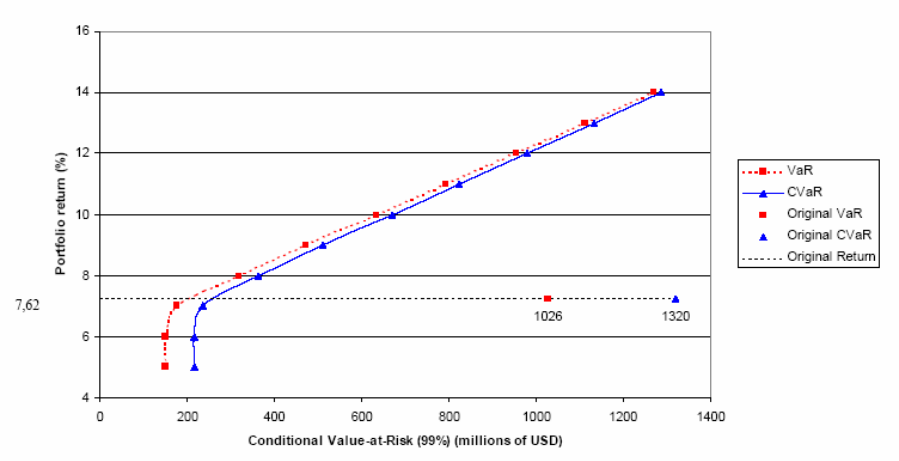

16
14
12
10
8
6
4
7,62
Portfolio return (%)
0
200
400
600
800
1000
1200
1400
Conditional Value-at-Risk (99%) (millions of USD)
1026
1320
VaR
CVaR
Original VaR
Original CVaR
Original Return

Figure 7

Chapter 3

Specification

To give a clear indication of the specification of the various aspects of the system, I will breakdown the specification in the following modular manner:

- Core Requirements
  - Front-End Specification
  - Back-End Specification
- Non-Core Requirements
- Extensions
- Documentation Requirements
- Implementation Requirements

The front-end specification outlines areas of the system which will directly interface with the user. The back-end specification outlines processes of the system which the user does not see.

3.1 Core Requirements

Ultimately, the system's main goal is to optimise a portfolio given a returns constraint and seek to minimise the VaR of the portfolio. In essence, the system should use the inputs given by the user, compute the optimal portfolio and display the results. In order to demonstrate the capability of the software and to aid with benchmark testing, the system will be set up so that it contains only FTSE 100 assets in its domain.

3.1.1 Core Requirements – Front-End Specification

In order for the optimisation to be carried out, the system will need to capture the following inputs from the user, and perform validity checks to ensure the input is of a valid value:

- *initial weightings of assets in the portfolio* – should be entered as either number of shares held of each asset or amount of asset held as a fraction of total initial wealth. If entered as fractions of total wealth, the fractions should sum up to 1. Due to the amount of data, the system should allow the weightings to be entered through the use of textboxes, or the import of a comma separated values (.csv) file. All values should be non-negative.
- *required return* – should be entered as a non-negative percentage figure.
- *confidence level* – should be entered as a non-negative percentage figure.
- *transaction costs* – should be entered as a non-negative percentage value for each asset. The system should only make it compulsory for the user to enter this once, and store the values for all future optimisations. It should, however, be possible for the user to amend any transaction cost at will.
- *number of days in which make up 1 time period* – should be entered as a non-negative integer. Once this has been set for a portfolio, the system should default to this time period, unless the user requests otherwise.
- *number of time periods to consider in optimisation problem* – should be a non-negative integer greater than 1.

- *number of scenarios which should be generated per scenario tree node* – should be a non-negative integer greater than 1

Once these parameters have been entered and verified, the system should seek to optimise the portfolio. Once the optimisation has been completed, the system should inform the user of the optimal distribution of the wealth between the assets it recommends.

Due to the large amount of data being returned, the system should display it in an as easy to read manner as possible. The Graphical User interface should adhere to Nielson's Ten Usability Heuristics, stated in [16], to ensure that it is an effective interface.

The system should also provide the facility for this data to be exported to file in a Microsoft Excel compatible format (for example .xls or .csv). The system should also output the VaR of the portfolio, which should equal the CVaR of the portfolio due to the reasons highlighted in section 2.3.

3.1.2 Core Requirements – Back-End Specification

The main purpose of the back-end of the system is to implement the optimisation problem stated in section 2.5 of this document. Given the inputs stated in 3.1.1 the back-end should return the following:

- optimal portfolio weightings
- optimal portfolio Value-At-Risk

In order to perform the calculation, the back-end needs to firstly generate the future scenarios. As stated in Section 2.3, this is done by analysing the historical data and using stochastic programming to create a scenario tree. The option exists to either incorporate the implementation of a new scenario tree generator into the system, or to use *Cluster*[1]. Either way, historical asset prices of the assets will be required. The system should automatically acquire the up-to-date prices of assets in the system on a daily basis. Since this is a repetitive daily process, there should be no user intervention required to perform the task.

Once all the parameters required to perform the optimisation have been obtained, the back-end should, as efficiently as possible, carry out the optimisation and return the required values.

3.2 Non-Core Requirements

The features described in this section are those which introduce greater functionality to the system but still remain directly related to the optimisation of a portfolio.

Rather than simply provide the user with the optimal asset weightings, VaR and details of which assets need to be bought/sold, the user should also be shown the scenario tree which has been generated and used for the optimisation. This will allow the user to have confidence that the system has generated acceptable future prices of the assets, and optimised the portfolio accordingly.

[1] Application developed by Department of Computing at Imperial College, as described in section 2.3

The system should also provide the facility for an efficient frontier to be generated. This will enable the user to visually see how different requirements of return will affect the amount of risk that needs to be taken.

Steps should also be taken to increase the speed of the optimisation process as much as possible. A possible way of achieving this is by having a separate automatic process to obtain asset prices. This will ensure that the price data within the system is always up-to-date and the system does not need to perform multiple ad-hoc acquisitions of prices. The system should also seek to learn the typical number of days which make up a time period, as entered by the user, and pre-generate the historical returns data for these time periods. This will help to ensure that the data required to generate the scenario tree is not being created at runtime.

3.3 Extensions[1]

As will be explained in greater detail in Section 7.1.2, back-testing will be used to verify the model. One possible extension to the system would be to incorporate back-testing as a feature of the system. This will be highly beneficial as it will provide an in-built way for someone to check the accuracy of the VaR model of the system.

To demonstrate how a tool of this kind can be used by real institutions, I have also specified optional features which will help to show how the system can be made more appealing for a professional portfolio manager seeking to optimise his/her portfolios with VaR requirements. These features aim to facilitate the seamless incorporation of the system into a professional's standard workflow.

- The system should allow the user to view the historical data of assets in its domain.
- The system should allow for the persistent storage and tracking of a user's portfolios. This will allow a user to use the system as a full management tool for his/her portfolios. The system can thus keep track of a portfolio over time, and can effectively record all trades booked by the user at each optimisation instance. The system should be able to provide a graphical method of showing the past performance of the portfolio and facilitate the option of comparing the portfolio to a user selected benchmark.

3.4 Documentation Requirements

In addition to the implemented system, thorough documentation should also be provided. This document should cover the following:

- outline of the operation of the system, including a user manual highlighting how to perform key tasks
- an evaluation of the implemented system, highlighting any weaknesses in the system
- ideas for further enhancements to the system

[1] The requirements stated in this section should be deemed desirable but are not strictly required by the specification

3.5 Implementation Requirements

When considering how to implement the software, care should be taken when choosing the programming language(s) it will be written in. The key high level requirements from a software usability perspective are:

- Easy to use Graphical User Interface
- Fast computation of numerically intensive calculations

3.6 Summary of Requirements

To provide a neat outline of the specification, I can use goal orientated Requirements Engineering methods. The fore mentioned specification is thus summarised in the following KAOS notation diagram:

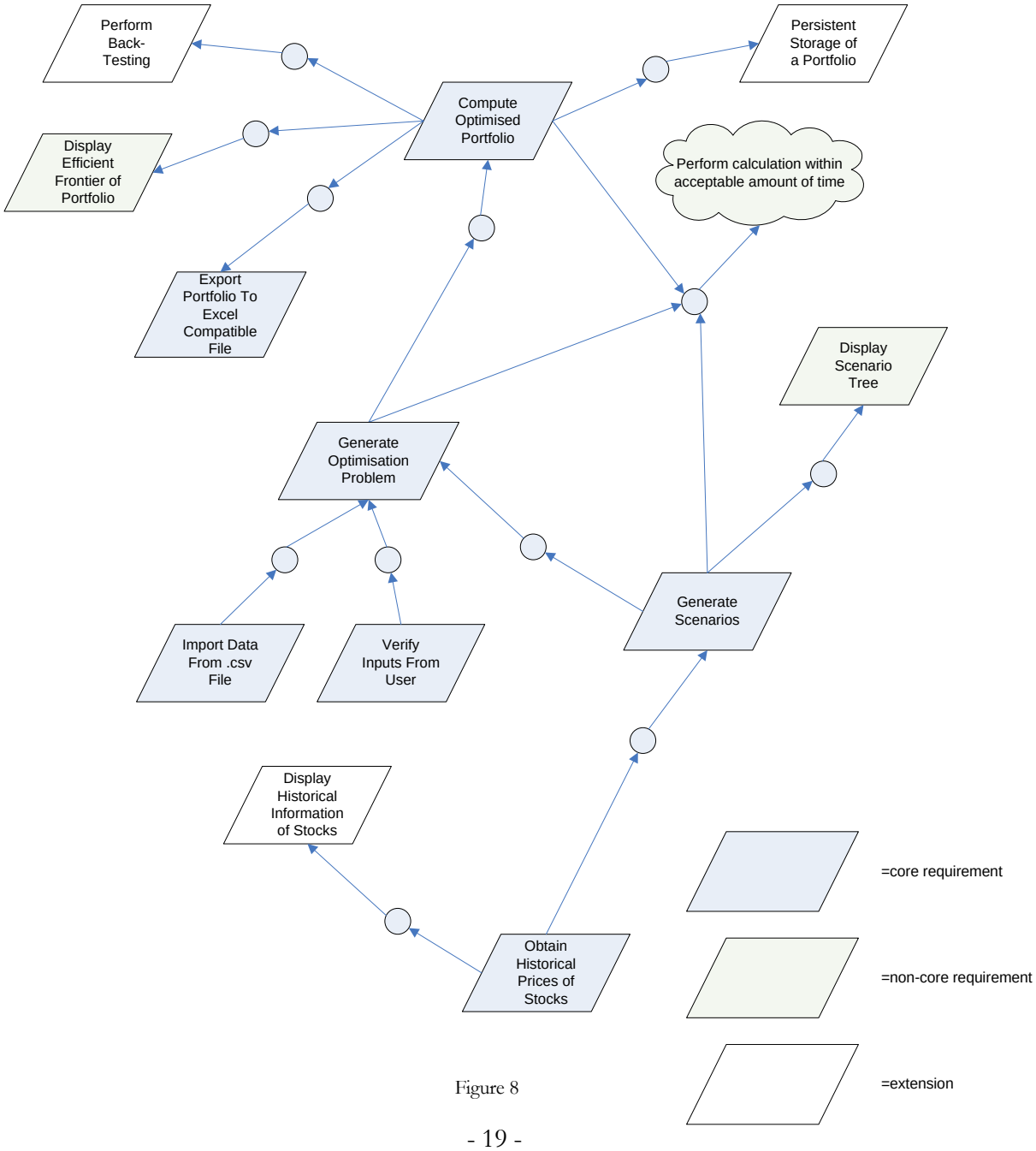


Figure 8

Chapter 4

System Design

If you are either a conservative or a very nervous investor, a mechanical system might be smart. It certainly achieves two goals: (1) limiting potential losses and (2) helping you sleep at night. As for maximizing your returns, I think that's another story. - Jonathan Steinberg

The intended software result of this project will be an application which ultimately implements the multi-stage optimisation problem. However, as stated in the Specification I also intend the application to be designed and have features which demonstrate how it could be used by a professional portfolio manager in his or her day to day work. The final application will thus hopefully demonstrate how a good theoretical model is the foundation of a good application, but without a good application a good theoretical model is useless to an end user.

4.1 System Overview

A good design is the basis of a good final application so before defining exactly how I will be implementing it, I will describe an abstract overview of my application. The figure below highlights the main components of my system and how they will interact.

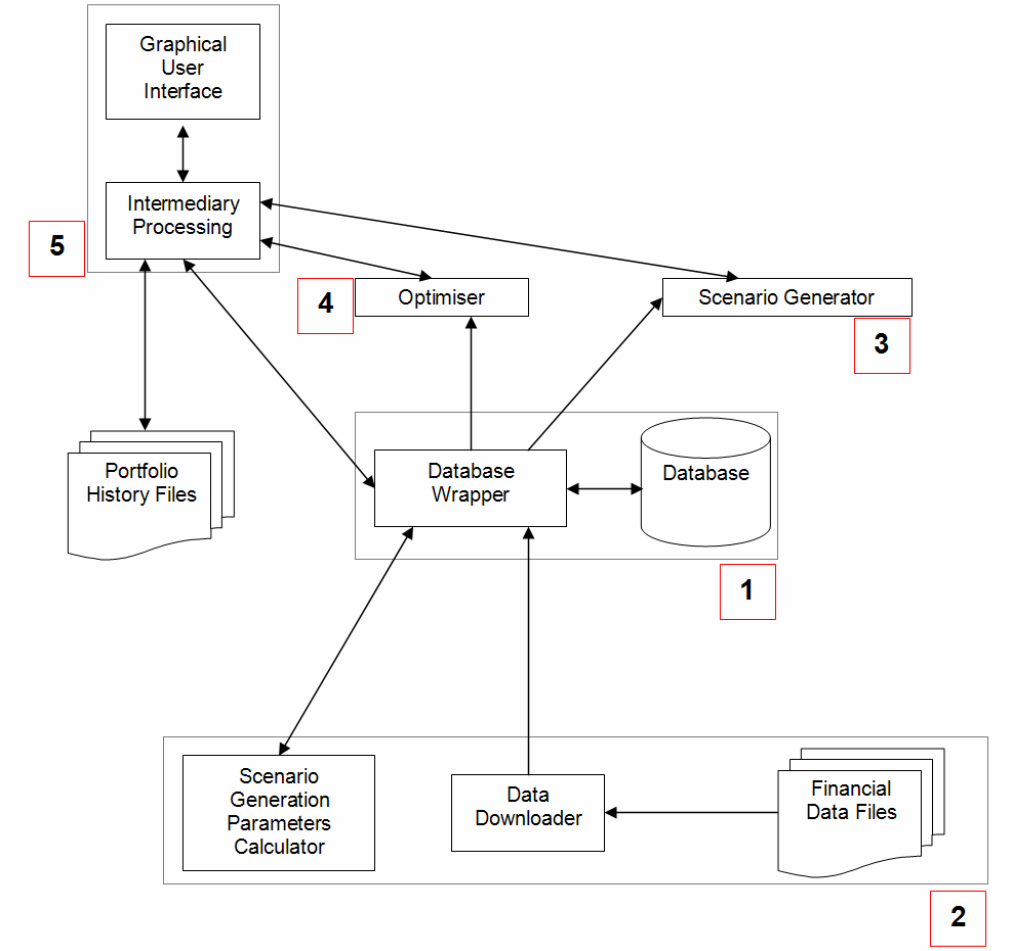


Figure 9

Module 1 – *The Database* – The database forms the core of the application. I will place a wrapper around the actual database, with which other components must interact to gain access to the database. This will ensure that only allowed commands are run on the database and that there is common interface for the other components to use. The database will store the following data:

- Names and symbols of assets in the FTSE 100
- Historical price data for each asset
- Covariance values of assets
- Exponential growth rate values of asset
- Transaction cost values

Module 2 – *Batch Data Downloader* – At the close of every business day the system will need to obtain the close of day prices for each of the assets in the database. Since this is a menial task, and as per the Specification, this will be automated in my application. The downloader will stream in files containing the latest asset prices and insert the data into the database. Once new price data has been inserted into the database the covariance and growth rate values will need to be re-calculated. This process should be spawned automatically once all of the data has been downloaded.

Module 3 – *Scenario Generator* – Scenario generation will be performed by the program *Cluster*, as described in the Background Research. This takes input in the form of a file and outputs the scenario information to a second file. In order for the system to optimise the portfolio over large time periods, rather than increasing the depth of the tree, it will have the option to represent either 1, 5 or 10 days at each level of the tree. In order to hook *Cluster* into my application I will create a wrapper which will encapsulate *Cluster*. This will mean that any other component wishing to perform scenario generation will not need to worry about performing the actual file Input/Output; it will just call on the wrapper.

Module 4 – *Optimiser* – The optimiser will need to take in the objective function, which seeks to minimise the CVaR of the portfolio, and a constraints matrix. The constraints matrix will implicitly contain:

- Minimum required return
- Transaction costs
- Current portfolio holdings
- Scenarios generated

After performing the optimisation it will return the value of the CVaR calculated, the future portfolio holdings and the future transactions.

Module 5 – *Graphical User Interface* – The Graphical User Interface is the only component with which the end user will interact directly. This component will be used to capture the inputs from the user and provide the means to start off the scenario generation and optimisation procedure. It will then display the results of the scenario generation and portfolio optimisation. The interface will also provide the means of importing and exporting portfolios, allowing the facility for portfolio histories to be generated.

4.2 Component Organisation

The component division shown in Figure 9 lends itself to the deployment of the application as a multi-tiered system. In an ideal situation Modules 1, 2, 3 and 4 would exist as individual services residing on a server(s), and Module 5 would exist as a thin-client application on the end user's machine. The thin-client could then use remote procedure calls to invoke the services to perform their respective tasks. Since tasks such as scenario generation and optimisation processes are highly processor intensive, placing them on a powerful server would allow them to run faster. It would also mean that, since the Graphical User Interface will just be displaying results, the end user's machine would not have to be extremely powerful. If a high speed communication medium also existed between the different services, the thin-client would have a means of quickly transferring the parameters to-and-fro, further increasing the speed of the overall application.

However, since I was unable to acquire a dedicated server to host the services, I was forced to re-organise my components so that the optimiser, scenario generator and graphical user interface are all part of the same package. This implies that the new package will have the following structure:

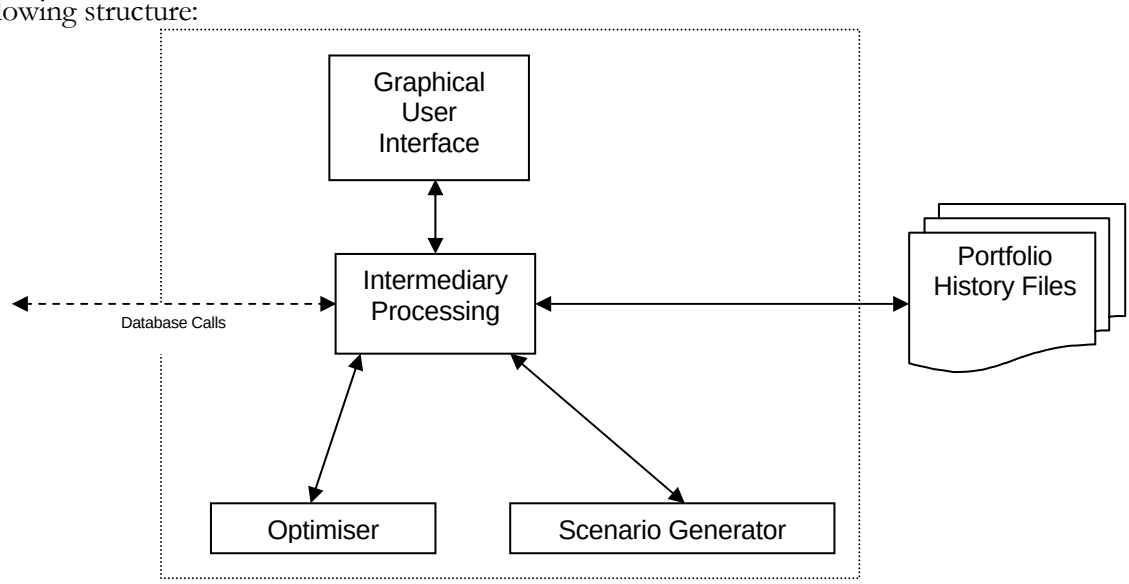


Figure 10

This package will exist on the client's machine and require a network connection to allow it to communicate with the database. In order to perform a complete optimisation the sequence diagram in Figure 11 highlights the steps that need to be taken:

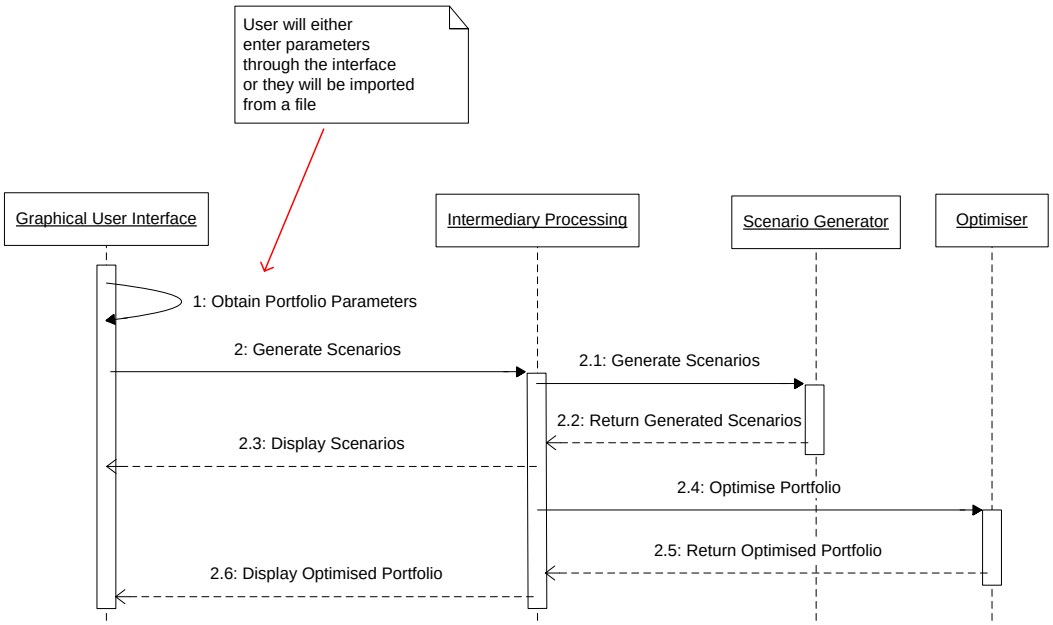


Figure 11

4.3 Graphical User Interface

As per the Specification, the Graphical User Interface needs to provide the user with a way to easily input the portfolio parameters and also display the outputs of the scenario generation and portfolio optimisation. Although there will be a large amount of data to display on the screen, not all of it will need to be viewable at the same time. For this reason, I will use a tabbed window structure to ensure that the user is not overwhelmed with information, but at the same time all of it will be easily accessible.

Figure 12 describes the overall structure of the application interface and has the portfolio parameters panel visible, highlighted in red. The annotations describe how the interface meets particular usability heuristics stated in [16].

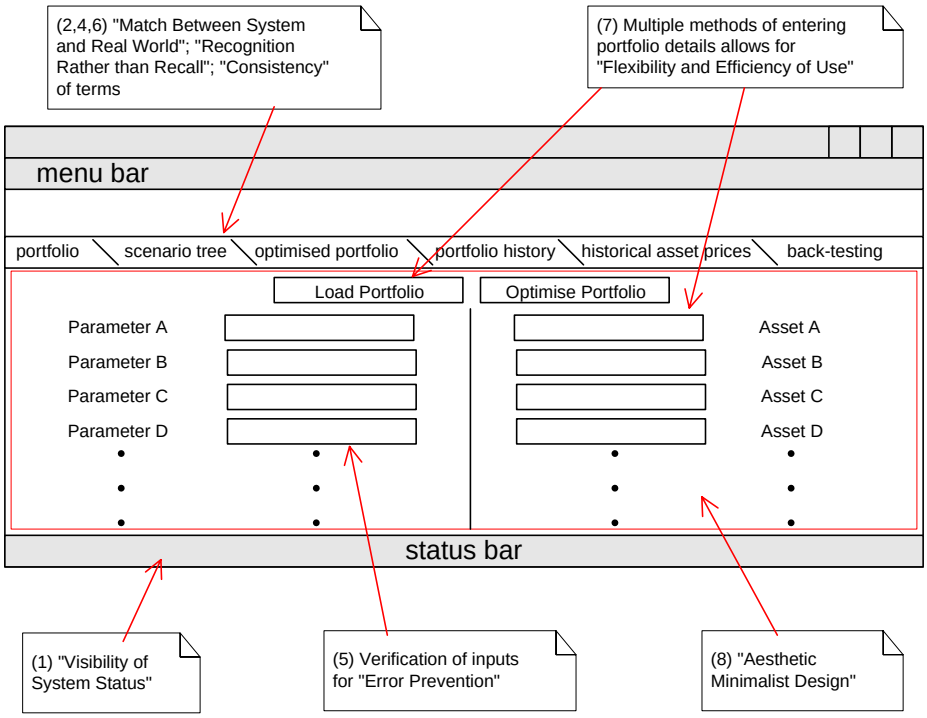


Figure 12

The details of the future scenarios and final portfolio optimisation will be displayed in a similar way. By drawing out the scenario tree visually, the user will find it easier to assimilate the displayed data. As can be seen from the figure below, I intend to design the interface such that tree is displayed on the left of the window, and by clicking on the individual nodes of it, the details of either asset prices or optimised portfolio weights can be viewed.

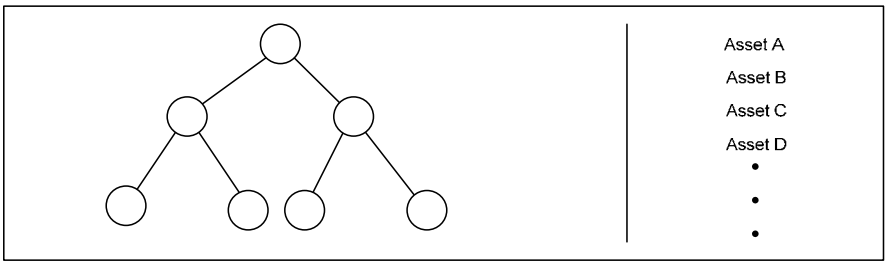


Figure 13

Chapter 5

System Implementation

5.1 Implementation Languages

The implementation requirements stated in the Specification, lends itself to a two-tiered system. A high-level language will be able to provide a good graphical user interface but will be slow to perform the back-end portfolio optimisation calculations. Conversely, since a lower-level language is closer related to assembly language, it will be able to compute the portfolio optimisation faster, but lack the ability to provide the user with an easy to use graphical interface.

Figure 14 from [2] can be used as an example to quantifiably compare the speeds of common high and low level languages. It displays the normalised speeds for sparse matrix multiplication with various matrices on a Pentium 1.5GHz PC running a Linux Operating System.

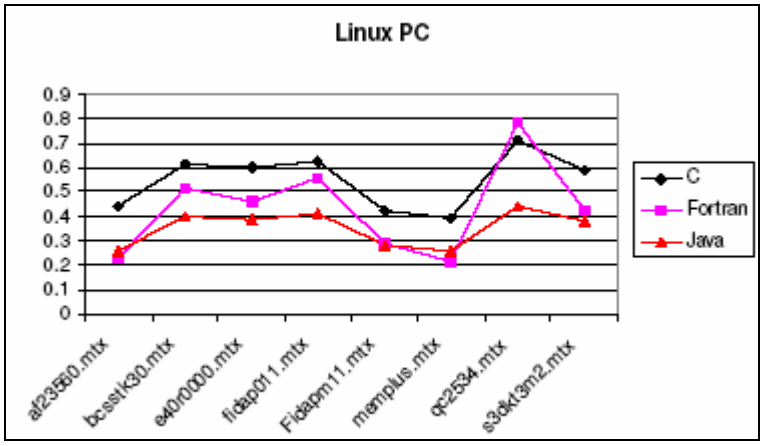


Figure 14

I have made the assumption that this is the minimum specification of a typical user's PC. Making use of its wide range of libraries, I have implemented the User Interface and intermediary processing in Java, a high level language. In order for the optimisation procedure to complete as quickly as possible I have implemented the numerical computation functions in FORTRAN, a low level language.

5.2 The Database

5.2.1 Database Implementation

In my system, the database is an integral component. It forms the core of the application, holding all of the historical financial data and statistical parameters for the optimisation procedure. The database will be a very active component of the system and will have a high throughput of data. The database has thus been designed in a way such that it is able to provide the facility to efficiently store the information as well as quickly retrieve it.

I considered the following options when deciding how to implement the database:

- Native XML database
- OLAP cubes
- Relational database

Since the database will be storing financial information which I am downloading from an external source, I considered storing the data in an XML format. This would be a good option if the financial data I am downloading is also in an XML format since no conversion of format would be required for the downloaded data. However, searching the database will be problematic. Maintaining and using index documents would bear a large overhead. In addition, when search results are to be passed back to the client machine, entire XML documents may be sent back, even though only a few attributes from each are required. This will lead to an excessive volume of data being transferred between the database server and client machine and extra processing will be required in order to extract the required data from the XML documents returned; which will increase the time it takes to perform a database query.

OLAP functionality is characterised by its multi-dimensional data model. Since my database will be storing a large quantity of data, searching it could potentially be very slow. OLAP cubes overcome this by taking snapshots of a relational database and restructuring it into multi-dimensional data. Queries can then be run against this at a fraction of the time of the same query on a relational database, since their execution effectively returns the intersection of the required attributes. The difficulty in implementing OLAP comes in formulating the queries, choosing the base data and developing the schema.

Implementing the database in a relational format will give it a concise structure that can be easily queried. As long as it is normalised there will be no redundant data in the database, and hence keep the data consistent and the size of the database low. It is also beneficial that queries can be constructed easily in SQL to interrogate the relational database. For these reasons I decided to implement my database in a relational format.

5.2.2 Database Structure

A relational database follows the relational model which states that data is represented as mathematical n-ary relations. The following Entity Relationship diagram describes the structure of the tables that exist within my database.

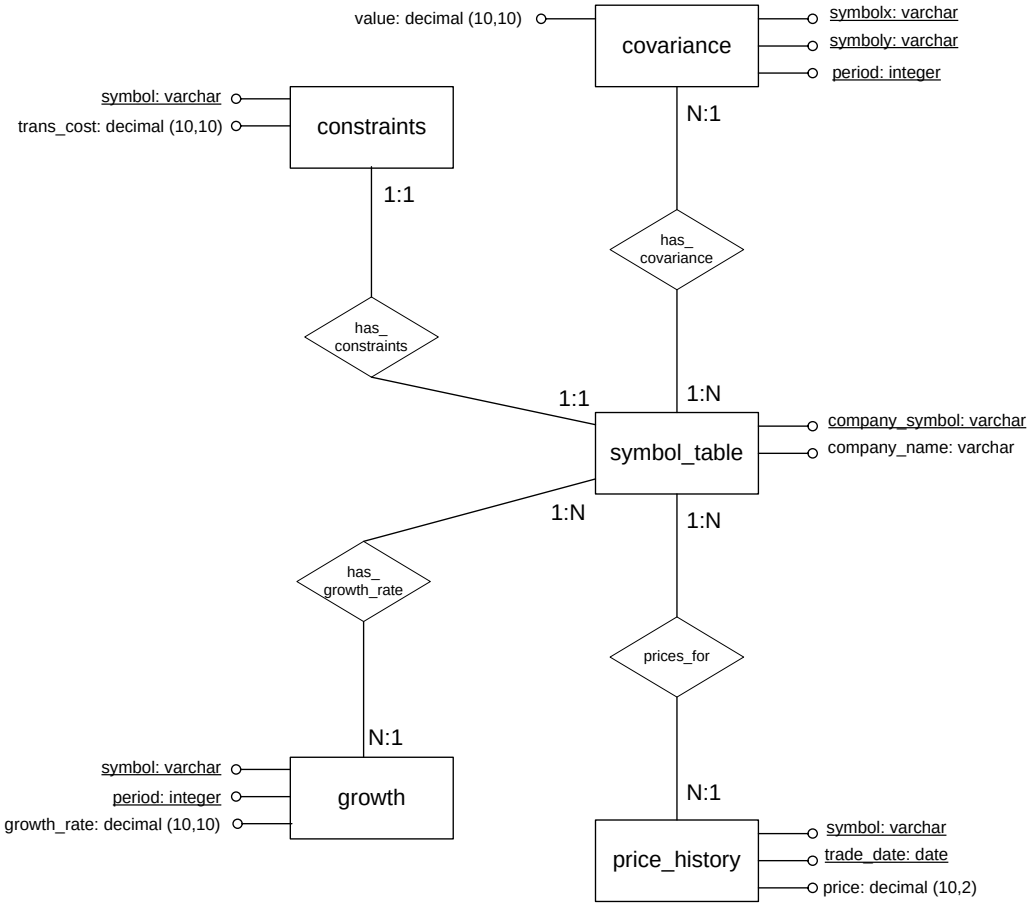


Figure 15

As can be seen from the diagram, the database is in third normal form since every non-prime attribute of each relation is:

- Fully functionally dependent on every key of the relation.
- Non-transitively dependent on every key of the relation.

Having the database normalised means that it is not possible to create data inconsistencies when adding, deleting and modifying data. It also means that no repetitive data is being stored, reducing the size of the overall database.

While implementing the database I wanted to maintain un-biased database structures as I feel that both reading from and writing to the database are equally important. For example, when calculating the covariance matrix for all of the assets within the system, it needs to parse through all of the historical price data to perform the calculation and then store the results back to the database. This task consists of high volumes of data being transferred both out of and into the database.

In order to achieve increased performance I have created SQL constraints on the tables. By enforcing primary keys on the database tables, the database management system (DBMS) implicitly creates an index structure on the key fields. This allows for the quick retrieval of data from the table if the data is being queried on the key. However, if I was to query the data using a non-key field, the DBMS retrieves data by performing a scan of the whole table. This clearly increases the time it takes for data to be returned.

By adding further indexes onto the table, data can be returned for queries using other fields by using these indexes, eliminating the need to scan through the whole table and thus decreasing the time it takes for them to run. However, in contrast, when writing data to a table with several indexes the DBMS must update each of the indexes, which obviously takes longer to perform when more indexes exist. Therefore, a balance needs to be achieved when deciding upon the number of indexes to attach to each table.

For my database, I have setup primary keys as stated in the above diagram. By analysing the queries I am running on the database, further described in Section 5.2.3, I discovered that all queries were using these fields and hence, these were the only indexes required for each table. In order to ensure data consistency in my database, I also created foreign key constraints on attributes of tables wherever required. This ensures that any manipulation of data is always consistent and no anomalous data can ever be stored.

5.2.3 Database API

Since the database is accessed by several different components of the system, I decided that it would be advantageous if it could be encapsulated by an API. This will ensure that other components can not illegally access the database and so that standardised results can be returned. I have implemented the API in two tiers:

- Stored Procedures
- Java API Class

A stored procedure is a program which is physically stored within a database. One benefit this provides is pre-compilation. When a stored procedure is created the query plan is calculated once, rather than every time the query is submitted. This removes the ad-hoc SQL compilation overhead that is required in situations where actual SQL queries are piped to the database. Having stored procedures will therefore mean that queries can be executed faster.

Stored procedures also allow the underlying tables to be encapsulated. From a security point of view, this means that access to the database is only allowed through narrow, well-defined procedures. Rather than having the other application components interact with the tables and other database objects directly, they will only be allowed to run the stored procedures.

As mentioned in Section 5.2.2, I conducted analysis of whether I needed to create any extra indexes on the tables in the database. To do this, I formulated a list of the representations of the stored procedures which search for data using relational algebra. This allowed me to see which fields are being used in my queries, and analyse whether any do not have an index on them. Appendix Section A contains a list of the stored procedures in my database and their relational algebra representation. As can be seen from the list, the stored procedures only contain select clauses on primary key fields of the tables; which is why I did not add any further indexes to them.

On top of the stored procedures I have written a Java API class which manages the connection of the application to the database server and also provides an interface to access the database. Each stored procedure stated in the list is accessible through its respective function in the API. Rather than simply return an abstract object result set, the API returns an object of the same data type as the underlying data. This eliminates the need for other components to parse the data being returned by the database and convert it into the required data-type.

I have implemented the database API as a singleton class to ensure that the client application only makes at the most one connection to the database server. This means that it will not be swamped by several connections from the same machine all requesting high throughput. Although I have not had to, having a singleton class also means that I could restrict the number of concurrent database calls the application makes, which could be used to avoid creating too much network traffic or to control the load on the database server.

5.3 The Data Downloader

5.3.1 Data Source

The historical prices of assets in the database form the underlying basis for the scenario generation and hence the optimisation procedure. Since I intend Portfolio Managers using my application to be interested in making long-term capital gains, I will only need to acquire close of day prices and not intra-day prices. I have stated in my Specification that I will include assets from FTSE 100 in my database. A free source of this data is "Yahoo! Finance UK". They make available a comma separated value (.csv) file which contains the latest prices of all of the components of the FTSE 100 exchange. This file can be downloaded over the internet via the URL:
http://uk.old.finance.yahoo.com/d/quotes.csv?s=@%5EFTSE&f=sl1d1t1c1ohgv&e=.csv

5.3.2 Data Acquisition

In order to download the data and insert it into the database, the following class structure was written:

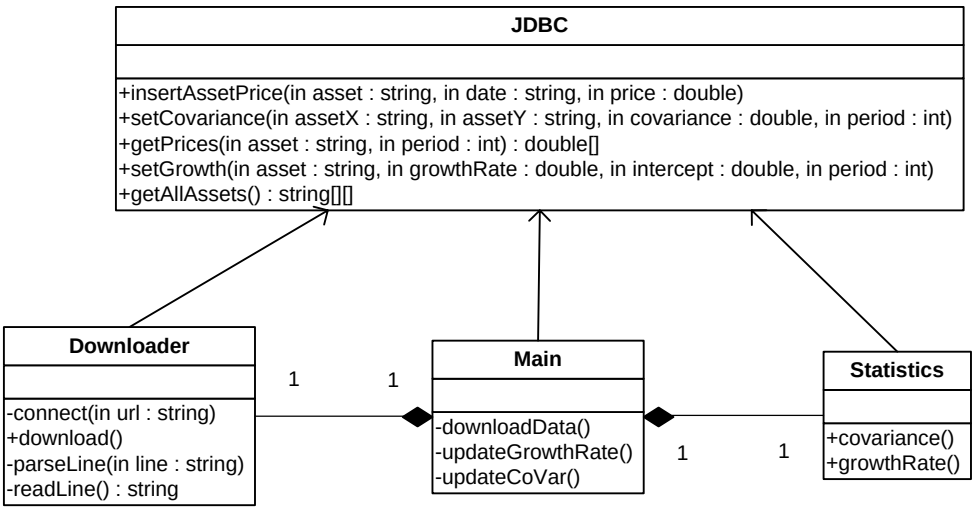


Figure 16

The Main class initiates the downloading of data, after which it instantiates the Statistics class to perform the updating of the parameters. All three classes reference the singleton database API in order to insert and update data.

5.3.3 Parameters Re-calculation

As is described in further detail in Section 5.4, in order to generate future scenarios, the Scenario Generator requires values for the growth rate and covariance of assets from the mean exponential growth curve. Once a new set of prices have been calculated, these values need to be recalculated. For this reason, I have included this process as part of the Data Downloader module. Once the new price data has been downloaded, the module performs the recalculation of the parameters and stores them in the database.

5.3.4 Job Automation

The downloading of prices and re-calculation of parameters is a task which will need to be done at the close of every day. Since this a menial task that does not require any human interaction, it lends itself to be fully automated. Having the job automated also means that when users wish to optimise a portfolio they can be sure that the latest price data and parameters have already been calculated, speeding up the whole optimisation process.

With the help of Duncan White, from CSG, a CRON was set up to run at 19:00 every weekday. The job runs a batch script in my user space which in-turn executes the compiled Java Data Downloader package.

5.4 The Scenario Generator

5.4.1 Cluster

To generate the future scenarios I will be using *Cluster*, introduced in my Background Research. I did not have access to the source of the program so the only method of executing it was through the command line. This is not the ideal way, since it would be quicker to be able to call the Cluster method and directly pass parameters to it via memory. By having to output the inputs to a file and then parse the output file to place the outputs back into memory, the overall execution time will increase since it is slower to perform file I/O rather than memory I/O.

Cluster requires as input a file with the following structure:

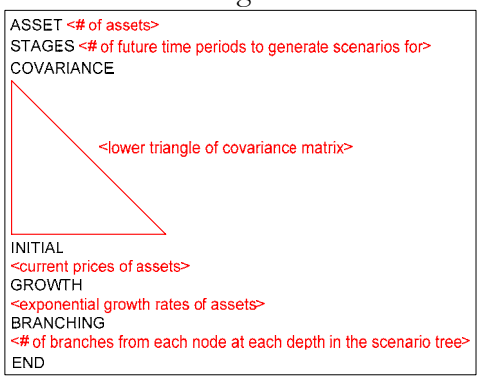


Figure 17

The output of *Cluster* is designed to be read in by *Foliage*, a multistage portfolio optimiser developed by the Department of Computing at Imperial College London. Not all of the output file will be required by my system. The useful section will begin with the "SCENARIOS" tag and have the format stated below:

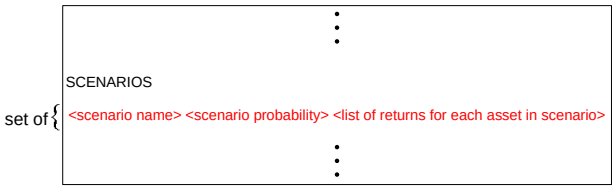


Figure 18

Cluster employs the following naming convention for scenarios:

$$\forall n_1 : "n"$$
$$\forall n_l :< name\_of\_parent > + "n"$$
$$0 \le n_l < b_l$$

where n_l is node n on level l, and b_l is the branching rate of level l

Figure 19 below displays a grounded example of a scenario tree with 2 levels and a branching rate of {2,2}.

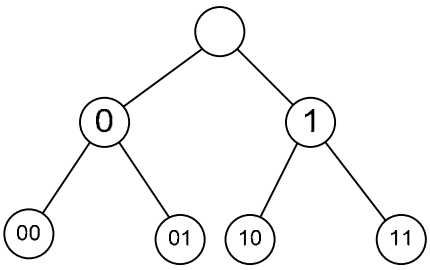


Figure 19

By default, *Cluster* will perform 100 simulations for each level and then cluster them to produce the final output nodes. To increase the accuracy of the outputs I will request the program to perform 10,000 simulations.

5.4.2 Java Wrapper

In order to encapsulate *Cluster* from the other modules in my application, I have written a Java wrapper around it. This wrapper can return:

- future prices in the form of a scenario tree
- a visualisation of the scenario tree generated

In order for future prices to be generated, the wrapper requires from the user:

- the number of future time periods to consider (number of levels in the tree)
- the number of days which constitute a single time period
- the branching rate (I have set up the wrapper such that the branching rate at each level will be the same)

These parameters will be obtained via the Graphical User Interface.

As described in Section 5.2, the current prices, growth rate and covariance will be readily available in the database. Due to the re-organisation of components stated in Section 4.2, I have implemented the wrapper in a way such that it will not directly make database calls to obtain the asset data. This will also be passed in by the calling component.

Once initialised, the Scenario Generator produces a file, as per Figure 11, and executes the command:

"./cluster –f <input filename> -o <output filename> -n 10000"

Once the command has been run, the module parses the section of the output file stated in Figure 18 and loads the values into memory.

The scenarios generated will be returned as a HashMap<String, Scenario>. I have used the name of the scenarios, as produced by *Cluster*, as the key element for the HashMap. Since the current prices are not required to be outputted by *Cluster*, but required by components which make use of this HashMap, I have added them into the HashMap defining the root node with the name "I". I have created a Scenario class as the value element for the HashMap, which is defined as follows:

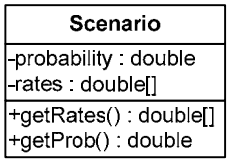


Figure 20

Given the initial prices and the HashMap of scenarios, the full set of rates of return for each asset at each of the nodes of the scenario tree can be generated and used in the optimisation procedure.

In addition to being able to return the HashMap, the Scenario Generator can return a Java Panel which visually displays the scenario tree and the prices of each of the assets at each of the nodes. This is of great of importance since it shows the user, in a non-overwhelming manner, the data which will be used to optimise the portfolio.

As the optimisation procedure is complex, this is one of the features I have implemented to explain to the user how the application works. Features such as displaying the scenario tree with the future prices will make the user more confident that the optimisation output is correct. By increasing the confidence of the user in the application, they will use it more readily.

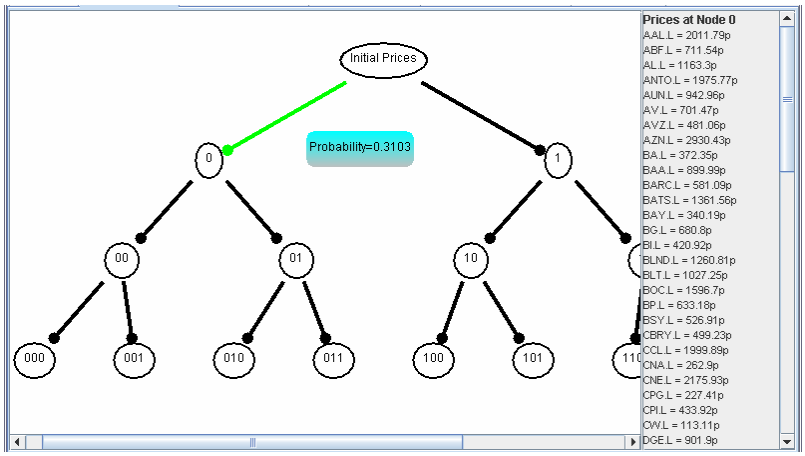


Figure 21

Figure 21 displays a snapshot of a Panel returned by the Scenario Generator. As can be seen, the scenario tree is drawn on the left of the panel, with the asset prices of the selected node displayed on the right. The user can select which node's prices should be displayed by simply clicking on the required node. In addition, as in the figure, by placing the mouse over an arc of the tree, the probability associated with that arc is displayed.

5.5 The Optimiser

5.5.1 Optimisation Library Selection

In order to perform the optimisation, I need to make use of an optimisation library. The optimisation, given an objective function and a constraints matrix, will return the optimal values of the variable. As stated in Section 5.1, implementing the optimisation in Java would lead to a slower optimisation than if using an optimisation procedure written in a lower level language.

Since numerous optimisation libraries already exist in the public domain, I did not need to write my own. My project supervisor suggested that I use an optimisation library produced by the Numerical Algorithms Group (NAG).

Looking at my optimisation problem, I noticed that constraint (36) was the only non-linear constraint. Therefore, for the purposes of this project, I have decided not to implement this constraint. Keeping the optimisation as an LP problem enabled me to use a Linear Programming library which is able to optimise my objective function in a quicker amount of time. The only disadvantage of not having constraint (36) in my optimisation problem is that I can not force portfolio diversity. I felt that this was a fair compromise. In the end I chose the E04MFF library[1]. This is a Linear Programming library written in FORTRAN.

However, whilst working with NAG I encountered a few problems. Firstly, the project coincided with the Mark 21 release of the library. This meant that it was only available for use on a departmental server in the middle of January. Secondly, since my main application is written in Java, as is further discussed later in Section 5.4, the standard method of running the optimisation would be by using JNI to run a C procedure which, in turn, will directly run

[1] Documentation of this library can be found at http://www.nag.co.uk/numeric/fl/manual/pdf/E04/e04mff.pdf

the optimisation procedure. However, confusion existed on whether the Mark 20 header files, required for JNI, would be forward compatible with the Mark 21 library, since Mark 21 header files were not available. This added un-necessary delay.

In addition, when I performed trial runs of the optimisation, an IFAIL flag (error notification) was raised. On consultation of the documentation, I noticed that the IFAIL flag being returned was used to signify that a "weak" solution had been found. A weak solution implies that a local solution has been found, and not a global one. This can not be possible since the optimisation problem is Linear, implying that a local solution can not exist, due to it being, as shown below, inherently convex.

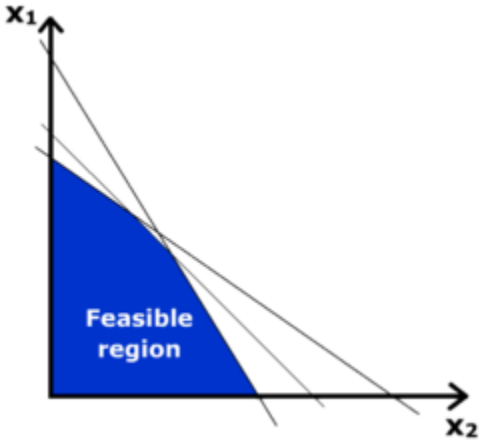


Figure 22

After communicating with NAG, it was discovered that the documentation for the E04MFF library was poorly worded. For this library, the IFAIL flag signifies that a non-unique solution exists in the optimisation problem. In terms of producing the optimal VaR value, this does not pose a problem. Having incurred the problem outlined, there was a time delay in finding the explanation. This caused an interruption in continuing my project.

Since, at the time, I was not sure whether all my problems could be rectified in time, I decided to search for another optimisation library as a backup. Even though this would take some time away from the rest of my project, finding a working optimisation library was of paramount importance. I finally chose the GNU Linear Programming Kit[1] (GLPK) as my backup. This is a freeware optimiser written in C.

In the end, I have used the NAG library; however, as is discussed later in Section 5.5.2, the delays encountered meant that I was not able to link it into my application using the optimal method.

5.5.2 Interlinking the Optimisation Library with the Application

In order for the optimisation library to be useful to my application, I had to implement a wrapper which would provide an interface for the other application modules to perform an optimisation. As briefly mentioned in Section 5.5.1, the ideal way to interface a Java program with a FORTRAN function is to use Java Native Interface (JNI).

[1] Documentation of this library can be found at http://www.gnu.org/software/glpk/

JNI is a programming framework that, using a header file which defines the interfacing functions, allows Java code running in the Java Virtual Machine to call and be called by native applications and libraries written in other languages, such as C, C++ and assembly[1]. By coding a small C function which could initiate the optimisation procedure, Java would be able to pass all parameters by converting the Java objects into native objects within memory, and receive the results via memory too.

As mentioned in Section 5.5.1, the delays encountered with NAG meant that I did not want to use my time to create the required JNI files for NAG in case I would end up using another optimisation library, such as GLPK. I, therefore, took the decision to implement the wrapper in a way such that, as with the Scenario Generation, it would create a parameters file which could be piped into the optimisation procedure, and the results would also be written to a file, which could be parsed by the wrapper in order for them to be put back into memory. I took this decision as I knew that, regardless of which optimisation library I ended up using, I could pass parameters to it via I/O files, but could not be a hundred percent sure that JNI could be used.

This has meant that the initialisation and finalisation of the optimisation procedure is slower than it could have been, but given the circumstances, I feel that not much could be done. If extra time was available, this is one part of the implementation I would redo.

5.5.3 Optimiser Wrapper

Since I decided to pass in and return values via files, I wrote a small FORTRAN script which could read in the input file, of the format stated in Figure 23, call the library optimisation procedure and write an auto-generated output to a file. An example of an auto-generated output file can be found in the Appendix Section B. Once compiled into an executable file, this script can be run through Java via the command-line.

In order to perform the optimisation, the E04MFF library requires the following parameters to be passed in:

- Objective function
- Constraints matrix
- Upper bounds on constraints and variables
- Lower bounds on constraints and variables
- Initial value of variables

[1] As defined at http://en.wikipedia.org/wiki/Java_Native_Interface

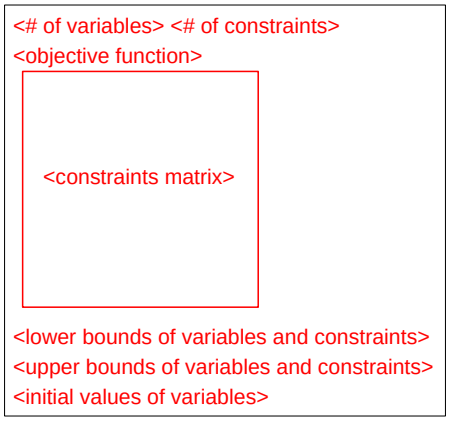


Figure 23

In order to pass the required data to the optimiser wrapper, I have created a parameters class, as per the class diagram in Figure 24. This enables all variables to be bundled together and easily passed around.

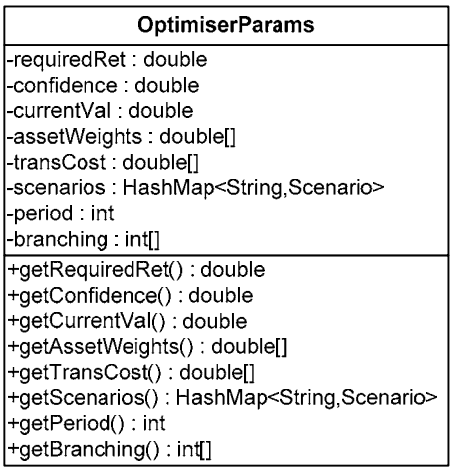


Figure 24

The optimiser wrapper will always be instantiated by the Graphical User Interface, so I have delegated it the responsibility of obtaining the required parameters. The "scenarios" variable refers to the HashMap passed back from the Scenario Generator. The "transCost" variable refers to an array of the transaction costs associated with each of the assets in the database. This can be obtained from the database. The remainder of the parameters are entered directly by the user via the User Interface.

Analogously to the Scenario Generator, the Optimiser can return a class, OptimisedParams, containing the results of the optimiser procedure, or a Java Panel, which visually displays the results. The panel is described in greater detail in Section 5.5.5.

5.6 The Graphical User Interface

The main application window provides the means by which the end user can interact with the application. However, it is also the controlling module for the optimiser and scenario generator. Consistent with the Separation of Duties principal, these two tasks have been separated, as can be seen in Figure 9, into two sub-packages; Graphical User Interface and

Intermediary Processing. This allows there to be high cohesion within classes but low coupling between classes.

5.6.1 Interface Concepts

Whilst implementing the user interface I bore in mind Nielson's Ten Usability Heuristics, stated in [16]. There are some quite complex concepts that my Graphical User Interface has to display to the user in an as simple and usable manner as possible.

I have also striven to make the Graphical User Interface aesthetically pleasing. I have done this in the following manner:

- Keeping continuity within the application, in terms of interface components used and layouts
- Using an underlying blue colour scheme
- Designing a logo for my application which gives it a unique identity
- Using images that do not impose on the user's workspace
- Using customised interface components and fonts

Since there will be periods of processor intensive activity when the interface will not have anything further to display for a while, I have made sure that user is aware that the program has not frozen with the use of indeterminate progress bars.

As I had planned in my System Design, I implemented my application with a tabbed pane being the main structure. This means that the user is not overwhelmed with the information currently on the screen. Having the interface split into separate tabbed panels, also means that while the long processes, Scenario Generation and Optimiser, are occurring, the user is able to roam around the interface with access to the panels currently being worked upon restricted. Threading of the Scenario Generation and Optimiser modules has allowed the user to concurrently access other features of the application.

5.6.2 Application Graphical User Interface

I noticed that when I performed a cold start of my application, it would take a couple of seconds to load up my main application window. This is because the application must initialise all of the components for the interface and also establish a connection with the database server. To ensure that the user is not kept looking at a blank screen I introduced a splash screen, seen in Figure 25. The splash screen houses a logo for my application and a progress bar, which informs the user that the application is indeed loading. Threading the main application window and the splash screen, allows both to run in parallel.

Figure 25

Once my application has been loaded, the user is presented with the screen shown in Figure 26. This window provides easy access to all features of the application, but also ensures that they are clearly defined and separated so as not to overwhelm the user. The tabbed pane allows for a common interface to surround the changing panel in the centre of the screen. This provides the basis of the look and feel of continuity in the application. As can be seen in Figure 26, the tabbed pane controls which tabs the user is able to view and hence is vital in making sure that the user's workflow is not interrupted by blocking access to panels which the user feels should not be blocked.

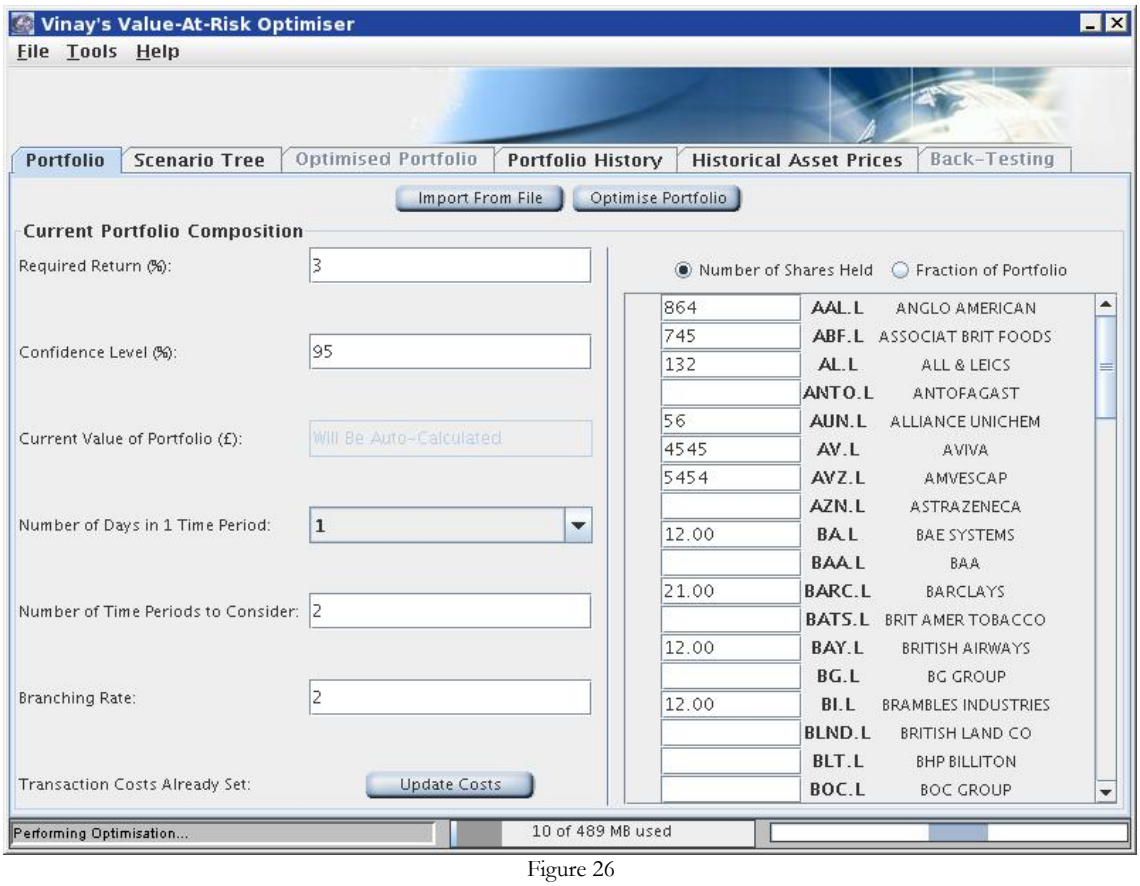


Figure 26

The status bar at the bottom of the application interface was initially designed to serve two main purposes. Firstly, as can be seen on the left, it is used to pass messages to the user. The messages can either inform the user on what task the application is currently performing or display error messages to the user. The right side of the status bar hosts an indeterminate

status bar. This ensures that, without making the user switch context, he or she can is able to see that the application hasn't frozen and is indeed busy.

A later addition to the status bar was the memory usage indicator, which can be seen in the centre. In the initial development of my application, I was running quite small optimisation problems, but as the application development progressed I was able to perform larger tasks. These larger tasks sometimes caused there to be a memory overflow error. By default the Java Virtual Machine allocates 64Mb of Heap Space to the application. However, by using the memory usage indicator, I realised that the memory usage would peak at around 70Mb. Given the resources in current desktop machines, I deem this to be acceptable. By going to the "Tools" menu item, I have included the facility for the user to force a Garbage Collection to keep memory usage as low as possible.

The menu bar also contains the "File" menu item, which provides the user with a way to exit the application, and a "Help" menu item which provides the user with an "About" panel for the application, as can be seen in Figure 27. The project website contains a user guide, attached in this report's Appendix, which users can use to aid them whilst working with the application.

Figure 27

The following sections discuss each of the panels in my application's interface.

5.6.3 Current Portfolio Composition & Parameters Panel

As can be seen in Figure 26, this is the first panel users view when they open the application. This panel is used for entering the parameters to be used in the optimisation procedure. The panel is essentially divided into two with the parameters for the optimisation on the left, and the current portfolio holdings on the right. This allows for the easy assimilation of the panel for the user.

In order to verify the validity of inputs, each of the input text boxes are of the type JFormattedTextField. This allows me to attach to them an instance of the DecimalFormat class with the format "00.00" being used for decimal numbers and "00" being used for integer, as required by Front-End Specification in section 3.1.1. I have extended Java's

InputVerifer class and defined my own verify() method which causes an error beep and does not allow focus of the text component to be yielded if an ill-formatted value has been entered. The class structure of each JFormattedTextField is thus as follows.

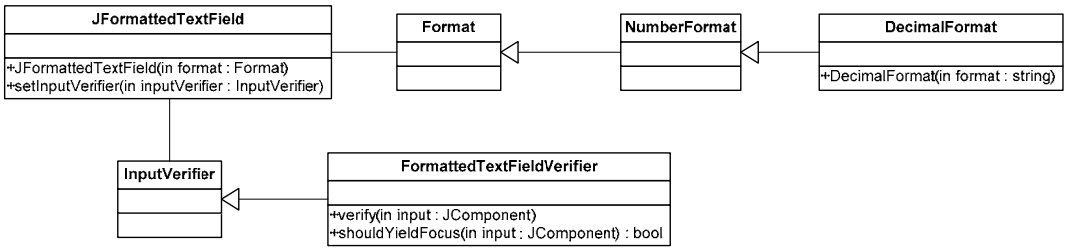


Figure 28

As can be seen on the right hand side of the panel in Figure 26, I allow the user to have the option of inputting the portfolio composition either by number of shares held or as a fraction of the total portfolio wealth. If the "Number of Shares Held" option is selected, as shown the input text box for the current value of the portfolio is greyed out, as the application will be able to auto calculate it using the current prices. This both saves time for the user, and also removes a source of possible discrepancy in the data being inputted.

In order not to clutter the screen, I have placed the updating of transaction costs into another JFrame. I have also taken this decision for the reason that the chance of the user wanting to change transaction costs is slim, so this concern can be separated into another window. The application remembers the transaction costs last entered, so they are not required each time the user wishes to run an optimisation procedure. Figure 29 shows the Update Transaction Costs panel. It has a similar format to that of the portfolio composition section of the main panel, continuing the concept of continuity in my application design.

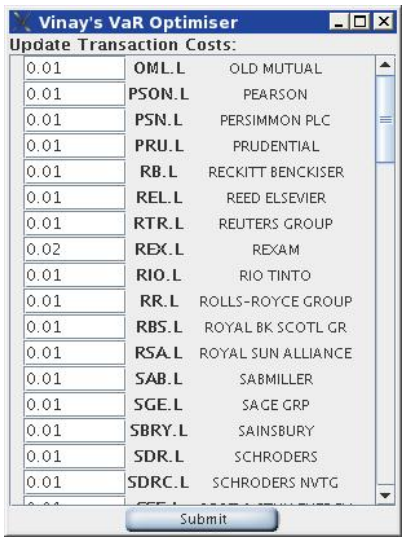


Figure 29

Consistent with Nielson's idea of providing shortcuts for experienced users, as stated in [16], the application also gives the user the option of importing a portfolio from a file. I have also included this feature to demonstrate, how this application, in a commercial surrounding, could link to other up-stream systems which stream in portfolio data. I have taken the arbitrary decision to accept comma-separated files (.csv) and my own defined (.var) files.

The rows of the (.csv) files have the following structure:

<company symbol>,<number of shares held in company>

This seeks to demonstrate how a simple file format, such as (.csv) or XML could be parsed in order to automatically populate the portfolio composition portion of the panel.

I have created the (.var) file format to demonstrate how all of the parameters required for the optimisation procedure could be piped into the application from an external source. The structure of a (.var) file is described in Figure 30. As with the (.csv) files, the (.var) files contain the portfolio composition. The (.var) files also contain a list of the parameters required for the optimisation and basic portfolio history to demonstrate how this application could be used to keep track of portfolios. This data could also be stored in the database or be formatted in an XML message, both of which just require a different parsing method.

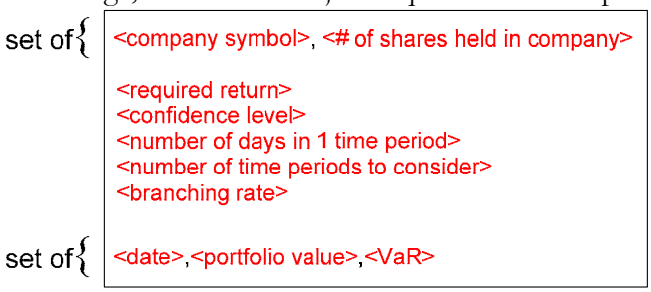


Figure 30

Once the parameters have all been entered, the user is able to begin the optimisation procedure by clicking on the "Optimise Portfolio" button. Validation checks are performed by the application to ensure that all required parameters have indeed been entered and are consistent. Once verified, the optimisation procedure can begin as planned in Figure 11.

5.6.4 Scenario Tree Panel

The first step of the optimisation procedure is to generate the scenario tree. As described in Section 5.4, the Scenario Generator module returns a HashMap containing the scenarios and a panel which visually describes the future scenarios generated. The HashMap is passed, along with the optimisation parameters which have been entered, as in instance of the OptimiserParams class, described in Section 5.5.3.

As can be seen in Figure 31, once the scenario tree has been generated, the tabbed panel allows user access to the "Scenario Tree" panel. Threading of the Scenario Generator and Optimiser modules is advantageous in this situation since the user is able to view the generated Scenario Tree whilst the application is still performing the optimisation.

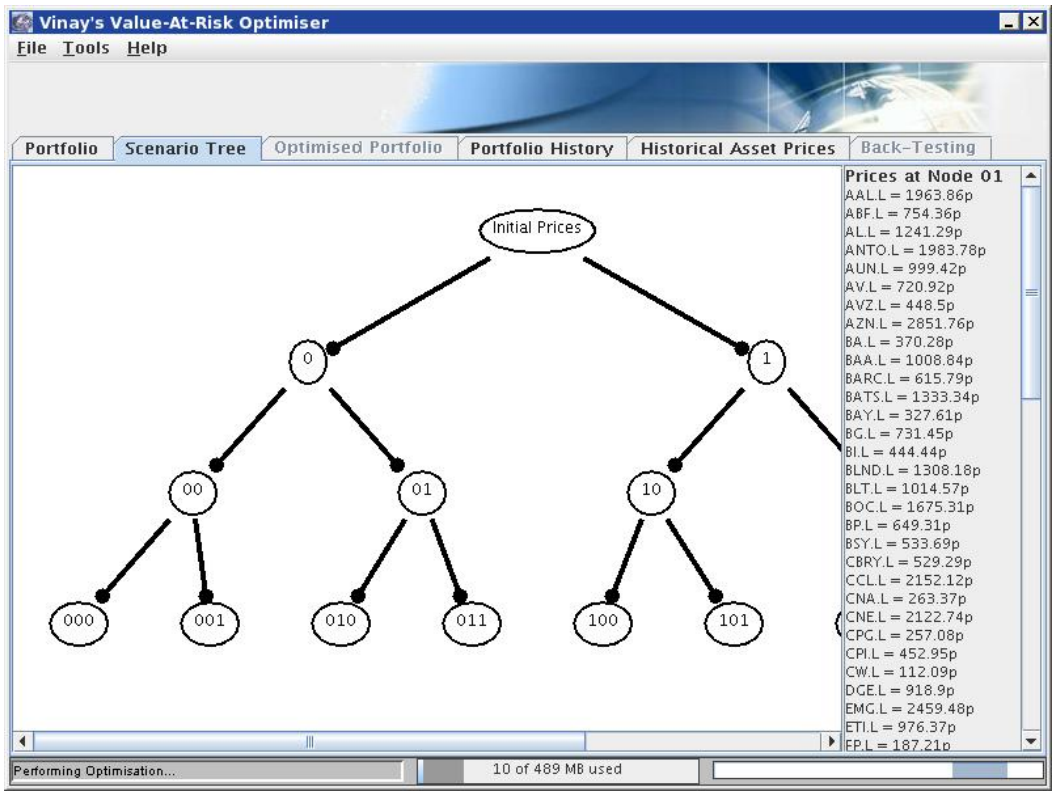


Figure 31

5.6.5 Optimised Portfolio Panel

The first step of the optimisation procedure is to generate the scenario tree. As described in Section 5.5.3, the Optimiser module can return a panel which can be displayed for the user to see the results of the optimisation. This is shown in Figure 32.

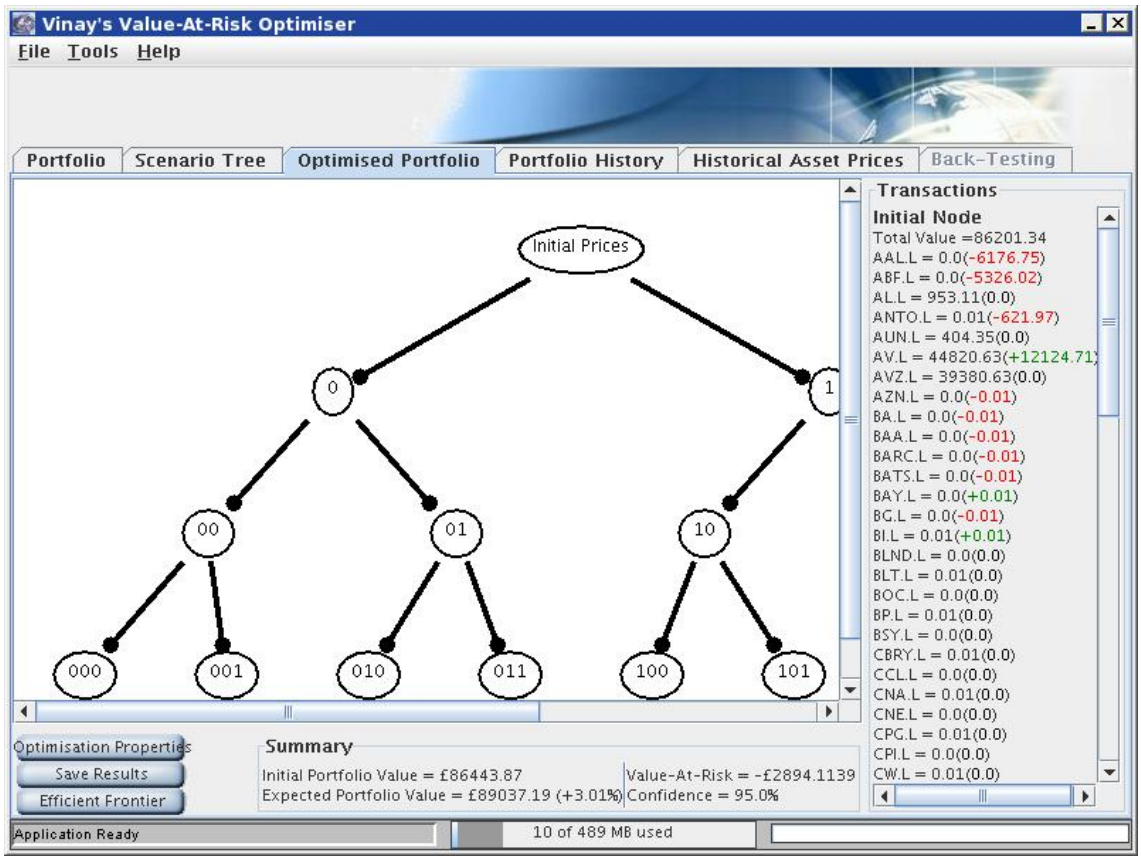


Figure 32

Again for continuity, this panel is structured similarly to the Scenario Generation panel. The scenario tree is used as a method by which the user can navigate through the transactions the application is recommending at each time period in a simple manner. By clicking on the nodes of scenario tree, the right panel detailing the holdings and transactions required at that node are highlighted. Each row of the details panel has the following structure:

<company symbol> = <value of holdings in company> (<value of shares which have been bought/sold at node>)

There also exists a summary panel at the bottom of the window. This panel displays the summarising facts about the optimisation. This allows the user to easily gain a high level view of the optimised portfolio. In addition, by clicking on the "Optimisation Properties" button on the left, the view can see a couple of the technical details of the optimisation, as per Figure 33. The workspace figures denote the amount of memory resources the optimisation procedure used.

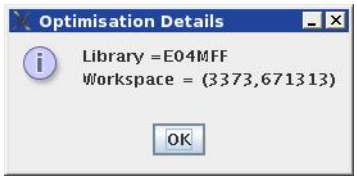


Figure 33

The application also allows for the computation of the efficient frontier. By clicking the "Efficient Frontier" button, the window in Figure 34 is displayed. The user needs to enter a range of required returns over which the curve should be plotted and the number of data points which should be plotted.

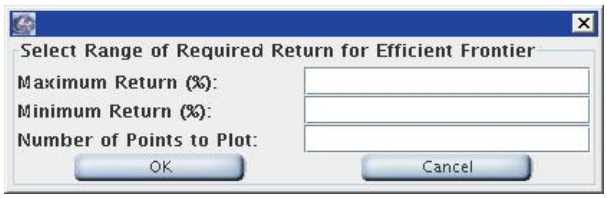


Figure 34

As with the portfolio composition panel, the input text boxes have automatic validation set up. Once the user has entered the details, threads numbering the number of data points to be plotted are created. The optimisations can all occur in parallel since they are independent of each other.

Each thread performs the same optimisation but with a different required return value. For this reason, the only difference in the parameters being fed into the optimiser is that the lower bounds on the returns constraint in the constraint matrix. This means that it is inefficient for each thread to produce its own output file from scratch. Realising this, I modified the Optimiser module so that when producing the original input file for the FORTRAN optimiser script, it also produces a duplicate temporary file. Thus, in order for the threads to produce their input file, they can simply make a copy of the temporary file and modify the lower bound for the returns constraint. Having each of the points calculated in parallel decreases the total amount of time required to calculate each of the points on the efficient frontier.

As each thread completes the optimisation, it returns the results an instance of the OptimisedParams class. These are collected by the thread handler in a synchronized method. Once all of the threads have returned their results, the thread handler plots the efficient frontier, and generates the window as per Figure 35. The top of the window hosts a drop down list which the user can use to select which return he or she is most comfortable with. By clicking on the "OK" button, the selected optimisation's details are displayed on the main Optimal Portfolio panel.

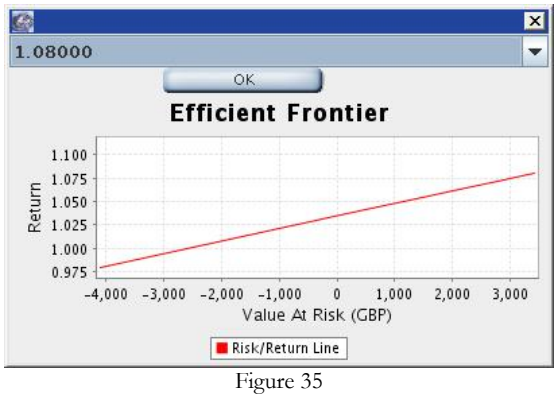


Figure 35

The optimisation panel also gives the user the facility to export portfolios. Analogous to the importing of portfolios, the exporting of portfolios seeks to demonstrate how my application could be linked to downstream systems in a commercial scenario.

The user is presented with the option to export a (.jpg), (.csv) or a (.var) file. Selecting the (.jpg) option allows the user to export an image of the scenario tree. This allows the user to keep a record of the structure of the produced scenario tree. Selecting the (.csv) option allows

the user to export the list of transactions at the Initial Node, as recommended by the optimisation process. This is to demonstrate how my application could send onto a booking system the transactions required for a portfolio. Each row of the (.csv) file has the following structure:

<company symbol>,<value of shares to buy/sell>

Selecting the (.var) option allows the user to export a file containing the historical information about the portfolio and the optimisation parameters. The structure of the file is as described by Figure 30.

5.6.6 Portfolio History Panel

I have included this panel as per Section 3.3 specifying possible extensions to my core specification. It has been designed to demonstrate how my application could fit into the normal workflow of a professional portfolio manager. Since the (.var) files, which are imported into the application, contain portfolio history values and VaR values, these can be plotted on a graph so that the user can visually see how profit and VaR have been affected over time. This is an advantage over an application that just performs one off optimisations.

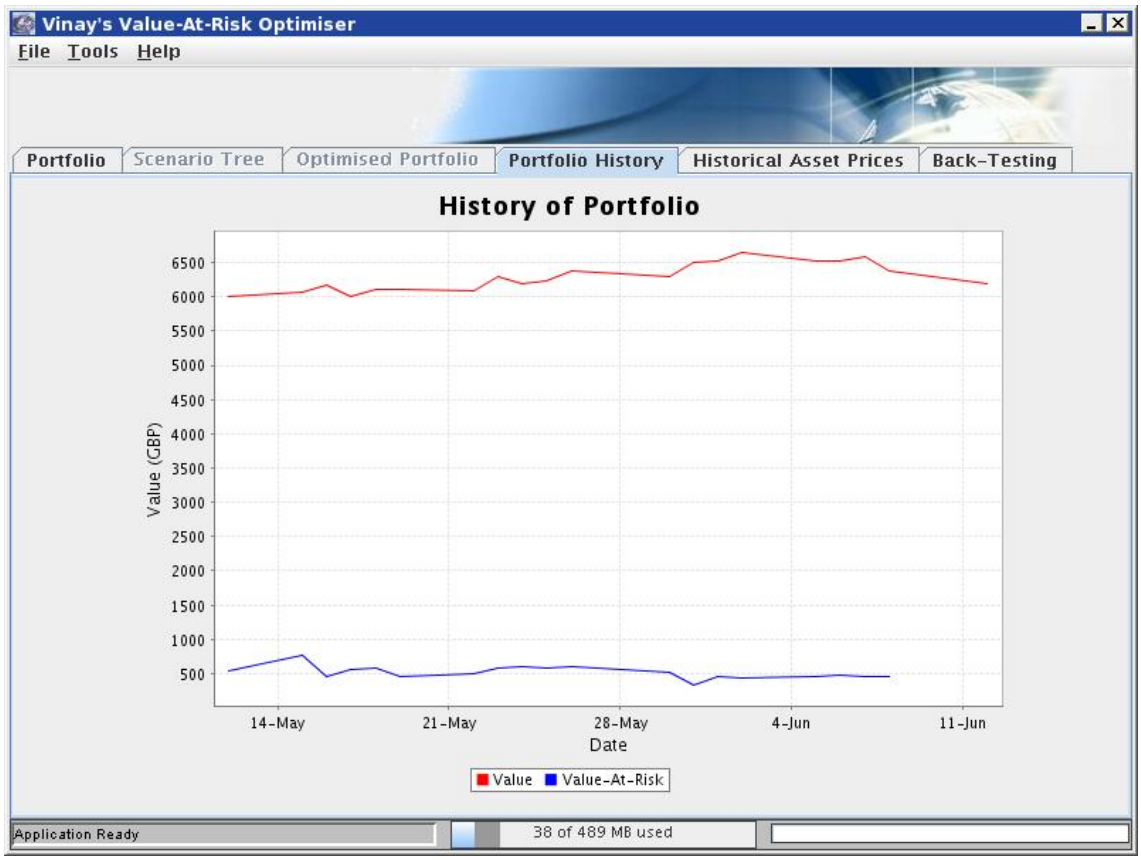


Figure 36

5.6.7 Historical Asset Prices Panel

This panel has also been included as per Section 3.3 specifying possible extensions to the Specification. Since the database already contains historical asset prices for the calculation of the parameters for the scenario generation, this data does not need to especially be downloaded. As can be seen in Figure 37, I have structured the graph so that the user can

check the company symbols on the right side of the panel in order for their prices to be displayed on the graph. This means the user has full control over which assets are being displayed.

In order to plot a line of the historical prices of an asset, the panel needs to obtain from the database each of the historical prices for each of the assets. This is a lot of data to transmit through the network. I have, therefore, decided to download and cache asset prices ad-hoc. If a user checks an asset which has not been previously cached, the data is downloaded and, cached and then displayed. This means that the first time a user selects an asset, there will be a short lag, but the user will not be faced with a long wait during the application start up whilst all of the data is downloaded in bulk. This is also beneficial, as the user would not always want to look at historical prices for each of the assets in the exchange; so pre-downloading all of the price data would mean that un-necessary data would inevitably be downloaded.

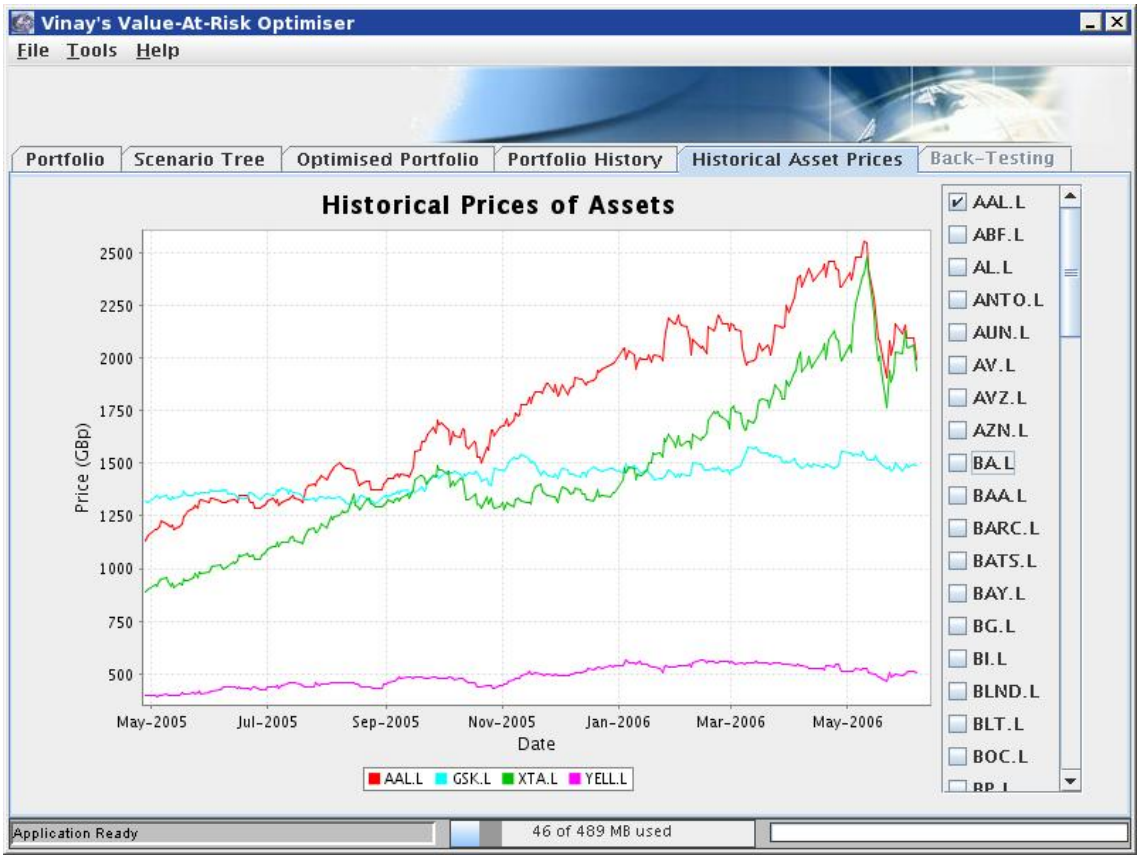


Figure 37

5.6.8 Back Testing Panel

In order to verify the developed VaR model, I have implemented an automated back testing facility. For reasons, which will be discussed in Section 5.7, the back tester has been implemented as a separate Java program which outputs a results file, of the structure described in Figure 38. Since the output of the back tester is a text file, it can be opened within the application for the user to visually view the results.

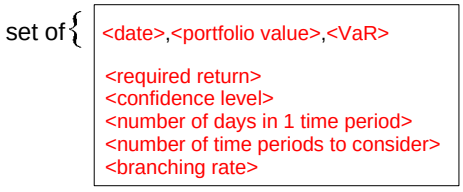


Figure 38

As can be seen in Figure 39, the back testing panel holds a graph displaying the wealth and VaR of the portfolio over the time period covered by the back testing. The other graph compares the rates of returns of the portfolio over time with the rates of return of the benchmark, the FTSE 100. The right side of the panel displays summarising facts about the back testing.

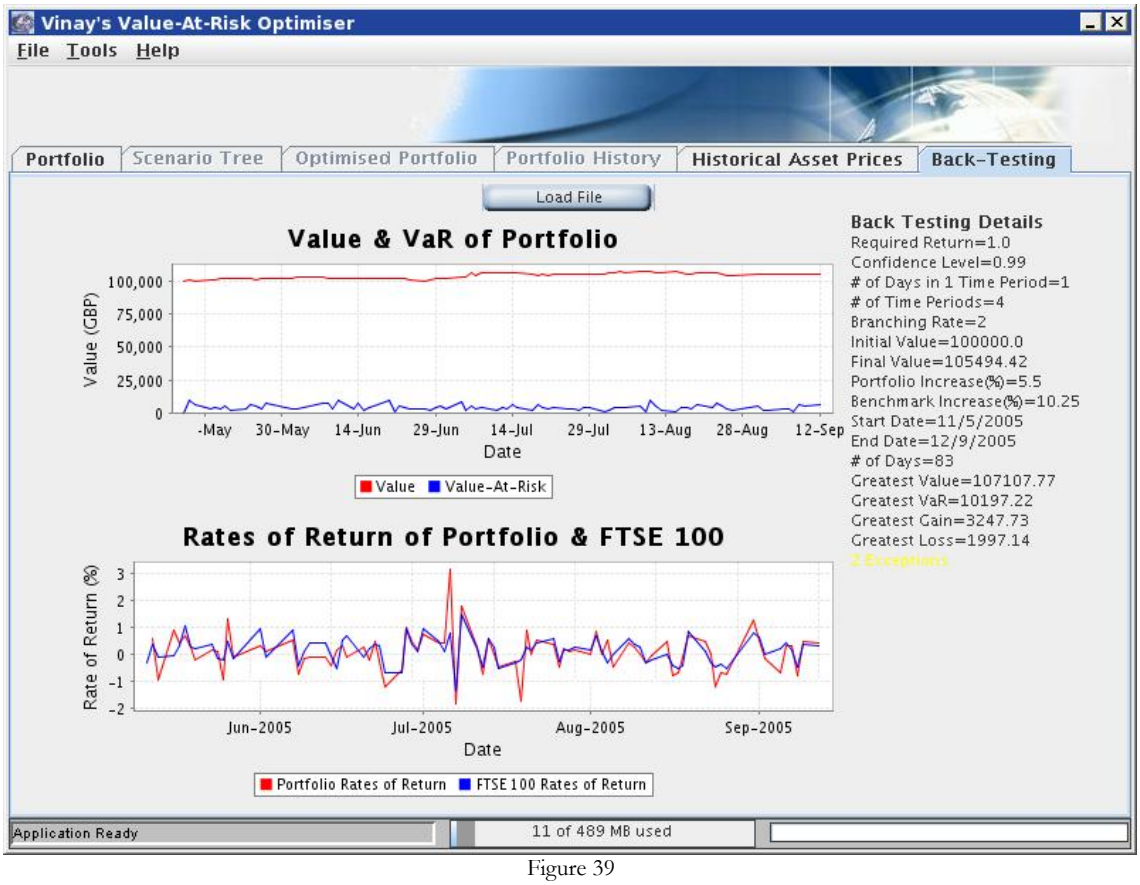


Figure 39

5.6.9 JFreeChart Graphs

As has been evident in this section, I have used graphs to visually display numerical data throughout my Graphical User Interface. To produce the graphs I have used the JFreeChart Class Library. JFreeChart is a flexible open-source, charting library allowing you to integrate charting capabilities into Java applications. JFreeChart can produce many charts including pie, bar and line. It also can generate interactive features of a graph such as tool tips, chart mouse events, and zooming in.

On researching into to chart classes available on the Internet I decided that the JFreeChart module was most appropriate. It is a free Java module with a support website ideal for assisting with the implementation. The package also has a great depth of features and

customisations that I have made use of to make the graphs more suited to the application. JFreeChart uses a factory design pattern to create different styles of graphs, so it has been relatively simple to create the graphs. For this project I have used the JFreeChart Class Library version 0.9.21.

5.7 The Back Tester

As mentioned in Section 5.6.8, I implemented my Back Tester as a standalone Java program. My original intention was to incorporate the back testing facility fully into the Graphical User Interface module. This seemed the obvious choice since the back tester needs to make use of the Scenario Generator and Optimiser modules. However, as is discussed in greater detail in Chapter 7, back testing needs to perform up to 250 optimisations in sequence with no scope of parallelisation. This means that back testing could take a significant amount of time to complete. For this reason, I approached CSG with the view of temporarily getting a project machine from which I could run the back testing.

This request was eventually declined. Speaking to Duncan White from CSG, he advised that it would be better if I implemented the back tester as a separate command-line program. He explained that if I was to implement the back tester as part of the Graphical User Interface, the project machine would need to maintain a logged in connection to the server auk throughout the duration of the back testing process. If the connection was ever severed, for example, due to the project machine accidentally being restarted, the process would be killed. This single point of failure was too great to warrant this kind of implementation.

By implementing the application as a command-line program Duncan explained how I could use the Unix command 'nohup' to execute the application. This command would allow for the application to continue to be executed even if the user was to log off the machine. I agreed that this option would be the best method of implementation.

Changing the method of implementation caused a short delay in my project as it meant that I had to redo part of the implementation of the back tester. Rather than initiating the back testing via the Graphical User Interface, I amended the module so that it now reads in a parameters file of the structure stated in Figure 40, and outputs the results to another file, as described earlier in Figure 38.

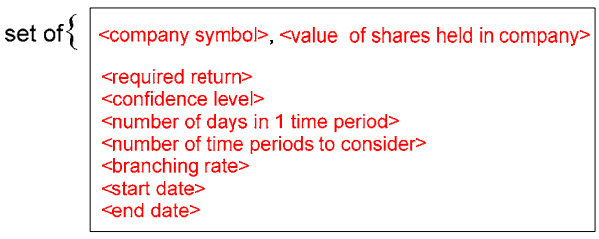


Figure 40

Writing the back tester made me realise the benefits of the separating my application into distinct modules, as I was able to reuse the Scenario Generator and Optimiser modules in my back tester. This reduced the amount of new code I had to write, and hence the time it took me to write the Back Tester.

Chapter 6

Project Management

In order to have a successful project, proper source code and time management has been essential.

6.1 Source Control

I based my project on the Agile Development methodology, and in particular the Extreme Programming standard. Once I had completed my rough Design of the system, I implemented it in short iterations. This entailed that I had a stable release of my application approximately once every 10 days. Each iteration implemented an additional feature compared to the previous one.

Using this method allowed me not only to have a productive implementation process, but also a sustainable one. Having short iterations meant that it was flexible to change. This was a huge advantage as issues I had with the NAG library or back testing, for example, which required change to my implementation could be dealt with relatively simply.

Having to compile a stable version of my application every 10 days also aided my testing. My development was test-driven, so each feature, once implemented, was tested individually and also with the rest of the implemented application. This allowed any errors to be caught early on.

To aid my testing, I also wrote a singleton Logger class that each other class in my application could obtain an instance of. I used the Logger class to output any details of the application at runtime into a file. It was useful referring to this Log file when I was debugging my application as it provided me a method of gaining runtime information from a persistently stored file.

I also used a CVS data repository to store my source code. This was a huge benefit as it provided a means of having a remote backup of my source code, and also gave me the ability to rollback to a prior release if I ever broke one of my tests in a new release.

6.2 Time Management

Since I had to work on my project whilst attending lectures and completing coursework for eight fourth year modules and having exams on them, good time management was vital to my overall success. My overall schedule was to complete all of my background research in the autumn term, allowing me to write my Outsourcing Report over the Christmas holidays and start my application design and implementation in January. I managed to meet this target. I had intended to complete my implementation by the end of the Spring term, however due to the issues with NAG I did not manage to meet this target. This meant that my latter targets were slightly delayed. Writing my final report in parallel with the last part of my implementation aided me to finish everything on time. The Gantt chart in Figure 41 overleaf displays my intended and actual schedule.


ID
Task Name
September
05/09
26/09
21 October
17/10
07/11
28/11
11 December
19/12
09/01
01 February
30/01
20/02
21 March
13/03
03/04
24/04
11 May
15/05
05/06
01 July
26/06
1 Background Research
2 Outsourcing Report
3 Systems Design
4 Database
5 Data Downloader
6 Optimiser
7 Scenario Generator
8 Main Application
9 Exam Revision & Exams
10 Final Testing
11 Final Report
Actual
Predicted

Figure 41

Chapter 7

Testing & Evaluation

Having a financial adviser enables the investor to carry a psychological call option. If the investment decision turns out well, the investor takes the credit, and if it turns out badly, the regret can be lowered by blaming the adviser. - Hersh Shefrin

A successful project will have accurately implemented the multi-stage optimisation problem, as per the core requirements. When evaluating the final system, I have segregated the testing into the following areas:

- Verification of the VaR model
- Validation of the interface and system features
- End user testing

7.1 Verification of the VaR Model

In order to verify my formulated VaR model, I can carry out the following:

- Perform individual optimisations
- Perform back testing
- Generate an efficient frontier

7.1.1 Individual Optimisations

Since I performed some testing whilst carrying out my implementation, I knew that individual optimisations were being performed properly. The only concern I had was with the time being taken for the optimisation library to perform each optimisation. Even though the model I have defined is linear, the optimisations preformed are unusually slow.

This table states how long it takes the NAG optimisation library, on average, to perform portfolio optimisations with the stated scenario tree sizes.

# of time periods	Branching Rate	# of Variables	# of Constraints	Average Time Taken to Optimise (min)
2	2	2,105	712	2.5
2	3	3,910	1,323	8
3	2	4,509	1,524	9
4	2	6,317	2,138	16
3	3	12,028	4,068	35

On communicating the time taken to optimise the constrained problems using the NAG library, my project supervisor suggested that I should try performing a couple of optimisation problems using another optimisation library. I decided to use the GLPK optimisation library. The best way for me to perform this test would have been to generate an optimisation problem from my application, feed it into the GLPK optimiser and compare the times taken to solve the problem. However, the GLPK optimiser requires the data to be inputted in the form of an MPS file. Since the NAG optimiser uses tab delimited files, I needed to convert the format of the outputted file.

Unfortunately, due to time constraints this was not possible. I did, however, manage to convert an MPS file into a tab delimited file which I fed into the NAG library. The optimisation problem contained 930 constraints and 3523 variables. The GLPK optimiser managed to solve the problem in 0.5 seconds, compared to the NAG library taking 20 seconds.

From this result, it is evident that the NAG optimisation library is contributing to the length of time it takes to optimise a portfolio. This could be because, as explained in Section 5.5, there are an infinite number of solutions to the optimisation problem, so the optimisation algorithm implemented by NAG could have a slow convergence. It could also be the case that the NAG library is possibly not suited to solving large optimisation problems of this nature; but further investigation needs to be carried out to confirm this.

7.1.2 Back Testing

Through back testing, I have tested my VaR model in the same way as is stated by the Basle Standard[1] for industrial VaR models. The Basle Standard requires back-testing – testing a strategy on a large historical database to evaluate the model's accuracy before risking any real money. It involves using the created model to control the VaR of a portfolio, and using the historical data to observe the actual VaR of the portfolio. The model will only be accepted if the number of observations falling outside VaR are inline with the stated confidence.

To formally carry out the testing the failure rate will be assessed. As stated in [7], the Basle standard states that the verification procedure consists of recording daily exceptions (the actual loss exceeds the stated VaR) for 1 year, which assumed to be 250 working days long, with a confidence level of 99%. With a 99% confidence level, one would expect there to be 2.5 exceptions, however, the Basle Committee has decided on the following:

Zone	Number of Exceptions
Green	0-4
Yellow	5-9
Red	10+

The model is deemed to be accurate if the results show that the model is in the green zone. If the model lies in the yellow zone, the committee suggests adding a safety multiplier to the model stated VaR of the portfolio to slightly increase it. Further monitoring of the model for an extended period of time is then required. If the model lies within the red zone, the model is deemed to be inaccurate and needs to be revised.

The following steps are required to perform the back testing:
Step 0 – Read in parameters over which back testing is required to be performed
Step 1 – Optimise portfolio with the current parameters
Step 2 – Move date one day forward and obtain actual prices of assets
Step 3 – Calculate and record difference between estimated VaR and actual profit/loss of portfolio
Step 4 – Go to *Step 1*. Once 250 days have passed, stop the algorithm.

[1] A standard drawn up stating international capital adequacy standards for banks. Set up by the Basle Committee; a committee set up by the Bank for International Settlements and based in Basle.

In this same manner, I have tested my implemented VaR model. The model was given the following parameters:

- Initial wealth = £100,000
- Initial weightings = Equal weighting of 10 randomly selected assets
  - AL.L, AAL.L, XTA.L, GSK.L, RIO.L, NXT.L, MKS.L, CBRY.L, BSY.L, BP.L
- 1 time period = 1 day
- Number of time periods to consider in model = 4
- Required return = 0.1623% per day (50% per year)
- Transaction cost = 1.0% for all assets
- Start date = 28/04/2005
- End date = 24/04/2006

The results I obtained are given in Figure 42.

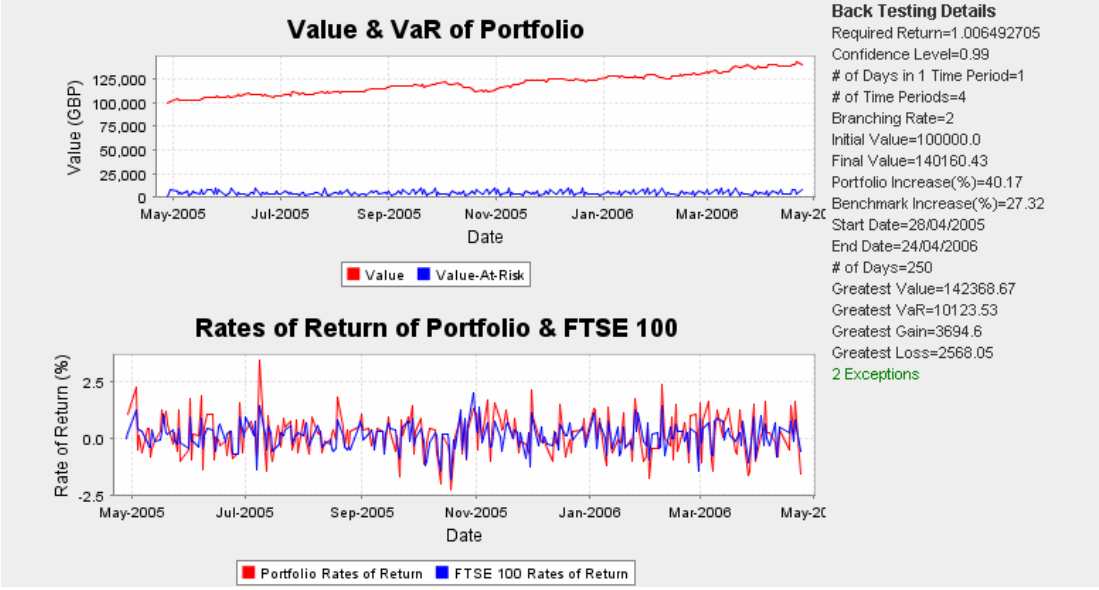


Figure 42

As can be seen from Figure 42, the optimised portfolio only had two exceptions during the course of the year when the actual loss of the portfolio was greater than the value of VaR calculated by the model. This places the model in the green zone. In addition, the portfolio outperformed the benchmark by approximately 13%. This is evidently a good set of results, however to further test the validity of the model, I performed numerous back testing simulations with different:

- Required return
- Confidence level
- Number of days in one time period
- Number of time periods / branching rate
- Start and end dates

The following table details the back testing simulations I ran.

Required Return (per time period – column 3)	Confidence Level	# of Days in 1 Time Period	# of Time Periods	Branching Rate	Start Date	End Date	Benchmark (FTSE100) Relative Gain	# of Exceptions	Greatest Loss	Greatest VaR
1.005	0.95	1	2	3	21/12/2005	15/02/2006	+0.17%	2	1,237.42	3,459.76
1.01	0.99	5	4	2	14/06/2005	20/01/2006	+9.78%	1	4,578.45	12,455.19
1.02	0.95	1	3	2	03/05/2005	30/01/2006	+2.65%	3	6,456.13	8,345.27
1.02	0.99	1	2	3	12/12/2005	10/04/2006	-3.65%	2	2,947.86	5,214.98
1.03	0.95	1	2	2	13/06/2005	20/01/2006	-12.87%	4	4,561.23	7,845.46
1.04	0.99	1	4	2	08/07/2005	12/04/2006	+4.54%	1	5,672.89	8,561.23
1.05	0.99	5	4	2	11/04/2006	09/06/2006	-17.12%	0	8,957.97	17,777.23

As can been seen from the table above, not all of the portfolios performed better than the benchmark. It is evident that setting the required return to be high in most cases will cause the portfolio to lose money. The optimal required return seems to be approximately in the range of 0.15-0.25% per day (equivalent to approximately 50% return per year). This return value has not been met by the optimised portfolio, however, it does consistently outperform the FTSE 100 benchmark.

It can also be observed that simulations which had the smallest scenario trees had a greater number of exceptions where the actual loss was greater than the predicted VaR. This demonstrates that having a larger scenario tree will indeed make the model more accurate.

Figure 43 displays the results of a poor performing back test. I used this back test to investigate what would happen if I had a high required return. In order to get a higher return, the portfolio has to hold riskier assets, which is why the lower graph shows sporadic rates of returns. Even though the portfolio ended up making a large loss, it can be seen that the application was able to predict the VaR of the portfolio accurately, with there being zero exceptions.

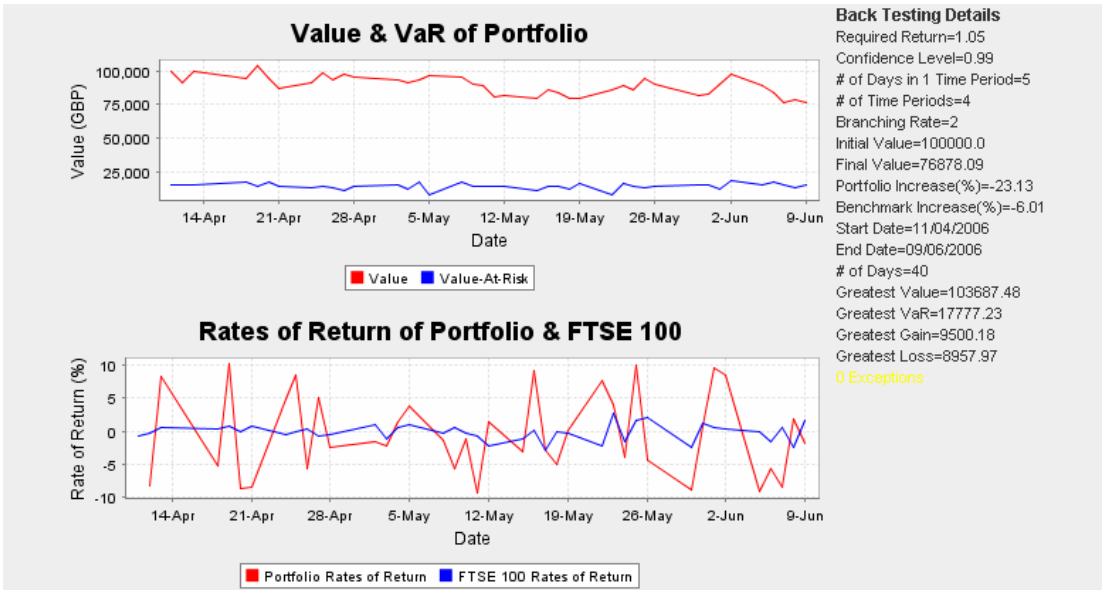


Figure 43

I have also used back testing to analyse the effect of varying transaction costs. Using the same optimisation parameters as stated at the top of page 53, apart from the start and end dates, which were 11/07/2005 and 12/12/2005 respectively, I varied the transaction costs as per the below table.

Transaction Cost	Benchmark (FTSE100) Relative Gain	# of Exceptions
0.5%	+7.45%	1
1%	+6.78%	2
2%	+4.82%	5
5%	-1.61%	3
15%	-19.17%	9

As can be seen from the table, increasing transaction costs increases the amount of friction within the model. This friction makes it sub-optimal to buy and sell assets regularly, and hence in high transaction cost situations the returns of the portfolio depend heavily on the initially held assets and their rates of return over time. Without being able to change portfolio holdings, the returns of the portfolio diminish.

7.1.3 Efficient Frontier

Another method of verifying the VaR model is to use it to plot the points of an efficient frontier. One problem I found was that, since I have not implemented a model to calculate the minimum possible VaR and maximum possible VaR values, it is difficult to define the range for the efficient frontier over which the points should be plotted. This means that, as shown in Figure 44, normally the efficient frontier which is generated ends up looking like an increasing line with a slight curve.

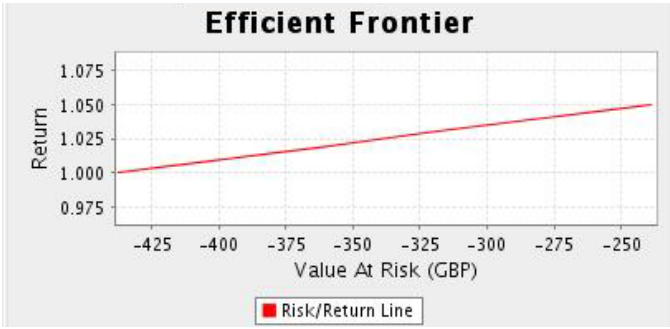


Figure 44

Through tweaking of the range of the efficient frontier for one of my portfolios, I was able to get my application to produce the efficient frontier shown in Figure 46. It is of a similar shape to the efficient frontier stated by Uryasev in [12] so I will deem my application to be able to successfully generate efficient frontiers.

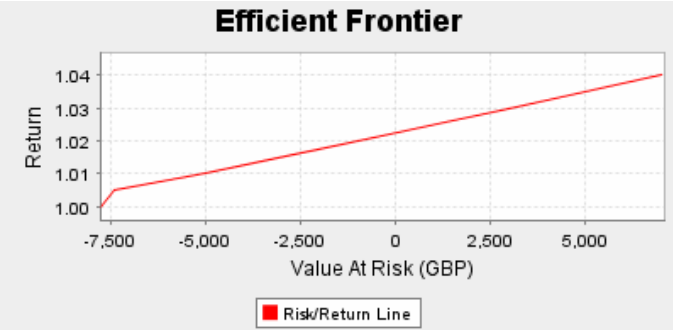


Figure 45

7.2 Validation of the Interface and System Features

In addition to the model, I have evaluated the various features of the implemented system surrounding the model. This is firstly done by ensuring that my interface meets Nielson's ten usability heuristics stated in [16]:

Visibility of system status
This has been accomplished through the status bar at the bottom of the application window. The status bar has a message panel that textually informs the user of the current task the application is performing. The indeterminate progress bar is also another component the user can use to see what state the application is currently in; whether it is currently busy or not.

Match between system and the real world
I have ensured that the terminology used within the application matches that used in the real-world. Following real-world conventions and laying out the application in a logical manner means that there is not a steep learning curve associated with my application.

User control and freedom
The user has full control over the application with the ability to run an optimisation, view the history of a portfolio, view the historical prices of assets or load a back testing output file at will. The threading of the Scenario Generator and Optimiser allows the user to carry on using other features of the application whilst these two processes are occurring in the background.

Consistency and standards
I have kept consistency throughout the interface of the application in terms of the layout of the panels and the colour scheme used. I have also been consistent with the terminology used throughout the application.

Error Prevention
All of the text boxes which require the user to input any value have input verification enabled on them. The file parsers which read in external files also have been implemented to handle ill-formatted files. This means that the user is informed of any invalidity before any process is performed on corrupted data.

Recognition Rather Than Recall
Using real-world terminology in my interface design, the user is able to use recognition rather than recall to perform a task or navigate around the application.

Flexibility and efficiency of use
The application allows the user to input the portfolio composition and optimisation via the text boxes on the interface. However, for regular users, the application also offers the facility for these parameters to be imported from a file. The user also has the option of exporting the results of the optimisation as (.csv) or a (.var) file. This means that the application is suited to fit into the workflow of different types of users.

Aesthetic and minimalist design
The design of the interface has been kept simple and modular so that the user finds it easy to use and navigate through.

Help users recognise, diagnose and recover from errors
The status bar message panel provides a quick means of passing simple messages to the user. For more critical messages a dialog box appears informing the user of the event that has just occurred. Dialog boxes have been implemented to provide high level details of the error and remedial steps.

Help and documentation
A user guide as has been written and is available from my project website. The application provides a link to this website in the "About" window. A copy of the user guide has been included in Appendix Section C.

Consistent with the "V" Model, as stated in [14], each of the goals in Figure 8 have also been tested. Working through the complete process of optimising a portfolio; from inputting the initial portfolio composition to obtaining the optimised weightings. I feel that this task is simple for a user to perform, and the user interface is intuitive to use with there being minimal context switches. In doing this, the following aspects were tested:

-Check the methods of inputting parameters into the system through the front-end – As stated in the specification, the front-end needs to capture the input of parameters for the model from the user. I have confirmed that the input of parameters from the front-end as well as through the import of a (.csv) and (.var) file works. Using valid, extreme and invalid values I exhaustively checked the handling of inputs.

-Check the acquisition of asset price data – The system is required to automatically obtain daily price data. I verified this is occurring daily and that the data is being stored in a table of historical asset prices.

-Check generation of scenarios – The system is required to use *Cluster* to generate multi-period scenarios. I verified that the historical data is being used to produce valid future scenarios which are used for the portfolio optimisation.

-Check the displaying of the scenario tree – I have verified that once the scenario tree has been generated, it is validly displayed on the screen for the user.

-Check the displaying of the optimised portfolio – Once the inputs have been obtained, and portfolio optimised, the optimal portfolio weightings are displayed to the user. I have also verified that the weightings are able to be exported to a (.csv) or a (.var) file.

-Check the efficient frontier generation – If the efficient frontier is required, given an initial portfolio, the application is able to generate the efficient frontier by varying the required returns parameter. Generally, the outputted graph is of the correct shape, though there are a few anomalous cases where there are spikes in the curve. This needs further investigation.

The extensions also need to be checked. The back-testing goal was checked by performing the back-testing stated in Section 7.1. As can be seen, the system was able to successfully carry out the back-testing and display the results.

To check that the system allows historical data to be viewed, I tried to view the historical prices of randomly chosen assets. The system performed this in an easily understandable manner.

I also checked the facility for the persistent storage of portfolios. To do this, I created a new portfolio, optimised and saved it. I then repeatedly loaded, optimised and saved the portfolio on successive days. The system successfully saved the portfolio after repeating the process five times and displayed a graph of the history of the portfolio, so I deemed the system as having successfully completed this goal.

7.3 End User Testing: Demonstration to Goldman Sachs

One of my key project objectives was to demonstrate how my application could fit into the daily workflow of a professional Portfolio Manager. In order to gain the opinions about my application from a real end user I approached Rezwan Ud-Din, an Executive Manager from Goldman Sachs. Rezwan manages the Equities Asset Management Technology desk at the London office.

Overall, Rezwan commented on how commendable this project is. A copy of the feedback email sent to me can be found in the Appendix Section C.

I demonstrated my application to Rezwan with a view of explaining to him

- How a portfolio is optimised by the application.
- The methods in which portfolios can be imported into and exported out of the application.
- Features which demonstrate how the application could fit into the daily workflow of a Portfolio Manager.

Rezwan felt that the optimisation procedure implemented by the application would be easy for a Portfolio Manager to perform. He also liked the fact that the application had the facility for the user to investigate what future prices had been predicted at each node of the scenario tree, and hence what the optimal transactions at each node would be. This would give the user confidence in that the recommended transactions to perform at the current time period would indeed be optimal.

Rezwan explained to me how Portfolio Managers at Goldman Sachs use several pieces of analysis to make their investment decision, and how my application could fit alongside their current tools. VaR is not currently being used by the Portfolio Managers, but this kind of tool could be an option for them to use.

Rezwan did comment on the fact that in the current implementation the optimisation procedure takes a significant amount of time. For the application to be useful to Portfolio Managers, it would need to perform optimisations within 15 seconds, as they would want to repetitively tweak the optimisation parameters and re-run the optimisation procedure.

7.4 Summary

From my testing I have learnt the following facts:

- When performing extended simulations, more profitable investment decisions will be made and more accurate values for VaR will be calculated if a larger scenario tree is used and the required return is in the region 0.15-0.25% per day
- As the required return value increases, riskier assets are held by the portfolio, increasing the chance of making a loss.
- If extreme parameters are used for the optimisation process, there is a greater chance that the model will fall into the yellow or red zones, otherwise it will lie in the green zone.
- Further investigation is required to speed up the optimisation process
- High transaction costs leads to a high amount of friction in the system, increasing the chance of making a loss
- My application can be used by professional portfolio managers to aid them in making their investment decisions

Chapter 8

Conclusion

8.1 Remarks

During the course of my project I have developed a new optimisation model and an application that implements it. Given that no model existed implementing multi-period portfolio optimisation using VaR when I started my project, I feel that I have risen to the challenge and produced a sound model.

The automated back testing facility which I have implemented was a huge aid in testing my application. It has demonstrated that my portfolio optimiser is able to produce lucrative returns. Within the boundary of normal optimisation parameters, the model is also able to reliably predict the VaR of the portfolio.

Rezwan's positive response to my application shows how the core and non-core features I have implemented successfully demonstrate my application to be simple for an end user to use and could also fit into the work flow of a professional portfolio manager

As mentioned in my Implementation Section, I did encounter a few delays, which did affect the implemented application. Given extra time, I would have liked to use JNI as the interface between my application and the optimisation library. I would have also liked to perform a greater number of back testing simulations to gain more evidence on the accuracy of my model.

8.2 Future Work

There exists the scope for future work to be carried out on both my VaR model and my application. As mentioned in my Background Research, my model estimates the value of VaR for the portfolio by considering only the leaves of the scenario tree. Further investigation is required to allow the model to incorporate the VaR at the intermediary nodes in the final VaR value. This will not be as simple as averaging the VaR over all of the nodes since the VaR of a node should be weighted depending on its level within the tree. A node lower down the tree has the potential for a greater profit, and hence a greater loss, when compared to the value of the portfolio at the initial node.

Further investigation also needs to be carried out on the choice of optimisation library used. As stated in my Evaluation, the GLPK optimisation library was able to solve the constrained problem significantly faster than the NAG library. Although, I did not have time to run further tests, changing the optimisation library the application uses could produce sizeable improvements in the time taken to optimise a portfolio.

As suggested by Rezwan, the application could be further developed by categorising assets into sectors, and allowing users to investment money in aggregated sectors rather than individual companies. I could also add the facility for users to state if they did not want to invest of their money in certain sectors, for example, in those companies which produce alcohol.

References

[1] Artzner, P., Delbaen F., Eber, J.M., and D. Heath (1999): Coherent Measures of Risk, *Mathematical Finance*, 9, 203-228

[2] J.V. Ashby: A comparison of C, Fortran and Java for Numerical Computation

[3] C. Butler: Mastering Value at Risk, FT Prentice Hall, 1st edition, 1999

[4] N. Gulpinar, B. Rustem, R. Settergren (2004): Multistage Mean-Variance Portfolio Analysis with Transaction Costs

[5] N. Gulpinar, B. Rustem, R. Settergren (2004): Simulation and Optimisation Approaches to Scenario Tree Generation, *Journal of Economic Dynamics and Control 28*, 1291-1315

[6] G.A. Holton: Value-at-Risk: Theory & Practice, Academic Press, 2003

[7] P. Jorion: Value at Risk: The New Benchmark for Managing Financial Risk, McGraw-Hill Companies, 2nd edition, 2000

[8] J. Palmquist, S. Uryasev, P. Krokhmal (2002): Portfolio Optimization With Conditional Value-At-Risk Objective And Constraints, *The Journal of Risk*, Vol. 4, No. 2

[9] R.T. Rockafellar, S. Uryasev (2002): Conditional Value-At-Risk For General Loss Distributions, *Journal of Baking & Finance 26*, 1443-1471

[10] R.T. Rockafellar, S. Uryasev (2000): Optimization of Conditional Value-at-Risk, *The Journal of Risk*, Vol. 2, No. 3, 21-41

[11] S. Uryasev: Lecture 7 from course ESI 6912: Optimization Using CVaR-Algorithms and Applications

[12] S. Uryasev: Lecture Notes: Conditional Value-at-Risk (CVaR): Algorithms and Applications

[13] Wiener, Z. (1997): Introduction to VaR (Value-At-Risk), Risk Management and Regulation in Banking, Jerusalem, Israel

[14] http://www.coleyconsulting.co.uk/testtype.htm

[15] http://www.golriamundi.org

[16] Nielson, J.: Usability Heurtistics, http://www.useit.com/papers/heuristic/heuristic_list.html

Appendix

Section A: Stored Procedures

List of the stored procedures in my database which return data and their relational algebra representations:

- `sp_get_all_assets () returns setof varchar`
  - $\pi_{company\_symbol}(symbol\_table)$
  - Returns a set of all of the symbols of the assets within the database
- `sp_get_all_assets_details() returns setof (varchar, varchar)`
  - $\pi_{(company\_symbol,company\_name)}(symbol\_table)$
  - Returns a set of all of the symbols of the assets within the database and their names
- `sp_get_asset_price_history(varchar) setof record`
  - $\pi_{(trade\_date,price)}(\sigma_{symbol=x}(price\_history))$
  - Returns a set of all of the historical prices for asset x
- `sp_get_covar(varchar, varchar, integer) returns covariance`
  - $\sigma_{symbolX=x,symbolY=y,period=z}(\mathrm{cov}\,ariance)$
  - Returns a record of the covariance table with the required symbolX, symbolY and period
- `sp_get_growth(varchar, integer) returns growth`
  - $\sigma_{symbol=x,period=y}(growth)$
  - Returns a record of the growth table with the required symbol and period
- `sp_check_trans() returns bigint`
  - $\Im_{COUNT}(constra\,\mathrm{int}\,s)$
  - Returns the number of rows in the constraints table
- `sp_get_all_trans() returns setof record`
  - $\pi_{(trans\_\cos t,symbol,company\_name)}(\sigma_{c.symbol=s.company\_symbol}(constra\,\mathrm{int}\,s\times symbol\_table))$
  - Returns the transaction cost associated with each asset in the database

Section B: Example Auto-Generated Optimisation Output File

An example of an auto-generated file from my FORTRAN script which runs the optimisation procedure.

```
E04MFF Program Results

 *** E04MFF

 Parameters
 ----------

 Problem type...........        LP

 Linear constraints.....      1224       Feasibility tolerance..  1.05E-08
 Variables..............      3609       Optimality tolerance...  1.05E-08

 Infinite bound size....  1.00E+20       COLD start.............
 Infinite step size.....  1.00E+20       EPS (machine precision)  1.11E-16

 Check frequency........        50       Expand frequency.......         5
 Minimum sum of infeas..        NO       Crash tolerance........  1.00E-02

 Print level............        10       Iteration limit........     24165
 Monitoring file........        -1

 Workspace provided is     IWORK(   10000),  WORK( 4000000).
 To solve problem we need  IWORK(    7221),  WORK( 3032633).


  Itn     Step Ninf Sinf/Objective  Norm Gz
    0  0.0E+00   23   1.950371E+01  0.0E+00
    1  1.8E-08   23   1.950371E+01  0.0E+00
    2  2.2E-08   23   1.950371E+01  0.0E+00
<A row exists for each iteration carried out in solving the optimisation problem>
   70  4.3E-01    0  -2.626854E-02  0.0E+00


 Varbl State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

 V   1    FR  -2.626854E-02      None          None
 V   2    FR   0.200000           .           None           .      0.2000
 V   3  D FR  -3.307347E-10       .           None           .     -3.3073E-10
<A row exists for each variable optimisation problem>
 V3609    LL    0.00000            .           None       9.890         .


 L Con State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

 L   1    EQ  -0.200000     -0.200000     -0.200000     -0.5057     -1.2212E-15
 L   2    EQ  -0.200000     -0.200000     -0.200000     -0.5159      8.3267E-17
 L   3    EQ  -0.200000     -0.200000     -0.200000     -0.5057      6.1062E-16
<A row exists for each constraint in the optimisation problem>
 L1224    FR    1.14448       1.00000          None           .      0.1445

 Exit E04MFF - Weak LP solution.

 Final LP objective value =  -0.2626854E-01

 Exit from LP problem after    71 iterations.

 ** Weak LP solution.
 ** ABNORMAL EXIT from NAG Library routine E04MFF: IFAIL =     1
 ** NAG soft failure - control returned
```

Section C: Feedback Email From Rezwan Ud-Din

The feedback email received from Rezwan regarding my portfolio optimisation application can be found overleaf.

Section D: User Guide

This User Guide has the following structure:

- Prerequisites
- Starting the Application
- Application Window Layout
- Walk-through of the process of performing a portfolio optimisation
-

Prerequisites

For the application to run it requires access to the NAG library. This limits it to being run only within the Department of Computing. In order to set up access to the NAG library, the following commands need to be run on the Unix Shell:

```
ssh auk
setenv LM_LICENSE_FILE /opt/NAG/license.lic
```

Starting the Application

Only once you have run the above commands will the application be able to run without errors. The following command will start the application:

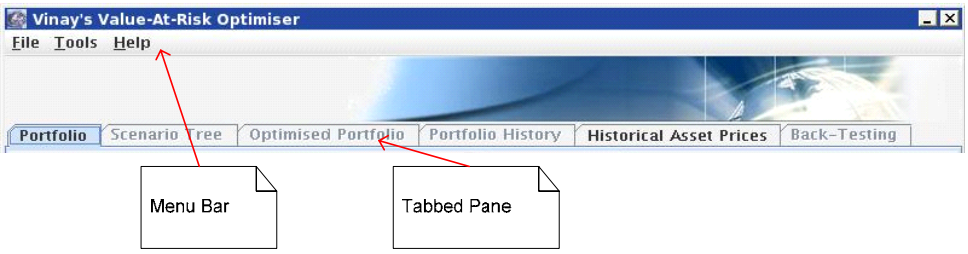
```
java –jar VAR_Portfolio_Optimiser.jar
```

By default the Java Virtual Machine will allocate the application with 64Mb of memory. If the memory of your machine allows, you can run the application with more memory using the following command:

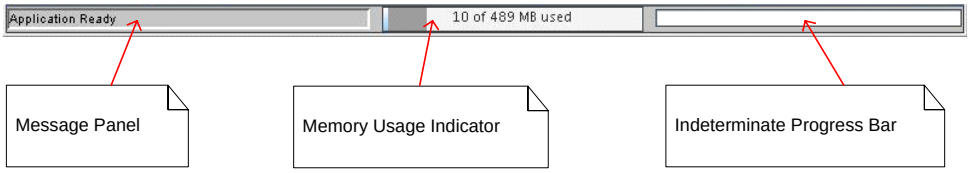
```
java –jar –Xms50m –Xmx256m VAR_Portfolio_Optimiser.jar
```

Application Window Layout

The main application window has the following key components:



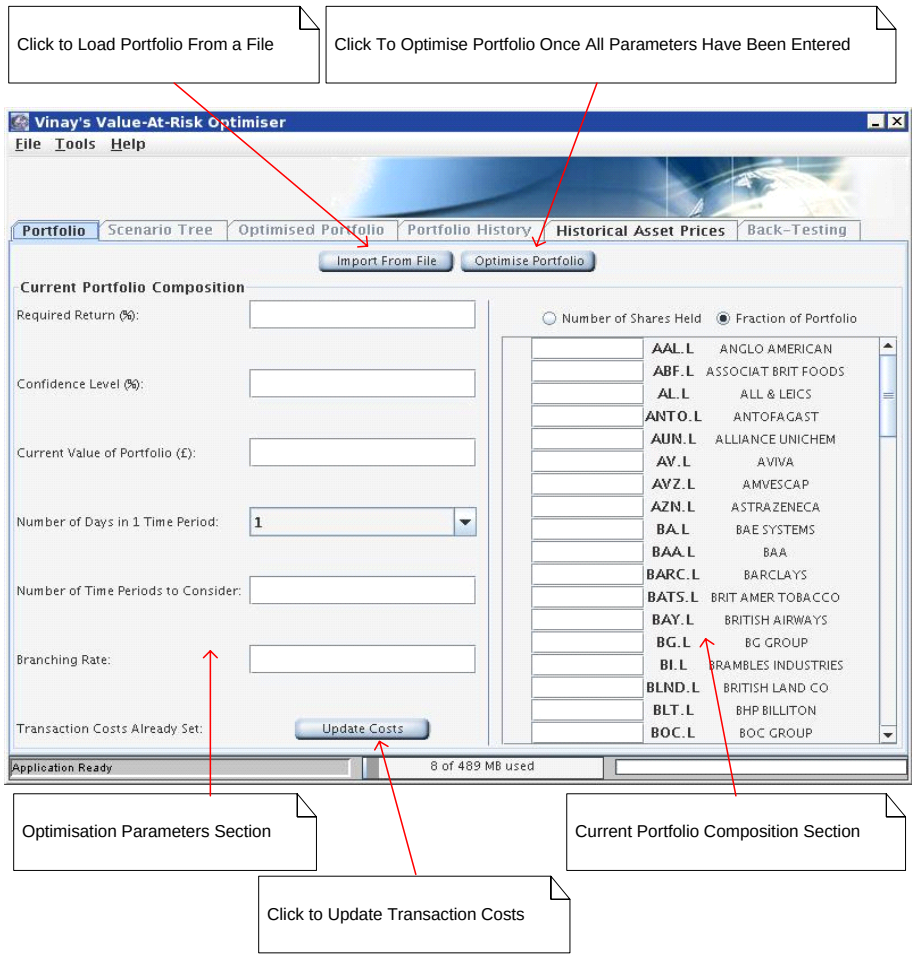
The menu bar at the top of the provides you with the means of exiting the application, performing a Garbage Collection to destroy any unused objects in memory and to view the About window, which also provides a link to the project website. The tabbed pane provides you with the main source of navigability within the application. It controls which panels you able to view during your use of the application.



The status bar at the bottom of the window is a very useful component giving an overview of the current state of the application. The message panel on the left textually informs you of the current task the application is performing. The memory usage indicator informs you how much memory the application is currently using. The indeterminate progress bar is used to ensure you that the application is busy and has not frozen whilst performing optimisations which may be time consuming.

A Guide Through the Process of Performing a Portfolio Optimisation

After loading up the application, you will be presented with the below screen. This initial panel is used to enter the current portfolio composition and the parameters with which you would like to optimise the portfolio. The left half of the panel requires you to enter the parameters you would like to optimise the portfolio with. The right side requires you to enter the quantity of the current assets you hold. You may do this by either entering the number of shares you own for each stock, or as a fraction of the total wealth you have in your portfolio. Please note, that if you specify the number of shares you own for each asset, you will not need to enter a value for the current wealth of the portfolio, as this will be auto-calculated by the application using the latest asset prices.



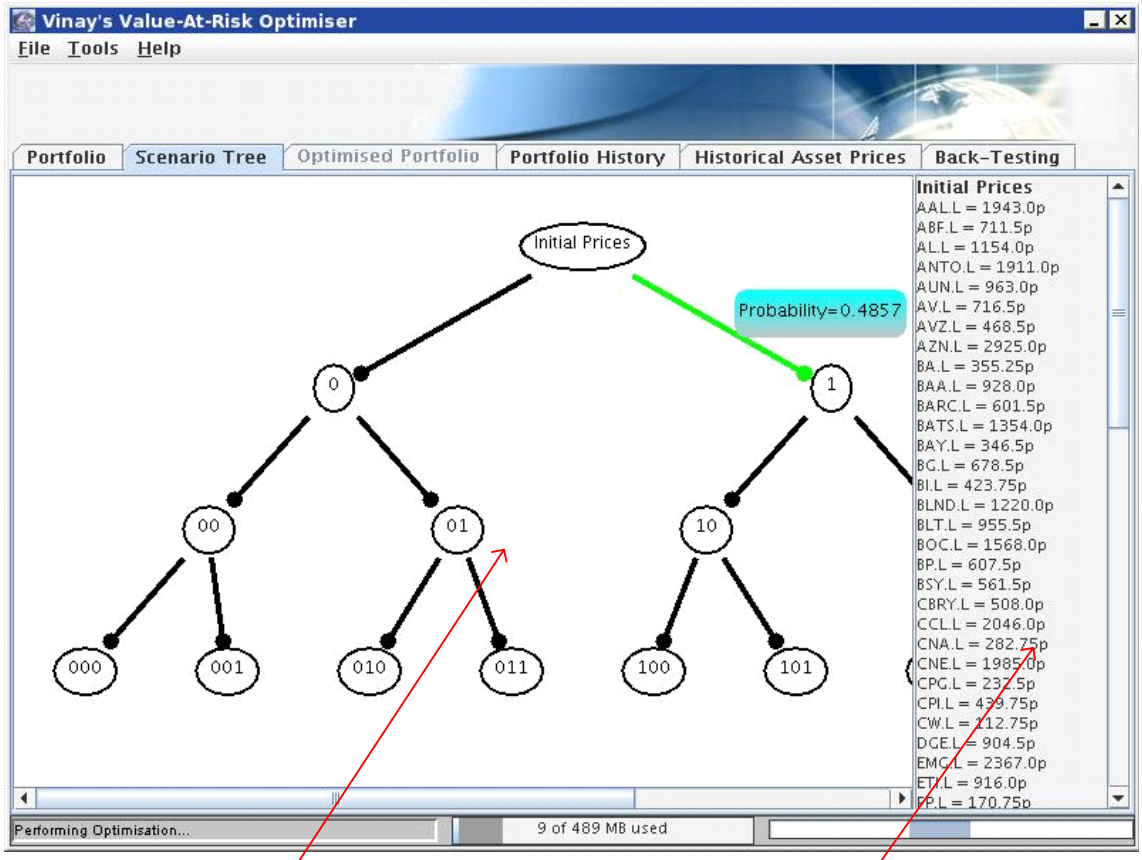
Instead of entering all of the details manually, you may also import them from a file. You can import a (.csv) file which contains a set of the current assets you hold in you portfolio, of the format: *<company symbol>,<number of shares held in company>*. You may also import the application's own (.var) which, in addition to the current portfolio composition details, will contain a history of the portfolio and a record of the parameters you used in your previous optimisation.

By clicking on the "Update Costs" button, you have the opportunity to update the transaction costs which exist for each of the assets in the system.

Once you have entered all of the parameters, you may click on "Optimise Portfolio" to begin the optimisation.

The first step in portfolio optimisation is to create a scenario tree. Once this has been created by the application, you will be able to view it, even while the actual optimisation procedure is occurring. By selecting the "Scenario Tree" tab from the tabbed pane, you will be presented with the below window.

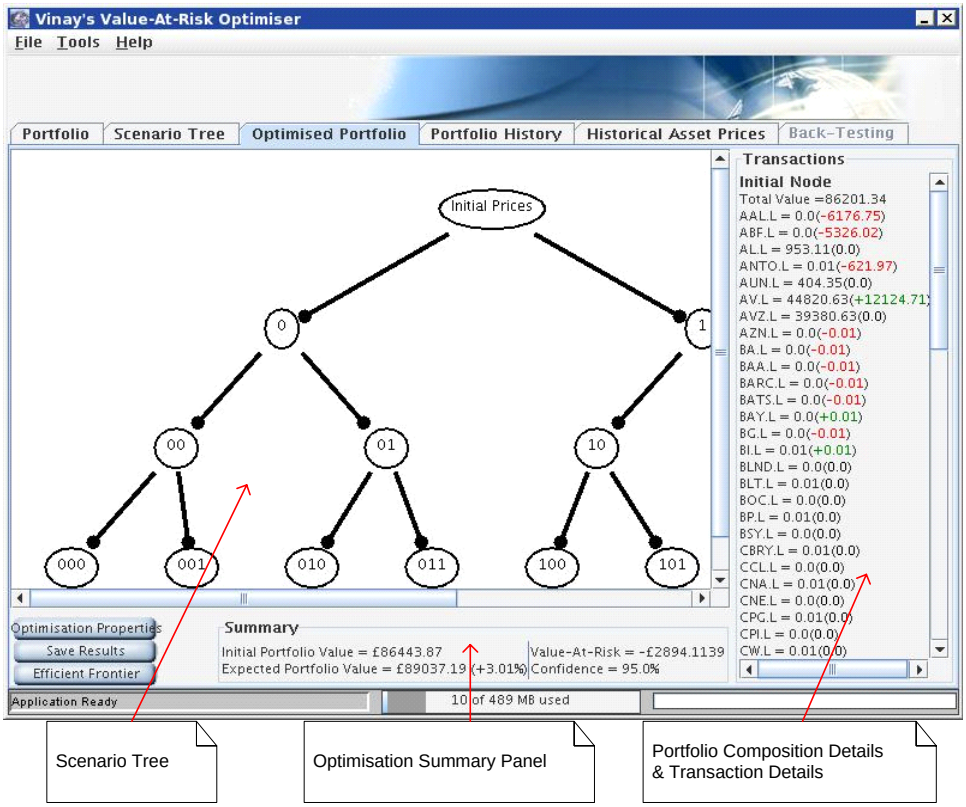
By clicking on any node in the visualised Scenario Tree, you will be able to view the predicted prices of assets at that node on the details panel on the right of the window. By holding your mouse over any of the arcs of tree, you will be able to see the predicted probability associated with that arc.



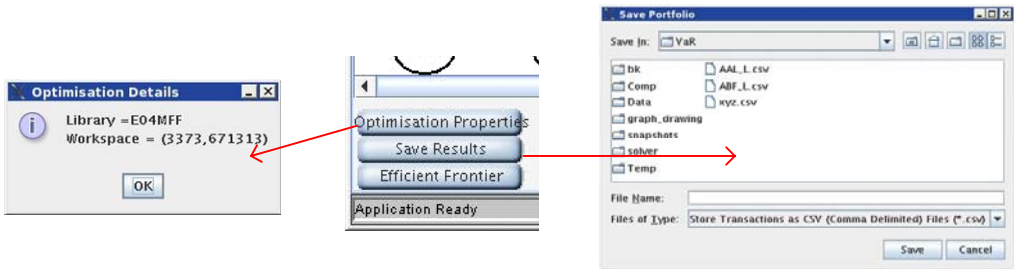
Visualisation of the Scenario Tree

Prices of Assets At Selected Node

Once the optimisation has been completed, you will be to click on the "Optimised Portfolio" tab to be presented with the below window.



Once again, using the scenario tree as your navigation tool, you can look at the portfolio details at each node of the tree. The details, shown on the right, signify the value of the shares you hold of each asset at the selected node. The numbers in the brackets signify the value of the shares the optimal portfolio has bought/sold at that node. Please note that the transactions stated take into consideration transaction costs. The summary panel, at the bottom of the window, gives you an overview of the optimised portfolio.

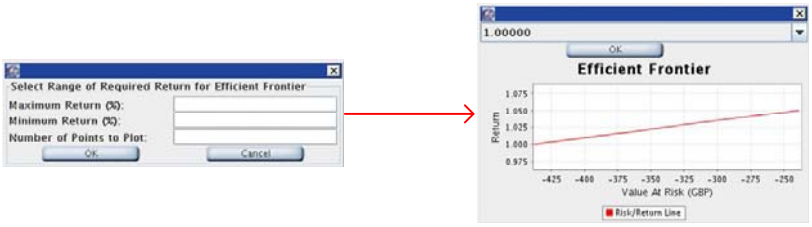


As shown above, by clicking on the "Optimisation Properties" button, you may view the technical details of the optimisation library. By clicking on the "Save Results" button, you may export the optimisation results as:

- A (.jpg) image of the scenario tree
- A (.csv) file stating which transactions are required at the initial time period for the portfolio to conform to the calculated optimal wealth distribution.
- The application's own (.var) file, which will store the portfolio history and parameters used, which you may import again at a later date.

Efficient Frontier

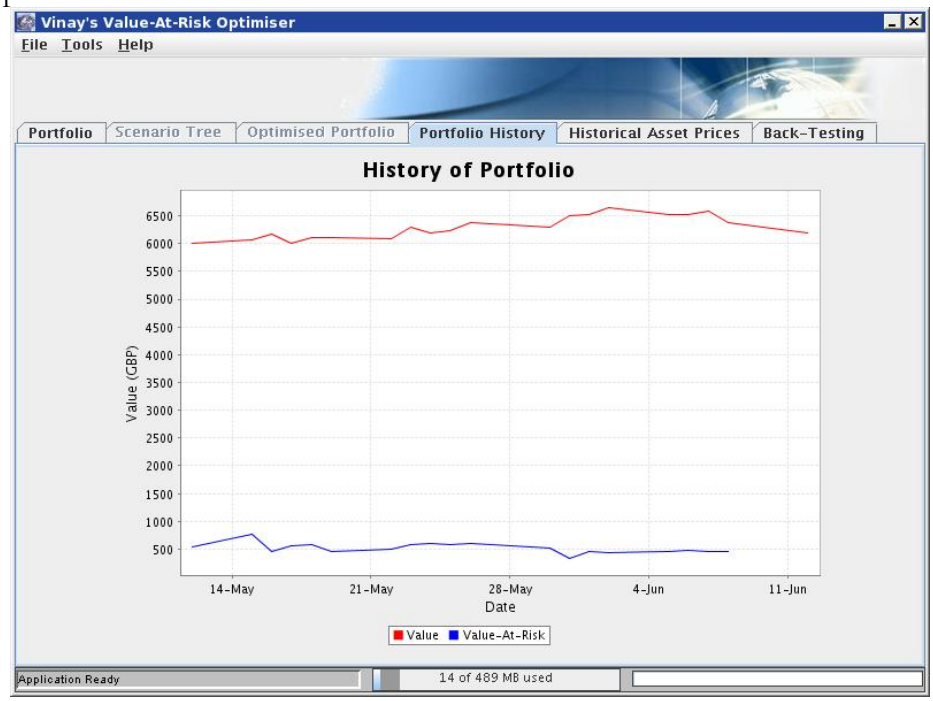
Given that you have optimised a portfolio, the application can also produce an efficient frontier, detailing a curve of varying returns against the risk required to achieve those returns. By clicking on the "Efficient Frontier" button on the "Optimised Portfolio" panel, you will be presented with a window in which you can enter the range of the frontier and the number of points you would like to plot. Once calculated, the application will display the calculated efficient frontier, as per below. Please be patient whilst waiting for the efficient frontier to be plotted, as the application will have to perform separate optimisations numbering the number of data point you would like plotted.



Using the drop down list at the top of the window hosting the efficient frontier, you may select to view any of the generated portfolios. The portfolio details will be shown on an updated version of the "Optimised Portfolio" panel in the main application window. The graph provides you with the facility of zooming in or out, printing or exporting the efficient frontier.

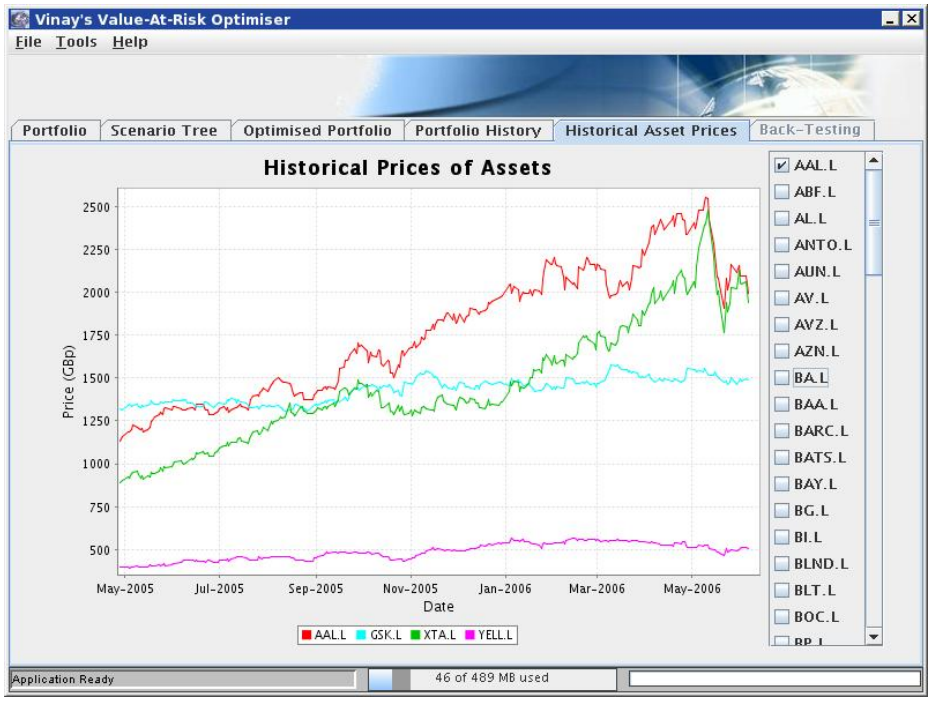
Portfolio History

If you have imported a (.var) file into the application, you will be able to view the portfolio's history using the "Portfolio History" panel. Using the tabbed pane to view this panel, you will be presented with a graph displaying the value and Value-At-Risk of the portfolio over time, as per below.



Historical Asset Prices

This application also provides the facility to view the historical prices of assets. By clicking on the "Historical Asset Prices" tabbed pane, you can check the symbols of the assets you would like to view the historical prices for. Un-checking an asset will remove its historical price line from the graph. This graph also provides you with the facility of zooming in or out, printing or exporting the historical prices.



Back Testing

To provide a method of automated model verification, this application implements back testing. The back testing process is two fold. Firstly, you will need to run a command line program to perform the actual testing. The program takes in a file with the following structure:

set of { <company symbol>, <value of shares held in company>

<required return>
<confidence level>
<number of days in 1 time period>
<number of time periods to consider>
<branching rate>
<start date>
<end date>

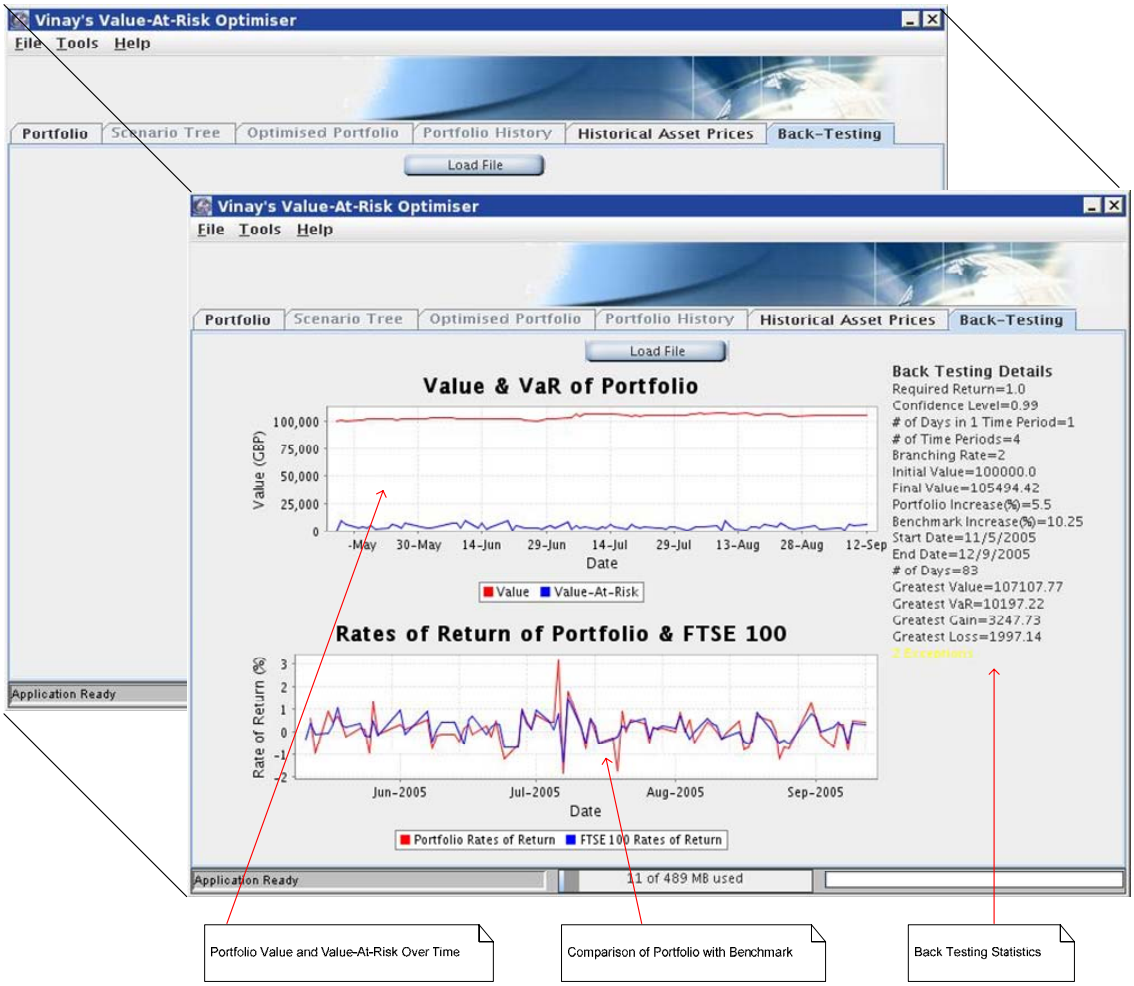
In order to execute the program the command below will need to be run. Once again, you will need to have the environment set up as described in the Prerequisites section of this User Guide.

```
java -jar VAR_Portfolio_Optimiser_Back_Tester.jar <input filename>
```

Depending on the number of days over which the back testing has to be performed, this may take a long period of time. In order to allow the process to run to completion even if you log off the machine, you may use the following command:

```
nohup java -jar VAR_Portfolio_Optimiser_Back_Tester.jar <input filename> &
```

Once the back testing program has finished a file BTesting.out will be produced. This can be loaded into the main application for you to view the results by going to the "Back Testing" panel in the main application and clicking on the "Load File" button. Once the file has been loaded, the window will be populated as seen below.



The top graph displays the value and Value-At-Risk of the portfolio over the duration of the back testing period. The lower graph compares the rates of returns of the portfolio and the benchmark, the FTSE 100. The right of the panel displays summarising statistics of the back testing. In particular, the number of exceptions where the amount of money lost in a day exceeded the stated Value-At-Risk calculated. The colour code is consistent with the green, yellow or red zones a model may fall into stated by the Basle Committee.