Final Year Project Report

Full Unit - Interim Report

Value at Risk

Nishant Mittal

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Professor V. Vovk

Department of Computer Science
Royal Holloway, University of London

December 5, 2013

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: approx. 8000

Student Name: Nishant Mittal

Date of Submission: 05/12/2013

Signature:

Table of Contents

Abstract

The report outlines the journey of the implementation of my **Value at Risk** Final Year Project. It covers the concept of Value at Risk, its uses in the Financial world and the various theoretical components which can be implemented in a program to compute the Value at Risk of a portfolio of assets. This report also contains information about the tools used in the implementation of the program and the software engineering practices followed in order to keep the source code easy to maintain, modify and update. There is also a project diary which shows how the project progressed from start to finish and highlights the reasons for any choices and decisions made.

The **Value at Risk** project involved writing a program which would take in some data about a set of stocks and assets, the portfolio, and would tell the user how much of their investment is at risk of loss over the next few days. Knowing the amount of risk in an investment is key to making decisions about what to do with the assets you hold in the financial world, Value at Risk has many uses and applications across the financial industry.

The implementation of the program was rather enjoyable as the individual components were small enough to be written within a day but came together to form a useful piece of software. The program was developed using Test Driven Development to ensure the individual components are working in line with their individual specifications.

The theory behind Value at Risk was sourced mainly from a book and supplemented with information from online resources which are listed in the literature survey section of this document.

Project Specification

Aims: Write a program for computing Value at Risk. Check its performance using backtesting.

Background: It is a very difficult and important problem to estimate the riskiness of a portfolio of securities. Value at Risk (VaR) is a tool for measuring financial risk. The goal of this project is to implement some of the methodologies for VaR computation (either under normal conditions or using "crash metrics") and test it on the real financial data. Other monetary measures of risk may also be studied.

Early Deliverables

- You will write a simple proof-of-concept program for computing Value at Risk for a portfolio consisting of 1 or 2 stocks using two different methods: model-building and historical simulation.
- Different ways of computing Value at Risk will be back tested and the statistical significance of the results analyzed.
- The report will describe the program (software engineering, algorithms etc.,).
- The report will also discuss different ways of estimating the variance of returns and the covariance between returns of two different securities (using the standard formula, exponentially weighted moving average, or GARCH(1,1)).

Final Deliverables

- The program should be extended by allowing derivatives (such as options) in the portfolio and using Monte Carlo simulation.
- Allow an arbitrary number of stocks (using eigen- or Cholesky decomposition).
- The final program will have a full object-oriented design, with a full implementation life cycle using modern software engineering principles.
- Ideally, it will have a graphical user interface.
- The final report will describe: the theory behind the algorithms.
- The final report will describe: the implementation issues necessary to apply the theory.
- The final report will describe: the software engineering process involved in generating your software.
- The final report will describe: computational experiments with different data sets, methods, and parameters.

Suggested Extensions

- Computing conditional VaR (also known as expected shortfall).
- Explore carefully the choice of parameters for EWMA and GARCH(1,1) using a loss function such as square or log loss.

- Empirical study of the two approaches to historical simulation for n-day VaR: reducing to 1-day VaR and direct estimation.
- Complement back testing by stress testing.
- Replacing the Gaussian model for stock returns by robust models.
- Computing monetary measures of risk different from VaR.

Reading

- John C. Hull. Options, futures and other derivatives, 7th ed., Pearson, Upper Saddle River, NJ, 2009.
- Hans Follmer and Alexander Schied. Stochastic Finance: An Introduction in Discrete Time, 3rd ed., de Gruyter, Berlin, 2011.
- Yahoo Finance as source of data: http://finance.yahoo.com/
- http://www.gloriamundi.org

Prerequisites:

- Java or C++ programming skills. Taking CS3930.

Chapter 1: **Introduction**

As part of the final year of my Computer Science degree, I've decided to embark on writing a program to compute the Value at Risk of a portfolio consisting of an arbitrary number of stocks and derivatives. I chose to do the Value at Risk project because I felt it would challenge my programming skills, allow me to use skills and knowledge acquired from the rest of my degree and complement my studies in the Computational Finance course.

During the implementation of the program I had the opportunity to learn new theory from various sources about how Value at Risk is calculated for different assets and used well-known industry standard tools with good software engineering processes to build up a series of algorithms and functions which come together to form the whole project.

In this report, I talk about the many aspects of my final year project and my journey from start to finish.

Chapter 2: **Concept of Value at Risk**

Individual traders along with large financial institutions often invest their money in various assets, such as stocks, shares, options and the like. A collection of these assets is called a portfolio. If someone is investing a large sum of money in a particular asset, or collection of assets, it is useful to know how much risk is associated in doing that.

Financial institutions usually calculate risks for every market variable to which they are exposed.[1] However, to provide a way of measuring the total risk the financial institution is exposed to, Value at Risk is considered a better solution.

Value at Risk (VaR), in essence, is a single number representing the total risk in a portfolio of financial assets. It is also used by treasurers of large corporations, fund managers and bank regulators. In banking, VaR is used to decide how much money a bank should keep (a reserve) for the risks it's exposed to, in case there is a financial crisis and the bank needs to be able to pay off some of the money it holds from its investors.

It is basically answering the question:"What is the maximum amount of money that can be lost if I have X amount of a stock over the next N days?"

One advantage of using VaR as a measure of the risk in a portfolio is that it simplies the risk as a single number which can be understood without the need for extensive knowledge about how it's calculated. However, this means that by decomposing the amount of risk a portfolio is exposed to into a single number, we may miss out complicated aspects of risk which may have been important to know.

2.1 VaR measure

When computing VaR, what we are really trying to obtain is an idea of the maximum amount of loss possible over a certain time period and our confidence in that estimate. VaR is simply a function of the time period in days and the confidence level (in percentage with 100 being absolutely certain). Banks usually calculate VaR over a period of 10 days with the confidence level being 99%. [1]

Over a time period of N days and a confidence level of $X\%$, VaR is the loss corresponding to the (100-X)th percentile of the distribution of the change in value of the portfolio.[1]

VaR, in the first instance, is usually measured with the time period as one day. The reason for this is that there simply is not sufficient amounts of data to estimate the behaviour of market variables over time periods of greater than a day. After computing VaR for one day, it is accepted that the VaR for N days is given by multiplying the one day VaR by $\sqrt{N}$.

There are two main ways of computing VaR: historical simulation and model-building.

2.2 Historical Simulation

Historial simulation involves holding a record of data about the change in value of all stocks over a period of time. Each i^{th} simulation assumes that the changes in value of each market variable are same as those covered on day i in the database. The change is portfolio value, ΔP, is calculated for each simulation and the VaR is calculated from the appropriate percentile of the probably distribution of ΔP.

This approach places an emphasis on using past stock data as a guide to what might happen to the same stock in the future. To compute VaR using the historical simulation approach data about the stock's performance over a large enough period of time (at least a year) is desirable so that we can build an idea of how the stock's value has risen or dropped historically. Using the stock's historical performance, we can then simulate all the possible scenarios of what may happen to the stock's value between today and tomorrow. These scenarios correspond to the changes in the stock value between subsequent days over the time we have the data for.

Initially, we would calculate the daily changes in the stock price between every pair of consecutive days over the time period of the collected data. This would result in a probability distribution of the possible changes in the stock value. We can put these possible changes in the value of the stock in ascending order. This would make it easier for us to compute the Value at Risk because if we wished to obtain the VaR with 99% confidence, we would simply have to look at the 1^{st} percentile of the ordered daily changes in data. For example, if we wished to compute the VaR with 95% confidence for a stock where we have 500 days of historical data, we would look at the 5^{th} percentile i.e. the 25^{th} worst loss and be able to say that the value of the portfolio will not decrease above this amount.

For each permutation of the possible value of the stock in the future (e.g. tomorrow), we have the following equation:

$$v_{\text{tomorrow}} = v_{\text{today}} \left(\frac{v_i}{v_{i-1}} \right)$$

where i denotes the i^{th} possible scenario for the change in value from all the possible scenarios in the historical stock data.

2.3 Model-Building

Computing VaR using the Model-Building approach is simpler than doing a Historical Simulation. Essentially, there are four things we need to know to computer VaR using this approach:

1. value of portfolio
2. time period (in days)
3. confidence level (as a percentage of 100)
4. (daily) volatility of stock price

In this approach, we assume that the change in value of the portfolio ΔP is linearly dependent on percentage changes in value of the stocks.

We can first calculate VaR for a single asset very simply. We first obtain the standard deviation of the daily changes in stock value by multiplying the volatility by the value of the portfolio (1). The one day VaR would then be the product of the standard deviation (1) and 2.33 (number of standard deviations our risk shall never exceed for 99% confidence) (2). The VaR for N days would then be the product of the one day VaR (2) and $\sqrt{N}$.

While this seems simple, VaR is more useful when being computed for a portfolio of a large number of varying stocks. This is where, to obtain a reliable value of VaR, we must take into consideration the covariance between two or more shares.

The model-building approach produces results very quickly in comparison to historical simulation, however, it assumes that the market variables have a normal distribution. In the real world, the distributions of the market variables are not normal. Model-building also gives relatively poor results for portfolios where the stock value changes are quite low.

2.3.1 Volatility

Volatility is very important in computing Value at Risk. It is the measure of the amount of fluctuation in the price of the stock over a period of time. For a highly volatile stock, there is high risk and for a stock with low volatility there is low risk of keeping it. The closing prices over a set period of time are used to calculate the volatility of a stock in different ways.

Standard formula

The standard formula is the simplest way of calculating the volatility of a stock. It involves comparing the closing prices from a period of time, over two consecutive days for example, and simply looking at the difference between them - known as the return. First we work out the returns of the stock price by using the calculation:

$$\text{return}_i = \frac{\text{closing price}_i - \text{closing price}_{i-1}}{\text{closing price}_i}$$

where i is used to keep track of the day we are working out the return for. From here, the volatility of the stock is the standard deviation of the range of returns over the specified time period. While this is an easy of estimating the volatility of a stock, it is not the most reliable. A slight improvement on this calculation can be made by using natural logs to work out the returns from the stock prices. We modify the calculation so that:

$$\text{return}_i = \ln\left(\frac{\text{closing price}_i}{\text{closing price}_{i-1}}\right)$$

which gives us a more convenient and slightly more reliable result. Using natural logs is considered to be the convention in finance. One of the benefits of using natural logs to work out the returns of stock prices is that logs are time-consistent i.e. we get the same result when calculating the change either over a collection of smaller time periods or over a larger time period of the same collective size as the first one. Moreover, this calculation is better at approximating the changes when they are relatively low (when changes in stock prices aren't

over 15%).

The volatility of the stock in this case would be the standard deviation of the series of returns over the given time period and can be generalised in one formula:

$$\sigma_n^2 = \frac{1}{m}\sum_{i=1}^{n} u_{n-i}^2$$

where σ_n^2 is the variance at day n, m is the number of days of stock data we have and u_{n-i}^2 is the square of the return between days n and i. The volatility, σ_n at day n would be the square root of σ_n^2.

While programming the volatility estimation, I first started using the the standard formula without the logs but found the VaR computation to be easier to understand and more reliable when I used natural log to compute the volatility.

The standard formula for calculating the volatility of a stock assumes equal weighting of each return. However it is accepted that more recent data about a stock's performance is generally a better indicator of its volatility than older data. Hence, there are still improvements which can be made on the estimation of the volatility of a stock price, such as using the EWMA formula.

EWMA - Exponentially Weighted Moving Average

While the standard formula is useful for computing the volatility of a stock, it doesn't take into account the validity and reliability of the closing prices with respect to time. The EWMA approach to calculating volatility gives more weight to the recent changes in stock prices than the older stock prices as it considers the recent data a better indicator of the performance of a stock than older data about the stock. This seems to be a valid argument and results in better estimations of the volatility of a stock. The EWMA approach requires four inputs and can calculate volatility for an infinite number of returns. The formula for the EWMA calculation for variance is:

$$\sigma_n^2 = \lambda\sigma_{n-1}^2 + (1-\lambda)r_{n-1}^2$$

where,
σ = volatility, σ^2 = variance
λ =*exponential weight coefficient, determines how returns are weighted (between 0 and 1)*
r = *periodic return*

This formula computes the variance at the current day which can then be used to compute the volatility of the stock's returns as $\sqrt{\text{variance}}$.

J.P. Morgan used a value of $\lambda = 0.94$ in their RiskMetrics program which is accepted as being convention in finance.

GARCH 1,1

Generalised Auto-Regressive Conditional Heteroscedastic further improves on the volatility calculation made by using EWMA through a technique known as *mean recursion*.

Mean recursion essentially limits the influence of larger fluctuations which can distort the mean of a range of data. The result of applying mean recursion is that the volatility estimates pull the variance closer to the long running average, an operation collectively known as *persistence*, so it gives more realistic estimates of volatility than EWMA and is the most popular way of calculating volatility of a stock price.

The formula for GARCH is:

$$\sigma_n^2 = \gamma V_l + \alpha r_{n-1}^2 + \beta \sigma_{n-1}^2$$

2.3.2 VaR for multiple stocks

So far we have thought about computing the Value at Risk for a single asset. In reality, a portfolio consists of many different stocks which have different volatilities and hence different VaR measures.

Let us consider the situation where we have two assets in our portfolio and we compute their individual VaR. The risk of the individual assets tells us nothing about the risk of the total portfolio as sometimes the two assets' returns may balance each other out and reduce the risk. On the other hand, the returns may also vary at a high rate between the two assets and end up increasing the total risk of the portfolio. Therefore, we need to consider the correlation between the returns of the two assets, which are random variables, in our portfolio which is known as the covariance.

Covariance Covariance is a measure of how similar the changes in two random variables are. If the returns of one asset increase in the same fashion as the returns of another asset, they are said to have positive covariance. Negative covariance occurs when their behaviours are opposite to each other i.e. greater values of one asset's returns correspond to lower values of the other. In a financial context, knowing the covariance between stocks is useful for investors so that they can reduce the risk of their portfolio in a process known as diversification. The EWMA and GARCH approach to calculating variances can be extended to calculate covariances. This is done simply by replacing the square of one stock's returns with the product of the two stock's returns we are finding the covariance for. The formulas for the covariance computation using EWMA and GARCH thus become:
EWMA:

$$\sigma_n = \lambda \sigma_{n-1} + (1-\lambda) r1_{n-1} \times r2_{n-1}$$

GARCH:

$$\sigma_n = \gamma V_l + \alpha r1_{n-1} \times r2_{n-1} + \beta \sigma_{n-1}$$

where σ_n is the covariance at day n, $r1_n$ and $r2_n$ are the respective returns of the two stocks.

Diversification Diversification is the concept of reducing the risk of a portfolio by investing in a variety of assets.[2] A diverse portfolio consists of assets whose fluctuations in value balance each other out over time, thus reducing the total risk of the portfolio.

Once we have the series of returns (daily changes in a stock's closing prices) for a number of stocks, we can create a variance-covariance matrix which holds the values of the covariances between a set of stocks. The matrix consists of rows and columns representing each stock in

the portfolio and the corresponding entries are the covariances between the stocks. We can model a sample covariance matrix for a portfolio of three stocks as follows:

$$\begin{pmatrix} cov(1,1) & cov(1,2) & cov(1,3) \\ cov(2,1) & cov(2,2) & cov(2,3) \\ cov(3,1) & cov(3,2) & cov(3,3) \end{pmatrix}$$

where *cov(i,j)* represents the covariance between stock i and stock j and if $i == j$ then the covariance is simply the variance of stock i.

For each pair of stocks, the variance of their values is calculated by:

$$\text{variance}_{i,j} = \text{investment in stock}_i \times \text{investment in stock}_j \times \text{cov}_{i,j}$$

The sum of the variances in the matrix is the variance of our portfolio of assets. This variance can then be used to calculate the volatility of the portfolio and then the VaR of the whole portfolio by:

$$\text{VaR} = \text{st.d.} \times \sqrt{\text{variance}_{\text{portfolio}}} \times \sqrt{N}$$

where *st.d.* is the number of standard deviations the loss should not exceed (2.33 for 99% confidence) and N is the number of days we wish to calculate VaR for.

2.4 Monte Carlo Simulation

Another way of calculating the Value at Risk of a portfolio of stocks is by using the Monte Carlo simulation technique which models the changes in market variables as Gaussian distributions. A general Monte Carlo method relies on repeated random sampling of a set of data to obtain a range of results which can then be used to approximate the probability of some outcome.

In the scope of this project the Monte Carlo simulation is a method where we repeatedly simulate possible changes in the value of our portfolio using random variables in order to estimate a value by which the portfolio is at maximum $X\%$ likely to reduce in value - the Value at Risk.

The simulation is implemented by running a large number of hypothetical trials over a certain number of days using Gaussian random numbers in a formula such as:

$$\text{S}_{i+1} = \text{S}_i + \sigma \text{S}_i \times \phi(0,1)$$

where S_i is the value of the asset at day i, σ is the volatility and $\phi(0,1)$ is a Gaussian random variable.

After simulating these trials we will have a series of possible values of the asset, which can then be ordered. From the series of ordered returns of the asset, we are able to select a percentile based on the confidence level we have, from which we can deduce how much value of our asset is at risk i.e. for 95% confidence our VaR can be obtained by selecting the 5^{th} percentile of the ordered returns of the asset.

In my implementation of Monte Carlo Simulation, I have computed both the final Value at Risk of the stock (from the values of the portfolio on the last day of each simulation) as well as the maximum Value at Risk of the stock which is the highest simulated risk experienced by the stock during the simulations.

Chapter 3: **High Level Design of Program**

3.1 UML Diagram

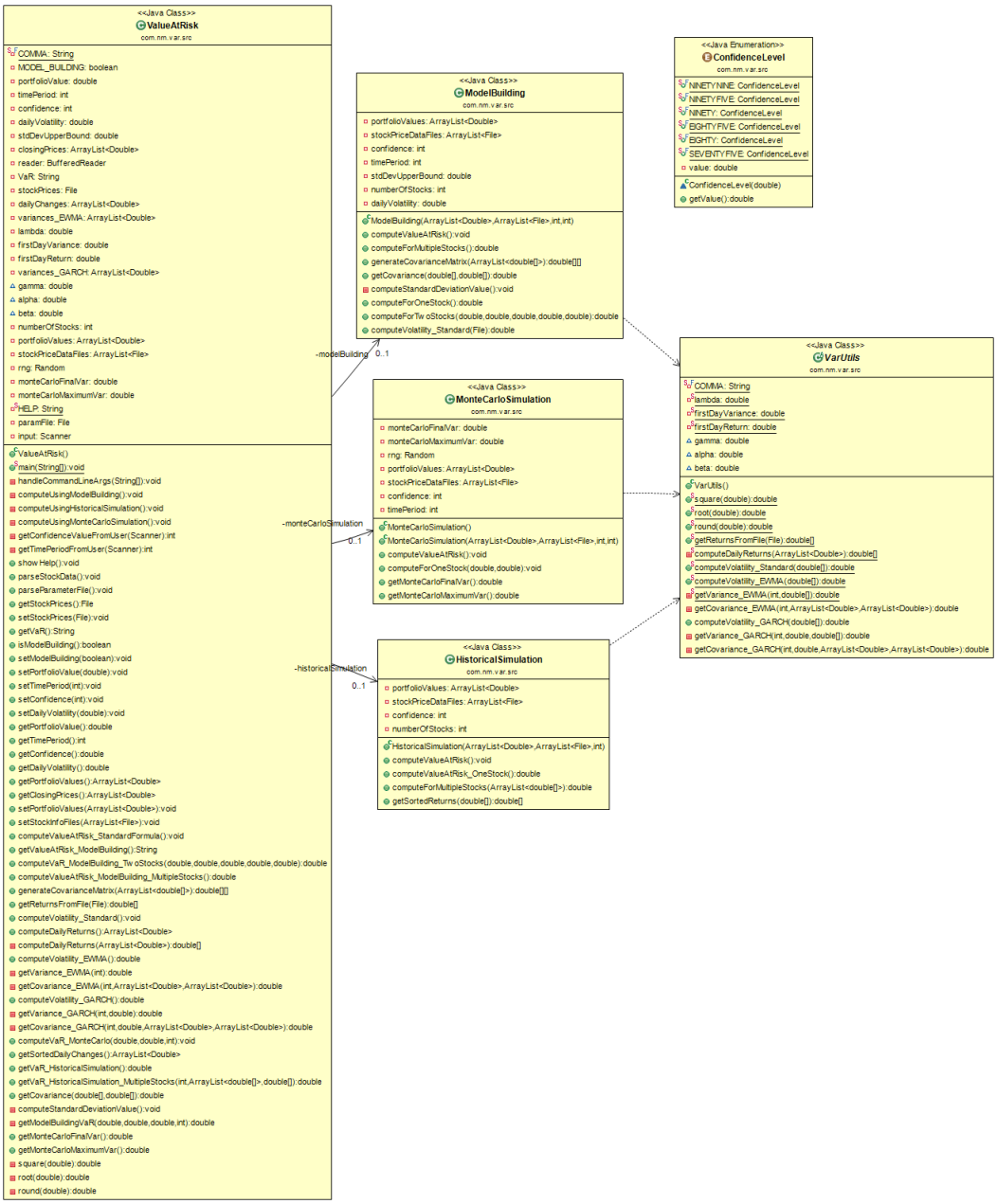

Figure 3.1: UML diagram for ValueAtRisk program

Initially, the program was written as one class being incremented with implementations of the Model Building, Historical Simulation, Monte Carlo and variance estimation algorithms. These implementations were tested through JUnit tests. For the interim submission, I decided to refactor the project so that it has a more logically sound structure, which can be seen in the UML diagram in this section.

3.2 Role of Classes in Program

3.2.1 ValueAtRisk.java

This is the main class of the program which acts at the entry point when the program is run from the command line. Its responsibilities include:

- verifying and handling command line arguments (input parameter file and mode of VaR calculation)
- showing the help or usage information for the program
- storing valid input parameters (e.g. confidence level and time period for VaR calculation)
- initialising the computation of VaR using user-specified approach (by calling relevant classes in project) and input parameters
- outputting helpful, informative messages to console in case some user input needs to be fixed

3.2.2 ModelBuilding.java

- storing user supplied parameters for VaR computation
- using parameters to compute VaR for a portfolio of multiple stocks
- generating a matrix of covariances between pairs of stocks in the portfolio
- computing covariance between returns of two stocks
- computing number of standard deviations the portfolio value should not decrease by depending on confidence level supplied by user
- outputting computer VaR to console

3.2.3 HistoricalSimulation.java

- storing user supplied parameters for VaR computation
- using parameters to compute VaR for a portfolio of multiple stocks
- running Historical Simulation algorithm to compute VaR
- sorting returns in ascending order
- computing desired percentile for VaR calculation from confidence level
- getting value at risk from desired percentile
- outputting computed VaR to console

3.2.4 MonteCarloSimulation.java

- storing user supplied parameters for VaR computation
- using parameters to compute VaR for a portfolio of one stock
- running Monte Carlo Simulation algorithm to compute VaR
- running simulations of a specific number of days each to simulation stock value changes
- outputting computed final and maximum VaR to console

3.2.5 EWMA.java

This class will eventually be responsible for computing variances and covariances using the Exponentially Weighted Moving Average approach (EWMA).

3.2.6 GARCH.java

This class will eventually be responsible for computing variances and covariances using the GARCH(1,1) approach.

3.2.7 VARUtils.java

This is a static class which is used across the project for executing commonly used functions. Its responsibilities include:

- storing parameters used in methods (for EWMA, GARCH)
- returning the square of a number
- returning the square root of a number
- returning the rounded value of a number
- getting the series of returns from a file containing historical stock price data containing closing prices
- computing returns of data from list of closing prices of a stock
- computing variance and volatility using the standard formula for a series of returns
- computing variance,volatility and covariance using the EWMA formula for a series of returns
- computing variance,volatility and covariance using the GARCH formula for a series of returns

3.2.8 ConfidenceLevel.java

This is an enumeration of the possible values of confidence which a user may wish to supply. The idea for this class is to help restrict the user from making unrealistic confidence settings for VaR computation. This class would be useful when implementing a user interface where the user is only able to select from a list of pre-defined confidence levels.

3.3 Sample flow of control

(From programInstructions.txt in project root folder on repository)

Instructions to run Value at Risk (VaR) program.

Standard usage: java -jar ValueAtRisk.jar ¡parameterFile.csv¿ ¡computationMethod¿

First command line argument accepted is the location of the parameter file which contains information about the location of the historical price data and the respective investments made.

Second command line argument is the method of computing VaR, 0 is Model Building, 1 is Historical Simulation, 2 is Monte Carlo Simulation.

Usage/help:
java -jar ValueAtRisk.jar
java -jar ValueAtRisk.jar help

download jar file from repository location: link

also download the sample parameter and historical stock price data files from: link

You may have to note down the file locations in the parameter files or modify them to suit your needs. testParams2.csv assumes all stock data is in same folder as itself for example.

for simplicity, keep jar file, parameter files and stock price data files in adjacent/same directories = program folder.

open a command prompt in program folder (Shift+Right Click in program folder then Open command window here) or navigate to it in command prompt

type in java -jar ValueAtRisk.jar testParams2.csv 0—1—2

program will output the method of computing VaR it has deduced from the input

program may also inform of any problems with the files provided

depending on the mode of computing VaR specified, the program will prompt for a confidence level and time period to compute VaR for

supply these and the program will output its results.

3.4 Implementation of Algorithms

The algorithms implemented for the interim submission have been implemented to be as readable as possible. However, there as some aspects of the computation of Value at Risk for which there were no improvement substitutions. These included:

- High number of copy & compare operations - most of the algorithms in the program need to work with arrays or lists of numbers in order to compute returns, variance, covariance and percentiles. These operations can slow down the program given a large

enough input.

- Array copying and sorting - algorithms such as the Historical Simulation and Monte Carlo Simulation rely heavily on sorted arrays to work. Sorting an array is an expensive operation in terms of running time.
- Recursive computations - the EWMA and GARCH variance estimations are recursively computed, which can be slow for a large number of returns.

In order to keep the running time of the algorithms low for a large input, I looked to optimise the code at regular intervals, which included discarding unused variables, removing unnecessary computations, minimising use of `if-else` ladders in favour of `switch` statements and combining iterative loops where possible.

After the interim submission, I will be looking into reducing the time taken for computing VaR for large inputs by implementing a multi-threaded program. This also opens up the scope of computing VaR using different approaches simultaneously meaning the user will not have to do them sequentially.

3.5 Testing

The computation of Value at Risk can be tested using two models.

3.5.1 Stress testing

Stress testing is conducted by measuring what changes our portfolio would have experienced under some of the extreme market moves over the past few decades.

3.5.2 Back testing

Back testing is conducted by measuring the number of times the actual loss in the value of our portfolio exceeded our Value at Risk estimate from data over a similar time period.

I will aim to implement some back testing functions before the start of next term and also introduce stress testing into my program.

3.6 User Interface

The user interface for the interim submission is a command line interface which tries to be as informative as possible to the user and takes in user input for some parameters.

For the final submission, I would aim to implement a graphical user interface (GUI) which would make it easier for the user to specify parameters and files to be used in computing Value at Risk.

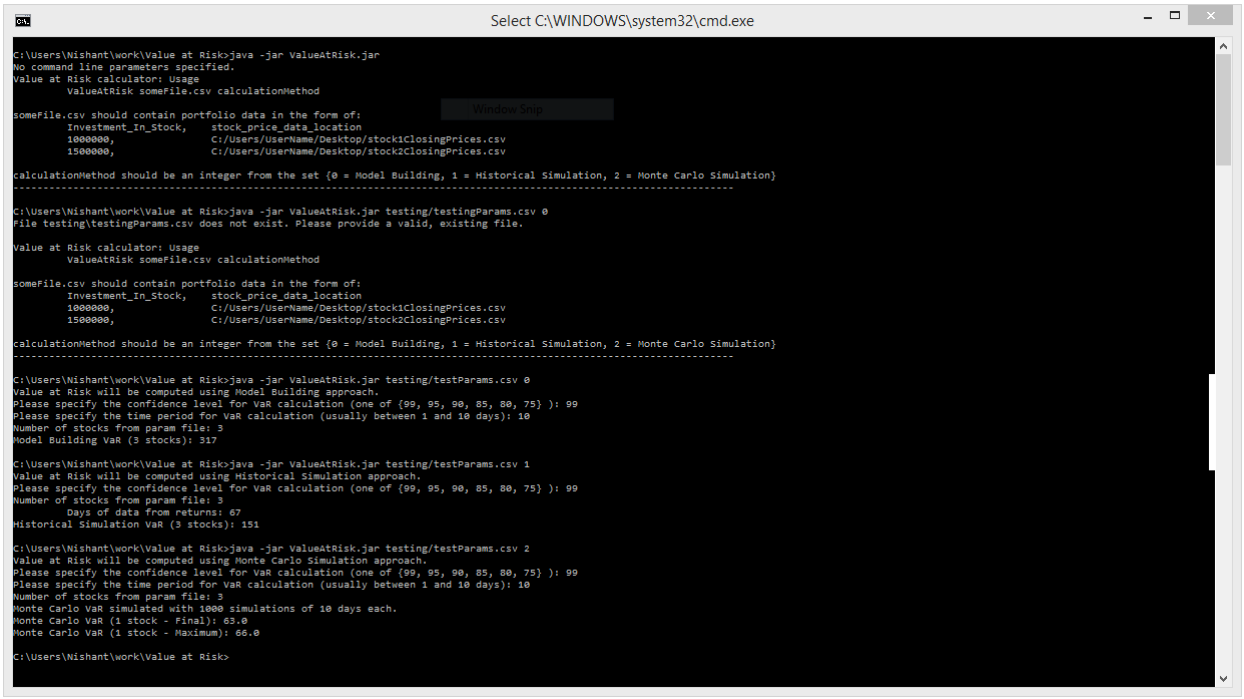

Figure 3.2: Example of program usage

Chapter 4: **Software Engineering Principles**

When programming a software based project I agree that it is important to follow some good Software Engineering principles so that the project not only functions well, but is also designed well to allow for improvements and extensions easily. In this chapter I talk about a few principles I have followed when writing my Value at Risk program which have helped improve the standard of my code.

4.1 Good code design

During the implementation of the Value At Risk program I have tried to follow some good software engineering principles, some of which are explained below:

- DRY (Don't Repeat Yourself) - Where possible, I have tried to avoid writing repeated code by extracting common code to its own method and call it wherever needed. The best example of this principle in my program can be seen in the VarUtils.java class which contains the most commonly used methods in the whole program.
- KISS (Keep It Simple, Silly) - I have tried to avoid using complex code in order to make it easier to debug and fix.
- Code readability - I have made use of descriptive variable names and assignmed complicated calculations to such variables in order to improve the readability of my code.
- Code formatting - I have used a consistent and clear code formatter by modifying the default formatter in Eclipse which also makes code easier to read.

4.2 Design patterns

I shall be implementing at the least the Model-View-Controller (MVC) and Facade design patterns for when I start writing the GUI for the program in order to decrease the coupling between classes and increase the cohesion of each class.

4.3 Usability

I have tried to make the program easy to use by providing plenty of command line handling logic which checks for incorrect user input and informs the user of what is required from them at every step.

I chose to accept a parameter file as input as it can be edited for use once and used repeatedly which also allows for batch operations in the command line.

Chapter 5: **Technologies Used**

5.1 Java Programming Language

I chose to write my Value at Risk project using the Java programming language. This is because almost all of my programming experience comes from working with Java, especially from my Year in Industry. Java provides ample resources, libraries and extensions for the scope of this project and I feel confident I will be able to implement all of the algorithms I need for this project using Java.

While not being as memory efficient as C++ or C#, I chose to go with what I know and even then I have improved my skills in certain areas when programming the project using Java.

5.2 Development of Code

In order to develop the code for the Value at Risk project, I made use of the Eclipse IDE (Integrated Development Environment). Eclipse supports multiple languages and numerous plug-ins for added functionality and interaction with third-party products. Eclipse is used widely for development using the Java programming language as it provides intelligent code-completion, spell-checking, checks for basic coding and compilation errors and a highly customizable and feature-packed interface. Moreover, I was able to consistently make use of a variety of keyboard shortcuts to aid fast navigation and code refactoring across my Java project.

5.2.1 External libraries

I have made use of the following external Java libraries in my project:
Apache Commons Math 3.2 - provides DescriptiveStatistics class which I used to compute covariances with equal weighting in some VaR calculations.
Apache Commons CLI 1.2 - library providing enhanced command line parsing features, may be useful if the program grows to accept more complex parameters from user
Apache Commons IO 2.4 - library providing enhanced file handling utilities for Java, may be useful when implementing the GUI where the user will be able to select files from their directories, which will need to be verified.

5.3 Testing

JUnit is a unit testing framework used in Java. It can be used for test driven development, where a feature is developed by writing tests first and then making them pass by implementing the feature incrementally. This development approach ensures all the code written has been tested extensively.

Unit testing is the process by which individual units of source code (e.g. a function/method, class or collection of methods) are tested to verify that they are successful in achieving their

individual goals. The idea of unit testing is to verify the correct functionality of the modules of the program to assert that a larger module is functioning correctly.

I made use of JUnit to unit test the Value at Risk project code where possible in order to test the functionality of individual methods and classes by preparing some input parameters and comparing the data returned from the module being tested with some expected output. Unit testing comes with benefits such as the testers being able to find additional problems in the code they are testing early on, allowing more time for the bugs to be fixed and to check if some refactoring/modification in the code has caused an issue by running the existing unit tests and checking for failures.

Through this practice, I will be able to build up a collection of hundreds of unit tests which can be run after every modification to ensure no previous functionality is broken.

5.3.1 Software versioning and revision control

I have used Subversion (SVN) for maintaining my project code safely in a secure repository. Subversion is a commercial version control system designed to provide secure repositories for software projects. It provides all the features expected from a version control system e.g. managing a list of modified files, submitting the files to the repository, checking out entire projects, merging and integrating files from one branch to another etc.

A version control system provides complete file revision history, a centralised repository which is access-controlled and management of modifications made to files (source code, in this case).

It is necessary to use a version control system in software engineering because there are always multiple users working with and updating the same source code. This code needs to be available for modification, fixing and updating to every user in a way which does not compromise its integrity.

Initially, I did not realise that I had not created a proper project structure in my SVN repository as I had not used SVN for a whole year prior to that. However, later on, I created a new project structure on my repository complete with the trunk, branches and tags which I was able to make better use of.

I regularly committed work to my repository and wrote informative comments for each update I committed to the repository.

After setting up the proper project structure, I also branched off the trunk to start the refactoring work on my program. I made regular commits to the refactoring branch until I was ready to merge the refactored program back into the trunk.

Finally, I created tags whenever I had a candidate, fully working, release of my program, one of which is the InterimSubmission tag.

I am sure I will be making better use of my SVN repository when the program extends in the second term.

5.4 New material learnt

Theory about finance and Value at Risk[1] and from the CS3930 Computational Finance course.

Probability theory - Gaussian Random Variables from the CS3930 Computational Finance course.

5.5 Knowledge from courses

- good coding standards - CS2800 Software Engineering
- design patterns - CS2800 Software Engineering
- algorithms and improvements - CS2860 Algorithms & Complexity I
- multithreading - CS3750 Concurrent & Parallel Programming and CS2850 Network Operating Systems
- planning work and client meetings - CS2810 Team Project in Software Engineering

Chapter 6: **Project Diary**

At the beginning of the project, I have to admit I did not have a very clear idea of exactly how I would be implementing the Value at Risk program. This was because I did not get a chance to review the project over the summer as I was on my Year in Industry, working full time as a Software Engineer. After reading around the concept behind Value at Risk I slowly began to understand its relevance in the financial world.

After discussing the initial expectations with my supervisor, I was able to plan my project work until December in terms of reports written and programs developed. Mathematics is not a strength of mine but I was confident that I would be able to implement the computations using my programming skills after I'd taken time to understand the individual algorithms by hand using John Hull's book: Options, Futures and Other Derivatives.

I had originally struggled to implement what I now consider to be relatively simple algorithms and I feel that was because I had not yet grasped their roles in the computation of Value at Risk. However, as the term went by, I began to take a great interest in the program and as the Computational Finance course covered Value at Risk everything seemed to come together and make sense for me.

As I had been writing the program in one big class, I decided to improve its design my refactoring the program into logical sub modules and implemented command line parameter handling which makes is relatively easy to use.

It would be fair to say that, at some points, the amount of time I spent working on the project was affected by the number of assignments ongoing for other courses, but I do feel that I could have worked a lot harder at the start of the term. I will be taking this message into the next term and will hope to realise my ambitions of making my project one of the best this year.

Bibliography

[1] John Hull. *Options, Futures and Other Derivatives*

[2] Sullivan, Arthur; Steven M. Sheffrin (2003). *Economics: Principles in action*