Particle Swarm Optimisation
to solve the Portfolio Selection Problem
in a Function Based Environment
(30 Credit Course)

Anthony Sergio Chapman

SUPERVISOR: **Dr. Wei Pang**

Bachelor of Science
of the
University of Aberdeen.

Department of Computing Science

2014

Declaration

I declare that this document and the accompanying code has been composed by myself, and describes my own work, unless otherwise acknowledged in the text. It has not been accepted in any previous application for a degree. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: 2014

Abstract

This research project aims to investigate how the Particle Swarm Optimisation (PSO) algorithm could be applied for multi-dimensional optimisation in a real-world domain, namely finding an optimal portfolio to invest in, given a set of assets and their expected returns. The Particle Swarm Optimisation algorithm, created by James Kennedy and Russell Eberhart, belongs to the field of Swarm Intelligence. It uses a collection of particles (a swarm) to explore a given search space for an optimal value. To be able to solve the portfolio optimisation problem, a way to measure and compare different portfolio had to be created. Using two-sided coherent risk measure, created by Zhiping Chen and Yi Wang, and concepts from mathematics, measure theory, the portfolio optimisation problem turned into a function what the PSO algorithm is able to optimise.

The main goal of this project is to create a reliable method for computing and comparing a portfolio's success or failure in order for the algorithm to be able to find an optimal one. Secondary goals such as experimenting and expanding the PSO algorithm were also achieved. Finally, the effectiveness of the resulting approach was assessed through experimentation and different parameter settings. The report shows the background research, design, implementation, testing and experimentation with the expanded Deterministic Dendritic Cell Algorithm based on datasets from two different domains (Port Scanning data and SEPA River data).

Acknowledgements

Contents

List of Tables

List of Figures

Chapter 1: Introduction

This section provides an overview of the project and explains the basic principles of the initial approach and ideas for expansion. It contains a list of primary and secondary goals as well as motivation for carrying out this research.

1.1 Overview

Swarm Algorithms that belong to the Swarm Intelligence Systems were inspired by the behaviour of ant colonies, bird flocking, animal herding, bacterial growth, and fish schooling. As flocks of birds, the artificial swarm is able find an optimal location through communication with local agents as well as the environment. Swarm Algorithms have proven to be effective in providing ways to find global optima in potentially difficult search spaces. The techniques used in Swarm Intelligence Systems closely related to artificial intelligence and are often applied to simulations, robotics and optimisation problems.

The Particle Swarm Optimisation algorithm belongs to the field of Swarm Intelligence Systems and optimizes a problem by having a population of candidate solutions, called particles, moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity, shown in Equation (1). Each particle's movement is influenced by its local best known position but, is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. Originally, PSO was designed to simulate bird flocking behaviour [15], it was later discovered that it would be used to find optimal positions, such as a swarm of birds finding food in a large city.

1.2 Motivation

Over the past twenty years the amounts of data that is being produced by the financial market have increased drastically, it is essential to use efficient algorithms to analyse such data. The Particle Swarm Optimisation algorithm has already been effectively used in various applications, mainly biomedical, clustering, and modeling. However, the use of PSO to find optimal values when trading stocks or other securities has not been exploited thoroughly enough. Given that dealing with the financial market requires dealing with vasts amounts of numerical data, I believe one can formulate complex financial concepts, such as optimising a portfolio, and apply them to PSO. The PSO algorithm could potentially be applied to various systems and applications to a wide range of domains and an application to solve the portfolio selection problem could start a new generation of evolutionary financial solutions.

This idea was greatly inspired by both my supervisor, by introducing me to the world of evolutionary algorithms, and recently discovered ways to measure a portfolio's risk [4]. The author provides a comprehensive overview of portfolio risk measurement, describing how it can be used to optimise a portfolio as well.

1.3   Primary Goals

A list of primary goals which need to be completed in order to classify the research project as a success.

- Understand the principles behind Particle Swarm optimisation.

- Design an appropriate expansion to solve the portfolio selection problem.

- Understand the principles begin portfolio risk measure.

- Test the resulting application and assess whether it can be used in a professional environment.

1.4   Secondary Goals

Secondary goals could be added to the project depending on time-constraints and success of completing the primary goals.

- Compare my implementation to that of an investment manages.

- Write a simple GUI for the application.

Chapter 2: Background

In order to expand and adapt existing Particles Swarm Optimisation methods an initial background research has to be done to fully understand the concepts involved. Similarly to apply the algorithm to solve the portfolio optimisation problem, a deep understanding on how to formulate the problem in a way that the algorithm will understand will have to be made.

This chapter describes the key concepts which I will use and eventually expand, implement and improve. Section 2.1 provides the ideas and methods for PSO. Section 2.2 gives some background on how to use two-sided coherent risk measure[4] to solve the portfolio selection problem. Section 2.3 describes some of Haskell's attractions.

2.1   Particle Swarm Optimisation

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Kennedy and Eberhart in 1995, discovered through simplified social model simulation [15, 30, 6, 22]. It simulates the behaviour of bird flocking involving the scenario of a collection of birds randomly looking for food in a search space. None of the birds know where the food is located, but they do know how far the food location is from their current positions. An effective technique for the bird to find food is to adjust their velocity according to the bird which is nearest to the food. PSO was motivated from this scenario and was developed to solve complex optimization problems, where the optimum position of the fitness function is where the food is located and all the birds are the particles searching for this optimum position.

An interesting thought is presented in [15] is that PSO can be used to model human social behaviour. One important difference, one that I find so fascinating, is its abstraction. It states that humans adjust not only their physical movements but also our cognitive position too; "we tend to adjust our beliefs and attitudes to conform with those of our social peers"[15]. One of the major distinctions in this model compared to that of bird flocks or school's of fish is that collision works differently; "two individuals can hold identical attitudes and beliefs without banging together, but two birds cannot occupy the same position in space without colliding".

In the conventional PSO, the behaviour displays particles in a multidimensional space where each particles has two properties: a position vector and a velocity vector. Each particle i in the swarm has the properties shown in (2.1):

$$V_i^t : \text{The velocity of particle i at time t.}$$

$$X_i^t : \text{The current position of particle i at time t.}$$

$$Pbest_i^t : \text{The personal best position of particle i at time t.} \tag{2.1}$$

$$Gbest^t : \text{The global best position of particle i at time t.}$$

At each step, the velocity of the *ith* particle will be updated according to the following equation:

$$V_i^{t+1} = \omega V_i^t + c_1 r_1 \times \left(Pbest_i^t - X_i^t \right) + c_2 r_2 \times \left(Gbest^t - X_i^t \right)$$

where

$\quad\quad V_i^{t+1}$: The velocity of particle i at time t + 1.

$\quad\quad V_i^t$: The velocity of particle i at time t.

$\quad\quad X_i^t$: The current position of particle i at time t.

$\quad\quad \omega$: Inertia weight parameter.   (2.2)

$\quad\quad c_1, c_2$: Acceleration coefficients.

$\quad\quad r_1, r_2$: Random numbers between 0 and 1.

$\quad\quad Pbest_i^t$: The personal best position of particle i at time t.

$\quad\quad Gbest^t$: The global best position of particle i at time t.

In the updating process shown in Equation (2.2) the acceleration coefficients c_1, c_2 (c_1 determines the importance one following the individual personal best whereas c_2 determines the importance of following the particles with the global best position)and the inertia weight ω (which states how stubborn a particle is at not deviating from their path, regardless of previous results) are predefined, and r_1, r_2 (these introduce the concept of randomness so that a particles is able to switch between following their personal best and that of the whole swarm) are uniformly generated random numbers in the range [0,1].

The following algorithm 1 is one of the standard PSO. The pseudo-code outlines the procedures implemented in [26], it will be followed by an explanation on how it works.

Initialisation

for *i = 1,. . . ,S* **do**

> initPosition(i)
>
> initBestLocal(i)
>
> **if** *i = 1* **then**
>
> > | initBestGlobal()
>
> **end**
>
> **if** *improvedGlobal(i)* **then**
>
> > | updateGlobalBest(i)
>
> **end**
>
> initVelocity(i)

end

Main program

while *not endingCondition()* **do**

> **for** *i - 1,. . . , S* **do**
>
> > createRnd(r_1, r_2)
> >
> > updateVelocity(i, r_1, r_2)
> >
> > updatePosition(i)
> >
> > **if** *improvedLocal(i)* **then**
> >
> > > | updateBestLocal(i)
> >
> > **end**
> >
> > **if** *improvedGlobal(i)* **then**
> >
> > > | updateGlobalBest(i)
> >
> > **end**
>
> **end**

end

Algorithm 1: PSO pseudo-code.

In Algorithm 1 *S* is the number of particles in a swarm, one can see the pseudo-code for a standard PSO. A step by step explanation of Algorithm 1 is as follows: First, there is an initialisation for all the particles in the PSO, for every particle *i*, *initPosition(i)* randomly created a particle with a designated position in the search space. For the first step, *initBestLocal(i) = initPosition(i)*, but it will be updated afterwards. Then the *if* function makes a first global best and if any other particles has a better global best position it will be set as the new global best. *initVelocity(i)* gives particle *i* an initial velocity which is randomly generated.

After the initialisation, the core of the PSO method is executed until the *endingCondition()* is satisfied, which can be the number of iterations or an improvement threshold. In the body of the *While* loop all particles are updated. The first step is to generate random numbers used in the velocity Equation (2.2). Then the actual velocity is updated. In the next step,*updatePosition(i)* updates the position of a particles in the search space and checks the fitness function value for improvement or not. At the end of the *for* loop, if *improvedLocal(i) = True* then updates the local best position and finally it is similar for updating the global best position.

2.2 Portfolio Optimisation

The term portfolio refers to a collection of investments such as stocks, bonds or other securities [34]. For this project we will focus on stocks but it would be trivial to include other such securities into the application as they behave in a similar manner to stocks. Portfolio Optimisation is the process of choosing the proportion of various assets to be held in a portfolio in such a way that any other choice would result in an equal or worst portfolio, here we have referred to the comparison of portfolio from the rate of return or the level of risk, ie one portfolio is better than another if it has a lower level of risk or higher rate of return.

This dissertation will focus on the two-sided coherent risk measure introduced in [4]. This revolutionises how to calculate risk as it takes into account both downside as well as upside risk in unison. Compared to previous risk measures this new measure has the following advantages:

- The whole domain loss distribution information is utilised, making it superior for finding robust and stable investment decisions.

- Suitably selecting the combination coefficient and the order of the norm of the downside random loss, it is easy reflect the investor's risk attitude and to control the asymmetry and fat tails of the loss distribution.

- It is easy to compute and therefore easier to apply to optimisation models.

Now this is where some mathematical background is needed, let $\|X\|_p = (E_Q|X|^p)^{\frac{1}{p}}$, $\|X\|_\infty = ess.sup\{|X|\}$, $\sigma_p^\pm(X) = \|(X - E_Q[X])^\pm\|_p$. As already mentioned, this two-sided risk measure takes into account both sides of the loss distribution, so relative to the expected return value $E_Q[X]$, the random variable $(X - E_Q[X])^-$ is the "downside risk" of X, which according to [4] might be more crucial to an application than the analogues "upside risk" variable $(X - E_Q[X])^+$.

This is where the name, *two-sided risk measure*, comes from; it considers both sides or risk, downside and upside. Due to the way downside and upside risk can be represented and the monotonically increasing property of $\| \cdot \|_p$ with respect to p, the two-sided risk measure can be determined by the following equation:

For $1 \leq p \leq \infty$ and $a \in [0,1]$

$$\rho_{a,p}(X) = a\sigma_1^+(X) + (1-a)\sigma_p^-(X) - E_Q[X] \tag{2.3}$$
$$= a\|(X - E_Q[X])^+\|_1 + (1-a)\|(X - E_Q[X])^-\|_p - E_Q[X]$$

$\rho_{a,p}(X)$ in Equation (2.3) is generated by first taking the 1-norm of the positive deviation and the p-norm of the negative deviation and then taking the combination of these two norms. The inclusion of $-E_Q[X]$ in Equation (2.3) ensures that $\rho_{a,p}(X)$ is coherent [4].

One of the advantages proposed for two-sided coherent risk measure was that it would be easy to reflect an investor's attitude towards risk, this is achieve by the variables p, a in $\rho_{a,p}(X)$ in Equation (2.3). An investor's risk adverse attitude is represented by p, where the larger p is, the more risk the investor is willing to take, so small p low risk, large p high risk. In Equation (2.3) a is the factor controlling the balance between good and bad volatility [4], $a \to 0$ represents an investors' attitude towards good volatility and $a \to 1$ would be the opposite. Thus different investors might choose different values for p and a.

A reader might not see how Equation (2.3) has much to do with optimising a portfolio from a first glance, a quick example will be shown to illustrate its use. Suppose we have N assets to choose from and for $i \in [1, N]$ let $x_i \in [0, 1]$ be the proportional weight of asset i in the portfolio, let $r_i \in \mathbb{R}$ be a variable which represents the rate of return of asset i and $\widehat{r_i}$ be the expected return of asset i. Then the rate of return of a portfolio can be expressed as $R = \sum_{i=1}^{N} x_i r_i$ and the expected return $\widehat{R} = \sum_{i=1}^{N} x_i \widehat{r_i}$

From Equation (2.3) we can measure the risk of a portfolio as follows:

$$\rho_{a,p}(R) = a\|(R - \widehat{R})^+\|_1 + (1 - a)\|(R - \widehat{R})^-\|_p - \widehat{R} \tag{2.4}$$

Now that there is a way to measure risk, the portfolio optimisation problem can be formulated as follows:

$$\min \rho_{a,p}(R) \tag{2.5}$$

Subject to constraints:

$$\widehat{R} = \eta$$
$$\sum_{i=1}^{N} x_i = 1 \tag{2.6}$$
$$l_i \leq x_i \leq u_i$$

Equation (2.5) refers to the minimisation of a portfolio's risk. Now we give an explanation for the constraints in Equation (2.6), η is the required rate of return of a portfolio, that is what the investor has to gain, this constraint ensures that the portfolio reaches its target return. The sum of x_i ensures the investors do not buy more than what they can afford and also that the investor invests all the capital it can. Finally, l_i and u_i are lower and upper limits for how much proportion of each asset an investor is allowed to invest in, this is useful for diversification [35].

2.3 Haskell

This section briefly introduces the main attractions of the functional language Haskell. One big advantage of pure functional programming languages is that the absence of side-effects provides a clear semantic framework to analyse the correctness of programs and algorithms.

The core notion of functional programming is that a program is a mathematical function, it has an input and produces as output. Simple programs can be created easily and through function composition, more complex programs can be created with minimal effort.

"Real World Haskell" by Don Stewart, Bryan O'Sullivan, and John Goerzen has been very useful in understanding how the language works and maximising any Haskell program's efficiency. Having a very strong mathematical background I often struggled to understand or accept very simple concepts in object-orientated languages such as Java or Ruby. Having been introduced to Haskell, I have endeavored to understand the language as best as I can.

Haskell is one of the leading lazy evaluation languages in the functional programming community [14]. Lazy evaluation is an strategy which delays the evaluation of each expression until

the exact moment when it is actually required, this is indeed a powerful tool. Haskell is also a strongly typed language which includes polymorphism and high-order programming facilities. In addition, Haskell is able to work with arbitrarily large numbers, unlike most other languages, by default all numbers are set as *Integer* which can contain arbitrary-precision integers. This feature ensures no overflow errors occur, unless the programmer changes the default. As this project is working over multi dimensional optimisation and in finance, precision is important, this suits the project very much.

List comprehension [36] is syntactic construct for creating list based on existing lists. This is a very useful given that we might store all expected return and rate of return values in lists. We can use list comprehension to formulate complex equations, such as calculating the expected portfolio return, which is $\widehat{R} = \sum_{i=1}^{N} x_i \widehat{r}_i$, as in Section 2.2, easily. The code could therefore be as follows:

```
expPortR :: [Double] -> [Double] -> Double
expPortR weight expRet = sum [ x*y | x <- weight, y <- expRet ]
```

The first line states the type which in this case mean, *expPortR* is a function which takes in two list of type *Double* and return a *Double*. The first part of the second line gives names to the assigned lists, the lists which will be given to the function. The second part of the second line with sum up the corresponding expected returns with their given weights.

Chapter 3: Related Work

Use pkg bbding with

For better understanding of the portfolio selection problem and particle swarm optimisation, their strengths, weaknesses and applications, a study of previous and related works for the measure of risk in a portfolio and alternatives to PSO was made. I will read and evaluate some strengths and weaknesses on the related models and outline where they fail when compared to what this project wants to achieve.

3.1   Markowitz Model

One of the first contributions to the portfolio problem was made by Markowitz [19] which was later described in more detail in his book [20]. Markowitz introduced the mean-variance model which considers the variance of the portfolio as the measure of the investor's risk under a set of assumptions. According to Markowitz, the portfolio selection problem can be expressed as an objective function subject to linear constraints.

Following Markowitz, the investment horizon includes a single period whereas the investment strategy is to select an optimal portfolio at the beginning of the period which will be held unchanged until the date of termination. The joint distribution of one-period security returns is assumed to be a multivariate normal distribution, which in turn follows that the distribution of the portfolio return is also normal.

Let $S = (S_1, S_2, S_3, \ldots, S_N)$ be the set of N assets where each asset S_i has a rate of return represented by a variable R_i with an expected return r_i and each pair of assets (S_i, S_j) has a covariance σ_{ij}. The variance-covariance matrix $\sigma_{n \times n}$ contains all the covariance values, furthermore, it is a symmetric matrix and every diagonal element σ_{ii} represents the variance of asset S_i. Let π be a positive value which represents the investor's required expected return. Generally the values r_i σ_{ij} are estimated from past data and are fixed during the period of investment.

According to Markowitz, a portfolio is a real valued vector $X = (x_1, x_2, x_3, \ldots, x_i)$ where each variable x_i represents the percentage invested in a corresponding asset S_i. The value $\sum\limits_{i=1}^{N} \sum\limits_{j=1}^{N} x_i x_j \sigma_{ij}$ is the variance of the portfolio and it is the measure of risk associated with the portfolio. From

this, we may obtain a constrained portfolio optimisation problem:

$$Minimise \; \sum_{i=1}^{N} \sum_{j=1}^{N} x_i x_j \sigma_{ij}$$

$$Constraints: \; \sum_{i=1}^{N} x_i r_i = \pi \qquad (3.1)$$

$$\sum_{i=1}^{N} x_i = 1$$

$$x_i \geq 1, i \in \{1, 2, \ldots, N\}$$

Here, the objective function minimises the total variance (risk) associated with a portfolio whilst the constraints ensure that the portfolio meets the expected required return of π and the proportions of all the assets sum to 1. The non-negative constraint means that no short-selling is allowed.

This framework presumes that the investor uses a quadratic utility function or that asset returns follow a multivariate normal distribution. This implies that the rate of return of a portfolio of assets can be completely explained by the expected return or variance of assets. The efficient-frontier is the set of assets which provide minimum risk for a specified level of return. Solving the Markowitz portfolio optimisation problems for different specified expected portfolio returns will give us a set which represents the efficient-frontier, it is a smooth semi-increasing curve which represents the best possible trade-off between risk and expected return, also called the set of Pareto-optimal portfolios [37].

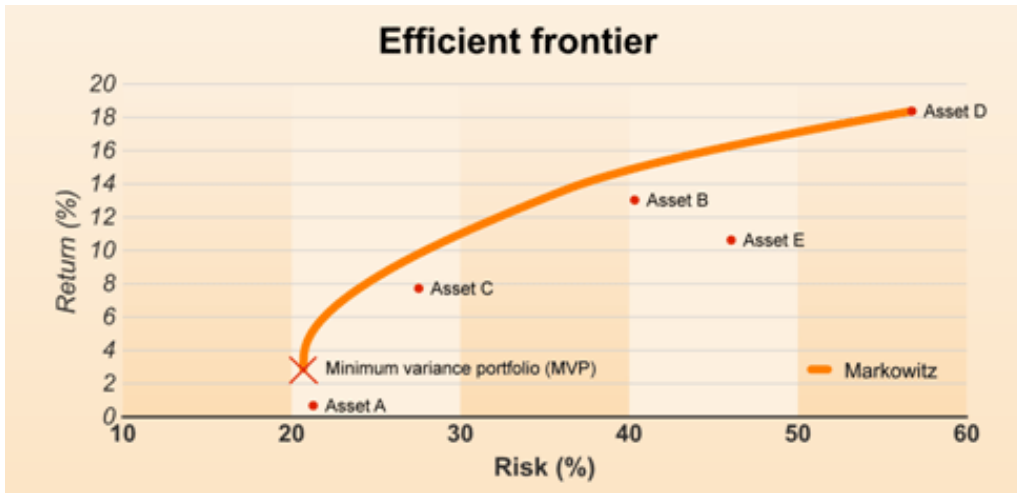**Figure 3.1:** Example of an Efficient Frontier Without a Risk-Free Asset [27].

Each point on the line in Figure 3.1 represents a portfolio which is considered to be efficient (ie no other portfolio provides higher return without more risk and similarly for risk). In Figure 3.1 Assets A-E represent what the portfolio will be made out of.

Markowitz's model is subject to serious criticisms as stated in [32], and the main ones being

that a measure of dispersion can be adopted as a measure of risk only if the relevant distribution is symmetric. Another problem is that the distribution of individual asset returns has a tendency to show a higher probability of being fat-tailed. In case of non-normal, non-symmetric distributions, the utility function must be quadratic [32].

This criteria should not be taken lightly since the assets' return does not follow normal distributions in real world situations [18]. This approach can only be used if the investor's utility function is quadratic with non-positive second derivatives or if the asset's return distribution can be fully described. Several research have indicated that that the quadratic utility function implies that beyond some level or return, marginal utility of the decision maker for wealth becomes negative [3, 10].

3.2 Hill Climbing

Hill climbing [28, 8] is an iterative optimisation algorithm which belongs to the family of local search. The algorithm must begin with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing single elements of the solution. It repeatedly iterates through new solution until no further improvements can be found.

The following pseudo-code is a basic hill climbing algorithm:

currentNode = startNode;

while *not endingCondition()* **do**

    L = NEIGHBORS(currentNode)

    nextEval = -INF

    nextNode = NULL

    for *all $x \in L$* **do**

        if *EVAL(x) > nextEval* **then**

            nextNode = x

            nextEval = EVAL(x)

        end

    end

    if *nextEval $\leq$ EVAL(currentNode)* **then**

        return currentNode

    end

    currentNode = nextNode

end

Algorithm 2: Discrete Space Hill Climbing pseudo-code.

When local search algorithms, such as hill climbing, come across functions, such as $f(x) = \sum_{i=1}^{n} -x_i sin(\sqrt{|x_i|})$ which was considered [38] as a particularly hard function to optimise due to the number of local minima increases exponentially with the number of dimensions of the searching space, they very quickly and easily get trapped in a local optima. This is absolutely not acceptable, specially when there are hundreds of thousands of pounds at stake.

There are some improved hill climbing implementations [13] where they force more randomness into the algorithm making it behave more like a genetic algorithm. The possibility of becoming stuck in a local optimal instead of the global is the biggest reason why I would choose PSO and other evolutionary algorithms over hill climbing.

Scalability is another major factor when considering an optimisation algorithm. Hill climbing

will just not be able to cope with the amount of dimensions that the portfolio optimisation function belongs to. These are the biggest reasons why I have chosen not to use Hill climbing or similar local search algorithms.

Chapter 4: Problem Domain

In this chapter I will discuss some reasons why I believe this is a worth while project. I will mention what methods I intend to implement and for what reasons. It will then follow by a brief section describing my approach, although the full description of the actual approach taken is document in Chapter System Design and Architecture.

Particle Swarm Optimisation has already been effectively implemented to solved various optimisation problems [29, 12, 33] in variant domains such as biomedical, networks, clustering, finance, combinatorial optimisation and many more [25].

This project mainly focuses on expanding the implementation created by Rabanal, Ro-drÄśguez and Rubio in their paper "A Functional Approach to Parallelize Particle Swarm Optimization" and more importantly, its application to solve the portfolio optimisation problem [20]. The optimisation algorithm has been successful by means of accuracy as well as efficiency [26], and when millions of pounds are at stake, choosing the right portfolio to invest in is extremely important, meaning that choosing the right algorithm is even more crucial.

Why PSO. I have chosen to use PSO as it is better than other familiar optimisation techniques such as the simplex method or hill climbing. It does not risk becoming stuck in local optima like the others. It out-performs the others with incredible efficiency when the optimisation function is over higher dimensional fields.

Why Portfolio. It is a fundamental problem in finance, making it an ideal candidate to test PSO's capabilities. As already mentioned, there is a lot of money at stake, so making tools which are efficient to help humans make these multi-million decisions would definitely be worth the hard work. After some brief searching, I noticed that there is not that much information in this field out there, making this a novel and exciting topic. This two-sided coherent risk measure is a well defined function and compatible with PSO function format.

Why Haskell. Firstly, Haskell is a passion for me, it has scared so many yet intrigues a few. Apart from its syntactical charm (subjective I know), there are many benefits, specially when working with mathematical applications. Haskell has been given by default to work with arbitrary large number calculation, this will help against overflow problems which are ever so common with most languages. The functional nature of Haskell makes it effortless to formulate complex mathematical equations [26]. Other extra influences are that it has a great online community, very friendly and helpful, or lazy evaluation and function and partial function composition.

4.1 Approach

This is a research project aimed at designing and evaluation a possible system to solve the portfolio optimisation problem through the use of particle swarm optimisation. A possible expansion to the PSO implementation [26] will also be considered and attempted. The new concept for improving the PSO implementation might be through the use of constriction factors and self-termination described in Chapter 8. Just as importantly as the optimisation method PSO, is was formulate the

portfolio optimisation problem. I will follow closely a revolutionary way to measure a portfolio's risk [4] written by Zhiping Chen. Although Chen completely rethinks the concept of measuring risk by taking into account both upside and downside risk, some work would have to be done to be able to turn this into a function which PSO will be able to handle as well as take into account the constraints which have to be satisfied.

As this is a research project for testing whether it is possible to solve the portfolio optimisation problem through PSO, the application will use simulation (offline) data instead of real-time streams. It is not too much a stretch of the imagination to be able to implement real-time processing as all the financial information is based on previous data. Changing a function, expected return for example, to recalculate its value after new data has come through would be trivial.

One of the main novelties of this project is the use of penalty functions in addition to the two-sided risk measure. The use of a penalty means one can formulate various constraints into a way that can be implemented for the PSO to understand. We use a penalty value to penalise any deviation from the constraint. This also means we need a way to turn deviation from the constraint into a distance measure, this is straight forward given the constraints are linear.

For further implementation and design details, including the reasoning behind the decision, please refer to System Design and Architecture.

Chapter 5: Requirements

This chapter describes the requirements for this project. Table 5.1 refers to the functional requirements from technical point of view. Section Non-functional focuses on the non-functional requirements of the system.

5.1 Functional

No.	Description	Priority
1	**Optimisation of Portfolio**	
1.1	**PSO**	
1.1.1	Initialisation of particle population	High
1.1.2	Processing swarm optimisation	High
1.1.3	Updating the local and global (at each step) particle values	High
1.1.4	Calculating an optimal solution	High
1.1.5	Presenting the results	Medium
1.2	**PSO for portfolio problem**	
1.2.1	Minimise portfolio variance	High
1.2.2	Maximise portfolio expected return	Medium
1.2.3	Use multi-objective for optimum solution	Low
1.2.4	Refining results output	High
1.2.5	Make results for readable for user	High
2	**User Input**	
2.1	Allow the user to enter the name of the data file	High
2.2	Allow the user to change the expected portfolio return	High
2.3	Allow the user to select the name for the output file	Medium
2.4	Allow the user to change the PSO particle size	Low
2.5	Allow the user to change the PSO iteration number	Medium
3	**Output format**	
3.1	Display the results during run-time	Medium
3.2	Make results more readable for output file	High
3.3	Store results into a separate file	High

Table 5.1: Functional requirements for system.

5.2 Non-functional

As this system is an extension on a PSO module [26], it is crucially important to devote a considerable amount of time to testing. This is to ensure that the alterations do not affect the performance of the overall efficiency of the algorithm and quality of the optimisation.

The system's scalability is something not to be overlooked. As each asset in a portfolio

represents one dimension in the fitness function (not to be confused with just another linear factor of the same coefficient in a function), optimising a function in, for example, 100 dimensions (100 assets) might be to much for the system to cope with.

Running the PSO requires setting up various parameters and thresholds for optimisation (size of the particle population, number of iterations, inertia weights and convergence coefficients). These parameters need to be optimised for the algorithm to be computationally effective and produce accurate results.

Chapter 6: Risk Assessment

Section 6.1 and Section 6.2 describe possible risks which might affect the development of my project and some worries which must not be overlooked.

6.1 Social Risk Assessment

I do not refer to the term 'Social' as a worry that I will be too sociable and not focus enough on my project. I simply mean "stuff not directly to do with my project area". The first thing that worries me is that I am reading a joint-honours degree in mathematics and computing science, almost all of my colleagues have nothing to do this term except their dissertation, where I have two more modules as well was this project. Most of them where smart enough to take an extra "meaningless" module last term so they did not have to do anything else this term. I was unable to do that as I had to do my Mathematics dissertation and 3 other modules last term.

Doing a join-honours degree has meant I have gotten much more out of studying here than most people but it also means I have had to learn (in terms of quantity) much more material. I worry that having to do these other two mathematics modules this term might affect my dissertation or *visa-versa*. I plan to work as long as I can on all subjects even if it mean burning the candle at both ends as it were. Careful planning of my days will help and I hope it will be enough to achieve the grades I desire.

6.2 Project Based Risk Assessment

6.2.1 Haskell

The nature of Haskell, as mentioned in Section 2.3 in Chapter 2, is strongly typed meaning that the type of a function (ie input an int and return bool) is very important. Because of this, expanding the PSO implementation by [26] will be much more work that on other less typed languages. Changing just one tiny part of a Haskell function will meaning altering any function which use this function. Fortunately, you can define partial functions and use these for partially giving function inputs as a sort of mathematical abstraction. This is very useful for defining function to be used as inputs for the PSO.

6.2.2 PSO

One thing that is a worry whenever trying to optimise any function is the possibility of becoming stuck in a local optimum value, and then mistaking that as the global optimal best solution. Randomness is key when dealing with such situations. This solves this problem by introducing randomness not only to each particle's momentum but also to the initialisation of the swarm itself. This ensures that the swarm is able to search the whole domain and then determine the optimal value.

When evolutionary algorithms come across functions, such as $f(x) = \sum_{i=1}^{n} -x_i sin(\sqrt{|x_i|})$ which was considered [38] as a particularly hard function to optimise due to the number of local minima increases exponentially with the number of dimensions of the searching space, the danger might

be that when it finds a certain local minima which might seem like a "good enough" solution, then the algorithm might converge to this false solution. Furthermore when what we are trying to optimise has local minima and global minima which differ by thousands of pounds worth of stocks when applied to financial situations it is crucial the algorithm does not get stuck in non-optimal solutions.

6.2.3 Portfolio

The current model has no notion of financial markets or knowledge of social economic climates, when the algorithm is given a portfolio to optimise, it will not look for or know of any problems that a company might be going through. It might be obvious that a company is going to crash any minute from now but if the system thinks, from previous financial data (and nothing more), that the company seems profitable then it will tell the user to invest in the company. One way to help this problem is introducing semantic web engineering into the system, this is not something I will be able to do or even look into in this project but the idea of it does seem both plausible and worth worth while for future developments.

6.2.4 Finance

This project required large amounts of finance background or knowledge. I have never studied or read any sort of financial theory or concepts. A worry is that I will have to learn more than the ideal amount when doing a short term project such as this, made even shorter by the amount of other modules I have to study at the same time, not to mention the time spent dealing with what to do in the future, ie. applying for jobs, masters etc..

Chapter 7: Methodology and Technologies

This chapter describes the methodology used in the project for the research, design, implementation and testing. It also mentions the technologies used to achieve the goals.

7.1 Methodology

This sections is basically an extension to the project plan which had to be made during the first week of the project. An approximate guideline to follow the project was set focusing on the project deadlines. I left a few weeks for margin for error in case something takes longer than planned, Figure 7.1 shows the plan.
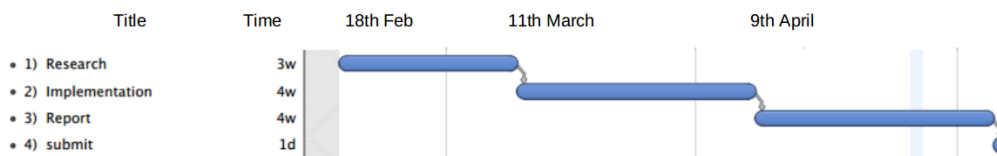

Figure 7.1: Time line for project completion.

For this project to be successful I am planning on spending the initial weeks researching relevant literature and becoming familiar with the concepts of Particle Swarm Optimisation. This is a completely new field to me and understanding the key ideas and models will be critically important. Not only will I need to understand PSO's background I will also need to study previous implementations and applications in order to become absolutely comfortable with it. Finally, as I am planning on improving an existing algorithm, I will have to spend some time becoming familiar enough with the code so that I will be able to modify it with ease.

The implementation stage will consist of designing the future system and the realisation of the plans. Key design decisions will have to be made during this stage and the solutions might be obtained from the analysis of previous work.

To complete this project test driven development will be carried out. I plan to test after every implementation or modification. This will be done to ensure that changes will not affect any previous functionality. The tests will evaluate the efficiency as well as the accuracy of my system. Given the nature of PSO's 'random' initialisation, I want to make sure that the results are consistent.

The writing of this result will be flexible, the sections will be written as needed or when the section arises naturally throughout the project.

7.2 Technology

7.2.1 Haskell

Coming from a strong mathematical background I find functional languages easier to understand. Also one huge advantage of pure functional languages is that the absence of side-effects allow

them to offer a clear semantic framework to analyse the correctness of programs.

As Haskell is the functional language I am more familiar with, I did not see the point in learning a new language as it would only restrict my project process, so Haskell was a clear winner.

There are other PSO implementations in other languages such as C and Ruby but as already mentioned, Haskell is my preferred language.

7.2.2   Operating System

As Haskell is platform independent (in the sense that it can be compiled in Windows, Linux or Mac) I have chosen to use Ubuntu 12.04 as it is my preferred OS and I feel the most comfortable with it. In addition, I would not be affected in the about of software needed for the project as it is provided for all three OSs already mentioned.

The work was carried out on my personal laptop (Intel CORE^{TM} i3 @ 2.6GHZ,4Gb RAM). If required due to any reasons, the university provide classroom PCs (Intel CORE^{TM} i3-2100 CPU @3.10 GHz, 3 Gb RAM) although I have faith that my own machine will be reliable enough for me not to have to change machines.

Sublime Text 2 was chosen as the IDE for the project. It has many useful functions [31] and similarly for the choice of OS, I am happy with this editor.

Chapter 8: System Design and Architecture

The following chapter describes the system's basic design and its program flow. As this project is partly an extension of an implementation of PSO [26], Section 8.1 outlines the core architecture of that implementation in order to provide a base to the changes made in this project. Section 8.2 explains the design and ideas behind the system's modification and further design on how to apply the algorithm to solve the portfolio optimisation problem.

8.1 Original PSO Implementation

8.1.1 Initialisation

The main purpose of this stage is to create a population of particles (called a swarm) which will be used to search the domain and find the optimal solution to our problem. The application uses a function called *initialise* to randomly create a list of initial particles to populate the search space. It is written in accordance with Algorithm 1 and thanks to Haskell's well designed and efficient random generators [21] it ensures that the particles are initialised in such a way to exploit the search space well. For details, see Algorithm 1. After the population has been initialised, the algorithm moves on to the optimisation phase.

8.1.2 Inertia Weights

PSO relies on randomness, as in Equation (2.2). As this project is not intended to experiment much with the basic PSO parameters we will use the inertia weights presented in [23]. If there is enough time, we may wish to experiment with different weights. This does not mean that testing will not be done, it just means that we will not deviate from our goal to look into this matter, as so much research has already been done[2].

8.1.3 Optimisation

This is done by "steps", at each "step" a particle is updated by a function *updateParticle* in accordance to Equation (2.2)which changes the position and velocity. The same function checks whether the new position is better that the best solution so far. A boolean system is used, if the new position of the particles is better that global best position, if it is then it changed the global best to this new global best. This is done recursively until all the particles in the swarm have been updated. Function had to be created to deal with the comparison of particles' results and both the position and the value of that position are stored for each particle.

For any more details on how the optimisation is done, please refer to Particle Swarm Optimisation in Chapter 2.

8.1.4 Termination

A function was created to deal with the termination, a user specifies the number of iterations ("steps") that the algorithm is allowed to run for. This independent function is simple, it runs recursively, applying one *updateParticle* at a time and terminates once there are no more iterations to be performed and returns both the value of the global best and its position.

8.2 Expansion for Portfolio Optimisation

8.2.1 Interface and User Input

The original implementation needed the swarm size and number of iterations to be hard-coded into the algorithm. This model allows the user to specify how many particles they would like as well as how many iterations the PSO should run for. I do not think the user should be allowed to set the inertia weights as they might not fully understand their purpose and therefore give non-optimal inputs.

8.2.2 Data file

The user will have to input a file with the assets' information in order for the system to optimise a portfolio. There is no standard format for such a file and it really is up to preference. I have decided to arrange the file as follows: "Name Rate-Of-Return Expected-Required-Return", where *Name* is the companies name (either full or just the ticker), *Rate-Of-Return* is the rate of return for the company and *Expected-Required-Return* is the expected required return of the same company with each company on a different line.

Figure 1 shows an example file containing a selection of assets which the user wants to maximise for their portfolio. As you can see the first item is the name of the company, followed by a single space and then the rate of return, another space and the expected return. Each company has their own line and they can have as many companies as they would like.
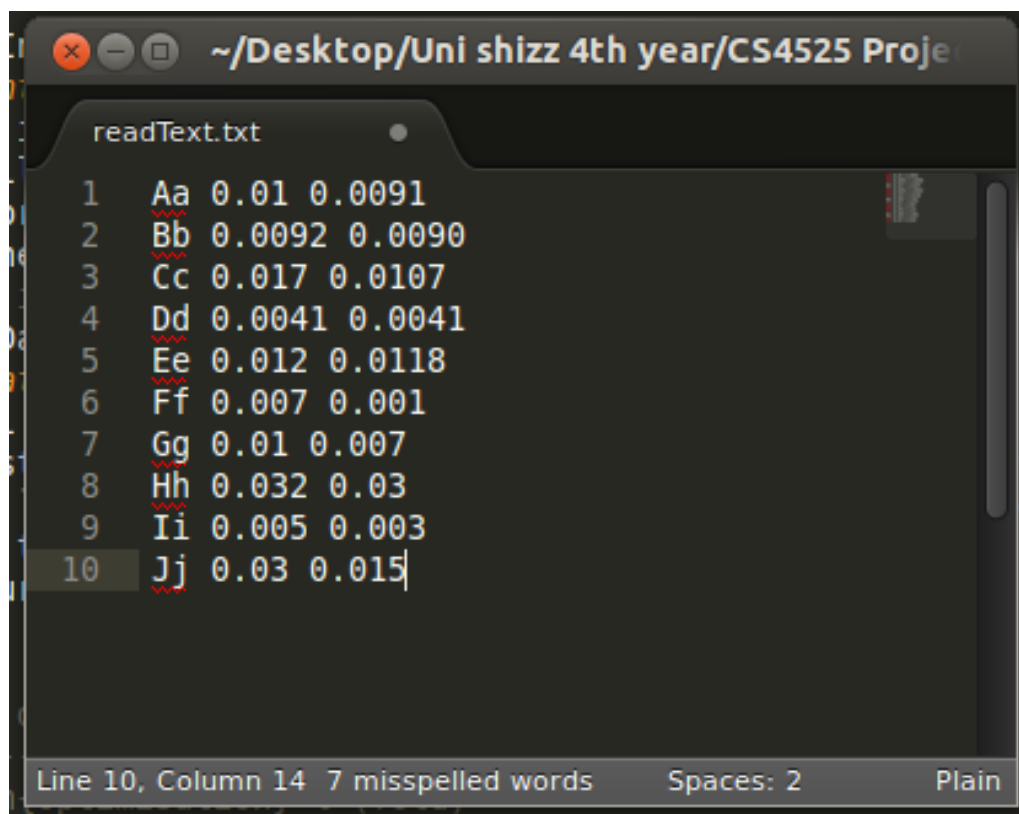

Figure 8.1: Example File of Assets.

8.2.3 Constriction Factor

Maurice Clerc in his study on stability and convergence of PSO [5] has introduced the concept of a constriction factor. Clerc indicated that the use of a constriction factor may be necessary to insure convergence of the PSO for certain fitness functions.

In order to ensure convergence of the PSO, the velocity from Equation (2.2) can be expressed as follows:

$$V_i^{t+1} = K \left[V_i^t + c_1 r_1 \times \left(Pbest_i^t - X_i^t \right) + c_2 r_2 \times \left(Gbest^t - X_i^t \right) \right]$$

where   (8.1)

$$K = \frac{2}{\mid 2 - \phi - \sqrt{\phi^2 - 4\phi} \mid} \text{ and } \phi = c_1 + c_2 \text{ s.t. } \phi > 4$$

The convergence characteristic of the system can be controlled by ϕ through choosing suitable c_1 and c_2 in (8.1). In this approach, ϕ must be greater than 4 to guarantee stability [16].

8.2.4 Termination

If time allows, it would be nice to have a self-termination PSO which, no matter how many iterations are left, terminates if it thinks that the optimal solution is reached. This could be done by the concept of a threshold, if all the solution reach this threshold, then there is no point in carrying on. For example, if all the solution are close enough to each other for a long period of iteration, say 10^{-10}, then there is no point in looking for thousands more iterations just to find something which will not affect the outcome.

In order to achieve this, we would need three extra parameters to be added, one to define the threshold, another one to define the maximal number of iterations allowed without improvement (call this *mni*, and another one to compute the current number of remaining iterations allowed without improvement(call this *cni*. After each call to the updating function, compare the current best solution to the past best solution to decide whether there has been a relevant improvement or not. The algorithm would either change the number of iterations to *mni* (in case there was relevant improvement), otherwise reduce the value of *cni* by one.

8.2.5 Fitness Function

Due to the nature of Haskell and with some mathematical background it is almost trivial to create a function to represent the measure of risk, namely $\rho_{a,p}(R)$ in Equation (2.4) in Section Portfolio Optimisation Chapter 2. What is much more challenging is finding a way to include the constraints in Equation (2.6) in a way that the algorithm can deal with the main function without violating these condition. Two methods were considered, using multi-objective optimisation or a penalty function. Due to the constraints being linear-independent, it will be straight forward to affect the optimal solution through factors of the constraints. The design of this will be described in the following Subsection 8.2.6.

8.2.6 Penalty function

One of the most straight forward approaches to solving constrained optimization problems is the use of a penalty function[9]. Using such an approach means transforming the optimisation problem, such our $\rho_{a,p}(R)$ in Equation (2.4), to include the behaviour of being "punished" if the constraints are not met. One can construct a single new objective function from combining $\rho_{a,p}(R)$ plus linear factors of each constraint in Equation (2.6), this new objective function will be used to find an optimum value for the portfolio problem. Thus a new optimisation function can be created:

$$\min f(R)$$
$$\text{where } f(R) = \rho_{a,p}(R) + \frac{1}{\delta}|\widehat{R} - \eta| + \frac{1}{\delta}|\sum_{i=1}^{N} x_i - 1| \tag{8.2}$$

Now an explanation on what is happening in Equation (8.2). Firstly, $\delta > 0$ is called the penalty parameter and $\frac{1}{\delta}$ is called the penalty value [9]. The penalty parameter should be small enough so that whenever a constraint is violated, the penalty value will be big enough to affect the result to the solution. Secondly, the use of absolute values refers to the distance traveled from the desired value, for example if the sum of the weights is not equal to 1 then $|\sum_{i=1}^{N} x_i - 1| > 0$ multiplying this by a penalty value will be ensure that the solution will not be taken as an optimal one. Finally, similar method in used for deviating from the required expected return.

One may notice that in this new optimisation function, the lower and upper bounds for how much one can invest in each asset is not included. This is due to how PSO handles search space domains and will be explained in the following Subsection 8.2.7.

8.2.7 Forced Diversification

Equation (8.2) did not include lower and upper limits for each assets as in Equation (2.6), this is because the implementations [26] already takes into account bounds for the solutions' domain for a given fitness function. Since our objective is to find weights for each asset, the set of assets is the domain for our fitness function. So the bounds for the proportions can be set which the algorithm.

The bounds inside PSO can be used to give a sense of diversification [35], by not allowing how much you invest on each asset to be too small or to large. For example, you might specify that you have to invest at least 5% and at most 50% of any asset in a portfolio, this forces the investor to consider at all assets and helps lowering the risk of a portfolio [35].

8.2.8 Presenting the Results

After the algorithm has finished, the results are displayed on the screen and saved in a separate file which will be called "output-DATE", where date is in (year,month,day) format, for example "output-2014,3,15". If there is multiple runs in one day, the new results are added to the end of that same file.

Chapter 9: Experimentation and Testing

This chapter shows the results of some of the tests I made, during the implementation I tested every function independently and then tested the full program after all the sub-programs where merged together. This project is not primarily about testing the PSO algorithm, it's more about testing whether we can use it to solve the portfolio selection problem and whether it's safe to use.

I condensed greatly the amount of data collected. I ran many tests, within each test there were many sub-test and each sub-test was ran 30 times. From this I calculated means and standard deviations which are used to give readable results. Overall there are about 7 files which represent test cases, each file contains the results for the sub-tests all grouped together and these files are each about 500 lines long.

9.1 Scalability

Unfortunately, I hindered the system by insisting that the minimum percentage of any asset in the portfolio is 5% and the maximum is 35%. This means that after 20 assets, the system stops giving credible results, as you can notice, if we have 21 assets and each one has to be at least 5%, then we will end up have 105%, which of course makes no sense. It will be essential to implement a better diversification technique if I plan to consider a large amount of assets.

This problem has nothing to do with the PSO algorithm, the algorithm has proven to be efficient even around 60 dimensional search spaces, this depends on the nature of the optimisation function of course.

9.2 Efficiency

Standard Deviation stuff, box plots blahh blahhh

9.3 Precision

These tests should verify that the system is stable, by this I mean that given the same input, the output should be almost the same. A there is a lot of randomness in the algorithm, one can not expect the outputs to be equal, but they should be equivalent up to some threshold. In this case the threshold will be 1.0×10^{-8}, this small enough to not affect the expected return from a optimised portfolio.

9.3.1 Testing Strategy

I will run nine tests and record the expected rate of return for the solution the algorithm gives. Each of the nine tests will be run 30 times with a mean and standard deviation calculated and shown in the following sections. The nine tests will be as follows:

- Test 1: PSO 20 particles, 100 iterations, 5 assets

- Test 2: PSO 20 particles, 300 iterations, 5 assets

- Test 3: PSO 40 particles, 100 iterations, 5 assets

- Test 4: PSO 20 particles, 100 iterations, 7 assets

- Test 5: PSO 20 particles, 300 iterations, 7 assets

- Test 6: PSO 40 particles, 100 iterations, 7 assets

- Test 7: PSO 20 particles, 100 iterations, 10 assets

- Test 8: PSO 20 particles, 300 iterations, 10 assets

- Test 9: PSO 40 particles, 100 iterations, 10 assets

9.3.2   Hypothesis

My hypothesis is that the application is reliable and that the results (for the same input) will have almost the same output, ie it is reliable and precise, up to the threshold already stated.

9.3.3   Results

Table 9.1 show the results for the 9 tests.

Test	Mean Result	Standard deviation
Test 1	0.0447	6.13954×10^{-11}
Test 2	0.0447	1.58041×10^{-17}
Test 3	0.0447	7.18283×10^{-13}
Test 4	0.0527	3.3924×10^{-10}
Test 5	0.0527	1.41115×10^{-16}
Test 6	0.0527	6.83323×10^{-13}
Test 7	0.101475	0.00279292
Test 8	0.1007	2.30303×10^{-12}
Test 9	0.1007	2.56085×10^{-11}

Table 9.1: Results for Precision, expected return.

As you can see from Table 9.1 all of the standard deviations are less than 1.0×10^{-8} which was our threshold, except Test 7. This is a very good sign as we have only given PSO 100 iterations to complete its task and when we increase this to 300, the results are by far below the threshold.

The only reason Test 7 did not pass is because increasing the number of assets to 10 whilst only letting the algorithm run for 100 steps and only 20 particles was just to much. Remember that each asset represents a dimension, so in Test 7, the algorithm have 100 steps with 20 agents to search for an optimal value in a 10 dimensional search space.

9.3.4   Conclusion

From the results and our test strategy, we can conclude that the algorithm is reliable by means of precision. To be safe, we will have a minimum cap of 100 particles and 1000 iterations. This will be enough to ensure consistency for the average investment manager.

9.4   Constriction Factors

In System Design and Architecture 8.2.3 the concept of a constriction factor was introduced.

9.4.1 Testing Strategy

I plan to test whether this will in fact affect the outcome of the algorithm when applied to the portfolio optimisation problem, furthermore if it does affect it, then whether it improves or worsens the result. In order to test this I will conduct six different experiments, three without a constriction factor where one has the adjustment parameters taken from [23], one with the two randomness coefficients that add up to less than 4 and one where they add up to more than 4, the importance of 4 can be read in [5]. Then three more with a constriction factor and the rest is the same as the previous three.

All tests will run 20 times and the results and the time taken will be recorded, all tests will be set to 50 particles, 500 iterations and 10 assets. A mean result and standard deviation will be computed to be able to compare the results. This will give enough indication on how the constriction factor affects the fitness function and how it differs under various criteria.

9.4.2 Hypothesis

Constriction factor when applied to the portfolio selection problem with appropriate coefficients will not improve the portfolio selection problem.

9.4.3 Results

This subsection shows the results and the following Table 9.2 contains the exact values for the results in my experiment, it will follow a short explanations of the results.

Firstly, the time it took for each test to run and there was no significant difference between them. Having a constriction factor did not affect the time it takes for the algorithm to complete.

Test	Mean Result	Standard deviation
WO-CF Pefersen	0.914099	$1.61088 \times 10^{--16}$
WO-CF < 4	0.914099	1.72748×10^{-16}
WO-CF > 4	0.9141	5.36229×10^{-6}
W-CF Pefersen	0.91415	0.0000389374
W-CF < 4	0.91416	0.0000448388
W-CF > 4	0.914099	2.81328×10^{-16}

Table 9.2: Results for Constriction Factors.

———————— Box-whisker plot ————————

Table 9.3 shows what the acronyms in Table 9.2 mean.

WO-CF Pefersen	: Without constriction factor and Pefersen coefficients
WO-CF < 4	: Without constriction factor and $\phi < 4$
WO-CF > 4	: Without constriction factor and $\phi > 4$
W-CF Pefersen	: With constriction factor and Pefersen coefficients
W-CF < 4	: With constriction factor and $\phi < 4$
W-CF > 4	: With constriction factor and $\phi > 4$

Table 9.3: Key for Results for Constriction Factors..

The reason for using Pefersen coefficients is stated in the design and architecture section of this report, Constriction Factor in Original PSO Implementation. One can see from Table 9.2 that importance having $\phi > 4$ when introducing the constriction factor, in W-CF Pefersen and W-CF < 4 one can see a serious decrease in the optimum found.

What is interesting here is that PSO for the fitness function is efficient and consistent without the constriction factor as shown in Table 9.2 where WO-CF Pefersen and WO-CF < 4 both have the same mean and almost exact standard deviation, meaning they behave the same. Once we introduce the constriction factor, it is almost catastrophic if we do not have $\phi > 4$, as both W-CF Pefersen and W-CF < 4 have less efficient means and huge standard deviations (in comparison to the other tests) meaning they are unstable and unreliable. Once we make $\phi > 4$, the algorithm settles back to normal but does display higher standard deviation.

9.4.4 Conclusion

Constriction factor when applied to the portfolio optimisation problem does not improve the results as in the hypothesis. It makes the results more unstable, the algorithm will therefore not include a constriction factor.

The use of the constriction coefficient can be viewed as a recommendation to the particle to "take smaller steps"[7], because of this it exploits optimal solutions, which is fine if there are not many, but it does mean that it travels less in the same amount of time. This is one of the main reasons why it did not improve the results. One has to bare in mind that each asset adds a dimension to out search space, this increases domain exponentially so as you increase the amount of assets in a portfolio coupled with making the PSO take smaller steps results in a much larger search space and less area covered by the end of the algorithm.

9.5 Penalty value

This experiment will see what happens with different penalty parameters. The penalty value, as explained in Section 8.2.6 from Chapter 8 is what ensures that the constraints in Equation (2.6) are not violated. If the penalty is too small, it will not ensure that the constraints are kept, but if it is to large, then it might not let the algorithm settle on the optimal solution. The aim is to find a suitable penalty parameter.

9.5.1 Testing Strategy

There will be 3 tests, each test will be run 30 times and an average and standard deviation calculated. This should ensure enough range to be able to find an adequate value.

- Test 1: PSO 100 particles, 1000 iterations, 10 assets, Pen Val = 0.01

- Test 2: PSO 100 particles, 1000 iterations, 10 assets, Pen Val = 0.1

- Test 3: PSO 100 particles, 1000 iterations, 10 assets, Pen Val = 1.0

9.5.2 Hypothesis

Induced suspect that if the penalty value is to high, then the fitness function will be penalised to severely resulting in more 'erratic' behaviour, it will be hard for the algorithm to settle down and find the global optimum. On the other hand, if it is too low, it will not penalise the function enough when it deviates from the constraints and may return optimal values which are incorrect.

9.5.3   Results

As you can see from Table 9.4, having the penalty parameter equal 1.0 in Test 3, is too big, resulting in a non-optimal solution by quite a bit although, from the standard deviation we can see that it is more stable (consistence/precise) than the other tests.

Test	Mean Result	Standard deviation
Test 1	0.100699	3.5×10^{-13}
Test 2	0.100699	3.5×10^{-11}
Test 3	0.100669	2.7×10^{-15}

Table 9.4: Results for Penalty value.

The question now is whether it is better to have a penalty parameter of 0.1 or 0.001. Test 1 vs Test 2: they have the same mean but Test 1 has a smaller standard deviation. This means that both Test 1 and Test 2 provide in average the same result, but Test 1 is more consistent.

9.5.4   Conclusion

For the reasons presented in the results, having the penalty parameter at 1.0 is too big, having it at 0.01 is too small and thus the penalty parameter will be held at 0.1. This should ensure consistent and reliable optimal solution to our portfolio optimisation problem.

9.6   Parameter Investigation

This test might seem a little peculiar at first but be assured, there is method in my madness. I want to see what the relationship is (if any) between the time it takes to run, the number of particles, number of iterations and number of assets.

9.6.1   Testing Strategy

I will run nine tests and record the time it takes to finish and the results just to make sure they are consistent. Each of the nine tests will be run 20 times with a mean and standard deviation calculated and shown in the following sections. The nine tests will be as follows:

- Test 1: PSO 20 particles, 250 iterations, 5 assets

- Test 2: PSO 20 particles, 500 iterations, 5 assets

- Test 3: PSO 40 particles, 250 iterations, 5 assets

- Test 4: PSO 20 particles, 250 iterations, 7 assets

- Test 5: PSO 20 particles, 500 iterations, 7 assets

- Test 6: PSO 40 particles, 250 iterations, 7 assets

- Test 7: PSO 20 particles, 250 iterations, 10 assets

- Test 8: PSO 20 particles, 500 iterations, 10 assets

- Test 9: PSO 40 particles, 250 iterations, 10 assets

9.6.2 Hypothesis

There is no hypothesis here, I hope that there is no exponential relationship between increase in assets to increase in time for PSO to finish. The point of this exercise is to see if anything interesting happens.

9.6.3 Results

Table 9.5 show that even after 250 iterations, the algorithm already meets the constraints which where applied though a penalty function. This is indeed a promising result.

Test	Mean Result	Standard deviation
Test 1	1	0
Test 2	1	0
Test 3	1	0
Test 4	1	0
Test 5	1	0
Test 6	1	0
Test 7	1	0
Test 8	1	0
Test 9	1	0

Table 9.5: Results for Parameter Investigation, sum of weight.

Table 9.6 show even the most computationally demanding process of optimising in 10th dimensional space takes less than a fifth of a second. This is using a standard dual-core machine. I imagine using a specialised server will make the algorithm much faster.

Test	Mean Result	Standard deviation
Test 1	0.0463947	0.00487954
Test 2	0.107632	0.00912289
Test 3	0.0812466	0.00542592
Test 4	0.0555938	0.00491575
Test 5	0.1253	0.00699414
Test 6	0.106167	0.00669427
Test 7	0.089034	0.00856579
Test 8	0.181696	0.00828043
Test 9	0.17403	0.00557404

Table 9.6: Results for Parameter Investigation, time to finish.

Table 9.7 show the expected return results for the optimal portfolio selected by the PSO algorithm.

Test	Mean Result	Standard deviation
Test 1	0.0447	6.13954×10^{-11}
Test 2	0.0447	1.58041×10^{-17}
Test 3	0.0447	7.18283×10^{-13}
Test 4	0.0527	3.3924×10^{-10}
Test 5	0.0527	1.41115×10^{-16}
Test 6	0.0527	6.83323×10^{-13}
Test 7	0.101475	0.00279292
Test 8	0.1007	2.3030×10^{-12}
Test 9	0.1007	2.56085×10^{-11}

Table 9.7: Results for Parameter Investigation, expected return.

9.6.4   Conclusion

As already mentioned, this section is intended to find anything (if any) interesting. The first thing I notice is the disproportional change in the results when increasing the number of iterations to the number of particles. By this I mean that Test 1 in Table 9.7 has the same mean at Test 2 and Test 3, but the difference in output is not proportional to the change in input. From Test 1 to Test 2 and increase of doubling the number of iterations results in an decrease of about 50%, whereas the difference between Text 1 and Test 3 is doubling the number of particles in a swarm which results in a decrease of around 20%.

This leads me to think that if one had to between a large swarm or a large number of iterations for the algorithm, a larger number of iterations will be more beneficial.

Something which might not be completely straight forward is how increasing the number of particles increasing the time it takes for the algorithm to finish, even though it is based on the number of iterations. The reason for this is that at each iteration, the algorithm has to update every particles, meaning that if there are more particles, each iteration takes longer to complete, thus the whole algorithm takes longer to finish.

Chapter 10: Future Work

10.1 PSO

This section mentions some future improvements or suggestions to improve the particle swarm optimisation algorithm.

10.1.1 Inertia weights

Something to look into for the future, is whether changing the inertia weights will improve PSO on solving the portfolio optimisation problem. Some testing will need to be made to find the optimal inertia weights. Other things to consider is dynamic inertia weights as introduced in [1] or possibly decreasing inertia weight as in [2]. It states that a weight of 1.9 encourages a better global search but a weight of 1.5 would be better for exploiting optima, having a continuously decreasing inertia weight from 1.9 to 1.5 might provide good guidelines for PSO to find all optima and then exploit the global optimum once the inertia weight settles.

10.1.2 Parallel Programming

One of the benefits of Haskell – Functional stuff... Yet another benefit of Haskell is that through function abstraction [26] one can split the PSO algorithm into different processes. What takes the longest in PSO is updating each particles after every iteration, through parallel processes one can split the updating process into different sub-processes and computing them in parallel, improving the speed even more. Eden [17, 11] is a parallel extension for Haskell, this would be what I would choose to implement parallel processing into the system.

10.1.3 Self-termination

This might be beneficial for making the algorithm more efficient. Why should the algorithm run for x amount of iterations if it will not improve the solution. There is no reason what so ever, the counter-positive though, might be crucial when calculating a portfolio's risk. There is the possibility that the share price has changes by the end of the algorithm. This also applies to the following Section Asset's covariance and Real-Time Processing when discussing real time processing. I have considered this method and described how it can be done in Chapter 8 Section 8.2. Please refer there for further details.

10.2 Asset's covariance and Real-Time Processing

This section mentions some future improvements for measure risk and calculating an optimal portfolio.

10.2.1 Covariance

Asset covariance refers to a measure on how two assets' return change together. It as a number which determines that if asset A changes, then asset B also changes in accordance to asset A. This is a powerful thing, if one hold some assets in a portfolio and one knows what the relationship is between their expected return, one can maximise the portfolio return. For example, if we knew

that whenever asset *A* increases value then asset *B* also increases value, then when we notice that one is increasing, we can start buying both as we know that they will both increase. Similarly if one starts to decrease in value, we know they both will, so we can act quickly to the dynamic market changes.

10.2.2 Real-Time Processing

Real-time processing has never been so crucial to market efficiencies and competitiveness as it is today. To operate in the highly dynamic global market and to have an edge over competition, access to real-time data and intelligent processing and correlation of the data has become an absolute necessity in functions such as algorithmic management, smart order routing, pre-trade analytics and risk management. For these reasons, I think that there must be ways to include real-time processing in order to have a more efficient system and less reliant on human input.

10.3 Diversification

One of the most interesting concepts in portfolio theory I found was that of diversification [24], unfortunately I found this very late into my project and unable, due to time constraints, to include this into my application. Diversification excites me as it contradicts intuition. One would think that if one have one risky asset, adding another one would only increase the overall risk further, in fact it does the exact opposite!

The most simplistic model to represent this concept is the proverb, "putting ones eggs in more than one basket". Regardless on what the probability of each egg is, having more baskets with eggs is more likely to preserve more eggs that less baskets with the same amount of eggs. In other words, if one basket crashes, one still have the the other eggs which where in different baskets.

Now something more useful and even less intuitive is that if ones invests in more than one company within the same sector, for example split all ones money equally (for simplicity in example) and invest in all the mobile phone networks there are. Now company *x* gets into trouble for some reason which affects the stock market (fraud, IT, quality etc.) and the stock price for *x* begins to fall, ones will find that the price of stocks for all the other companies goes up. This is due to the investors and business which was with company *x* now deciding to opt out of that company and therefore bringing more investors and business to all the other companies in the market.

My application has a sense of diversification due to the extra constraint which I added late in the project as a emergency diversification solution. It states that one must invest between 5% and 35% on each asset to force diversification. This is vaux or brute intelligence though, what could be useful if the application gives a little extra preference if it knows that some assets belong to the same sector.

Chapter 11: Evaluation and Conclusion

This chapter summarises and discusses the tasks accomplished, providing a critical evaluation of the work that was done and suggesting future development.

11.1 Evaluation

Looking back at the project goals and requirements, it is possible to say that almost all of them were successfully achieved.

- **Understanding the principles behind the Particles Swarm Optimisation algorithm and designing an appropriate expansion to solve the portfolio optimisation problem.** The Particles Swarm Optimisation algorithm, its modeling background, algorithmic concepts and expansion designs were thoroughly studied and described in theory as well as through practical applications. This was an excellent introduction into the field of Swarm Optimisation algorithms and has shown great potential for the algorithms to be used in the financial domain. Although the system was implemented as research to see whether it is possible to use PSO to solve real world problems, the portfolio selection problem in this case, a new design and architecture was laid to form a stepping stone for implementing real-time processing and other essential propertied for the system to be effective in the real world.

- **Test the resulting application and assess whether it can be used in a professional environment.** The testing and experimentation has shown stable and promising results. Firstly, it was verified that the system was stable in a sense of reliable and consistence results. Secondly, the scalability was tested and the results where promising for the PSO but my forced diversification did affect the results negatively when considering 20 or more assets. Finally, it is left to future testing whether it is possible to use real-time processing to improve the system.

11.2 Problems encountered

- **Haskell being strong typed:** One problem I encountered when trying to embed the portfolio risk measure into Haskell was that because of the nature of Haskell, the function's type had to be clear and defined. This caused problems when trying to turn it into a fitness function for the PSO algorithm to understand. The problem was that the fitness function's type had to match what the PSO required. This was overcome by using partial functions composition and list comprehension.

- **Results are too small:** When experimenting and testing the system, the results where incredibly good. Unfortunately, drawing a nice graph for results which has a standard deviation of 10^{-13} is far to difficult for such a simple task. Tables are not as pretty, but they do get the job done.

- **Financial Background:** Prior to this project, I had never studying any type of finance or economics course. The amount of concepts that had to be learn was almost overwhelming. I found online lecture videos and this helped a huge amount, I tried to back-track as much as possible when I heard a new term in order to fully understand the problem and hence be more able to solve it.

- **Project scale:** This isn't necessarily a problem, but the more I learn about financial computing, the more I realise I need to learn more. I wish I had more time to implement the future ideas stated, although I know as soon as I start to implement these improvements, I will find more to implement.

11.3 Discussion

When I first discussed this project with my supervisor Dr. Wei Pang, I had no idea how vast the scope was, I had no idea how big this topic really is. I believe I could do a 4 PHD in this topic and I would not even begin to break the ice. I have had to learn huge amounts of new things. I have never studied any sort of finance or economics course. I have to say, watching many hours of online lectures from various universities helped an incredible amount in understanding complex concepts, as well as basic ones, in financial risk management.

This project has given me a renewed motivation to continue studying as well as directions of what to study. I believe the two things I need to learn more about if I want to continue improving this system are, more knowledge on asset risk measure and real-time processing.

11.4 Conclusion

In conclusion, this project could be rated as a success. The research was an enlightening introduction into the world of evolutionary algorithms and their applications, specially the principles of the Particle Swarm Optimisation algorithm and how it can be applied in the real-world domain.

Most of the primary and secondary project goals were achieved. The modified PSO algorithm incorporates new principles for solving the portfolio selection problem and lots of designs have been stated on how to further improve this system. Dues to time and other commitment constraints, not all were implemented, but clear designs have been made and shown. The algorithm was tested on financial data and the results were satisfactory.

Many challenges were faced during the design and implementation stages of the improved algorithm; some alternative paths considered, Regardless, this research project has given me a valuable insight into the world of evolutionary algorithms, their principles and applications. The results gained from the testing have given me a more insight knowledge on the algorithm, it has taught me new facts about PSo and confirmed some previous known facts too.

There are plenty of opportunities for future development, I believe that this research project has opened a door for me to create something which I believe might change the way investment companies deal with portfolio selection.

Bibliography

[1] Alireza Alfi. Particle swarm optimization algorithm with dynamic inertia weight for online parameter identification applied to lorenz chaotic system, April 2010.

[2] J.C. Bansal and Et. All. Inertia weight strategies in particle swarm optimization.

[3] W. Breen. Specific versus general models of portfolio selection. In *Oxford Economic Papers*, volume 36, pages 361–368, 1968.

[4] Zhiping Chen and Yi Wang. Two-sided coherent risk measures and their application in realistic portfolio optimization. *Journal of Banking & Finance*, 32(12):2667–2673, December 2008.

[5] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages –1957 Vol. 3, 1999.

[6] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, pages 58–73, Feb 2002.

[7] M. Clerc and J. Kennedy. The particle swarm: Explosion, stability, and convergencein a multi-dimensional complex space. 2002.

[8] Hill climbing Search. Selman, bart and gomes, p.c., 2004.

[9] Carlos A.C. Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. In *Computer Methods in Applied Mechanics and Engineering*, pages 1245–1287, 2002.

[10] M. S. Fekdstein. Mean-variance analysis in the theory of liquidity preference of portfolio selection. In *Review of Economic Studies*, volume 36, pages 5–12, 1969.

[11] M. Hidalgo-Herrero, Y. Otega-Mallen, F. Rubio, and R. Pena. Analyzing the influence of mixed evaluation on the performance of eden skeletons, 2002.

[12] Wei Hongkai, Wang Pingbo, Cai Zhiming, Chen Baozhu, and Yao Wanjun. Application of particle swarm optimization method in fractional fourier transform. In *Image Analysis and Signal Processing (IASP), 2010 International Conference on*, pages 442–445, April 2010.
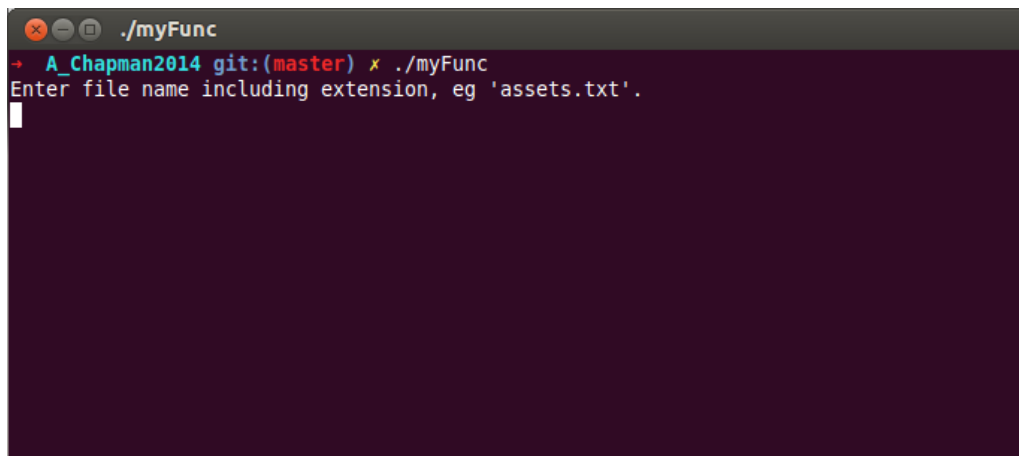
[13] WALKSAT Iterative Improvement Search Hill Climbing, Simulated Annealing and Genetic Algorithms. Moore, andrew, 2009.

[14] L.P. Jones and J. Hughes. Report on the programming language haskell 98. In *Tech. Rep.*, 1999.

[15] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, Nov 1995.

[16] Shi Lim and Mohammad Montakhab. A constriction factor based particle swarm optimization for economic dispatch, 2009.

[17] R. Loogen, Y. Otega-Mallen, F. Rubio, and R. Pena. Parallelism abstractions in eden, 2002.

[18] Billio M. Simulation based methods for financial time series, 2002.

[19] H. Markowitz. Portfolio selection. In *Journal of Finance*, volume 1, pages 77–91, March 1952.

[20] H. Markowitz. Portfolio selection-efficient diversification of investments, 1959.

[21] S.K. Park and K.W. Miller. Random number generators - good ones are hard to find, October 1988.

[22] K.E. Parsopoulos and M.N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, 1:235–306, 2002.

[23] M.E.H. Pedersen. Tuning & simplifying heuristical optimization, 2010.

[24] Krassimir Petrov. Financial institutions, lecture 05. `http://www.youtube.com/watch?v=ntNf5qwrq2c`. Accessed: 21,April,2014.

[25] R. Poli. An analysis of publications on particle swarm optimisation applications. In *Technical Report CSM-469 (Department of Computer Science, University of Essex, UKI*, May 207.

[26] Pablo Rabanal, Ismael RodÃ■guez, and Fernando Rubio. A functional approach to parallelize particle swarm optimization. `http://antares.sip.ucm.es/prabanal/english/heuristics_library.html`. Accessed: February,2014.

[27] Risk Scientist. `http://www.riskscientist.com/content/technologies-portfolio-optimization-theory.html`. Accessed: 4,April,2014.

[28] Richard Senington and David Duke. Combinators for local search in haskell, 2010.

[29] E. Shahamatnia and M.-M. Ebadzadeh. Application of particle swarm optimization and snake model hybrid on medical imaging. In *Computational Intelligence In Medical Imaging (CIMI), 2011 IEEE Third International Workshop On*, pages 1–8, April 2011.

[30] Yuhui Shi and R. Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence.*, pages 69–73, May 1998.

[31] Jon Skinner. `http://www.sublimetext.com`. Accessed: 17,February,2014.

[32] G Szego. Measure of risk. In *Journal of Operational Research*, volume 163, pages 5–19, 2005.

[33] Qingnian Wang, Kun Yan, Xiaofeng Wan, and Meiling Yuan. Application of particle swarm optimization in fussy neural networks. In *Information Assurance and Security, 2009. IAS '09. Fifth International Conference on*, volume 1, pages 158–161, Aug 2009.

[34] Wikipedia. `http://en.wikipedia.org/wiki/Portfolio_(finance)`. Accessed: 18,April,2014.

[35] Wikipedia. `http://en.wikipedia.org/wiki/Diversification_(finance)`. Accessed: 21,April,2014.

[36] Wikipedia. `http://en.wikipedia.org/wiki/List_comprehension`. Accessed: 30,April,2014.

[37] Wikipedia. `http://en.wikipedia.org/wiki/Portfolio_(finance)`. Accessed: 8,April,2014.

[38] X. Yao, Liu Y., and Lin G. Evolutionary programming made faster. In *Evolutionary Computation*, volume 3, pages 82–102, 1999.

Appendix A: User Manual

- To run the program on Unix/Linux

After starting the program you are asked to enter the name of the file containing the portfolio information. Make sure the file name does not contain any space and that the file extension is correct. If the file does have spaces, replace them with an underscore, example "file with spaces.txt" should be changed to "file_with_spaces.txt". This is just a precaution so the system does not get confused. Figure 1 shows what you should see.
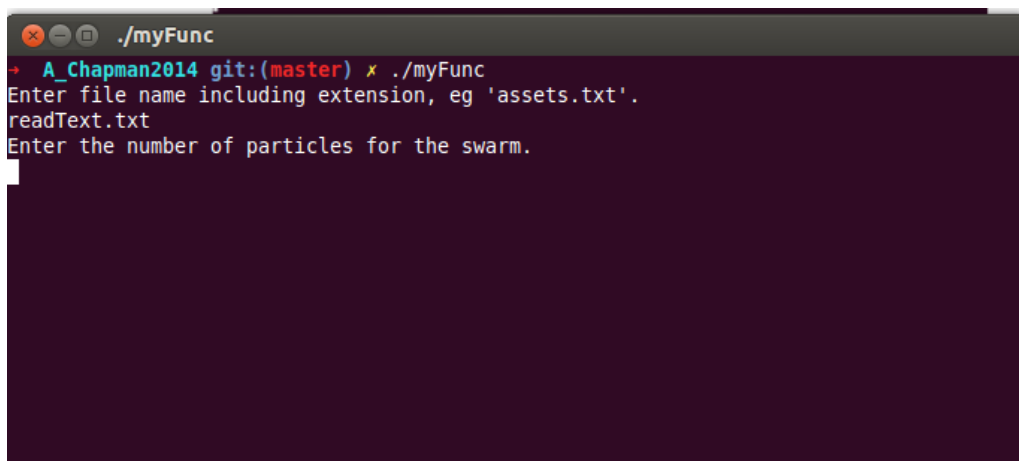
Figure 1: Enter file.

After entering the correct name of the file, you will be asked to type in the number of particles for the swarm in PSO, no more than 50 is needed. Figure 2 shows what you should see.

Figure 2: Enter the number of particles
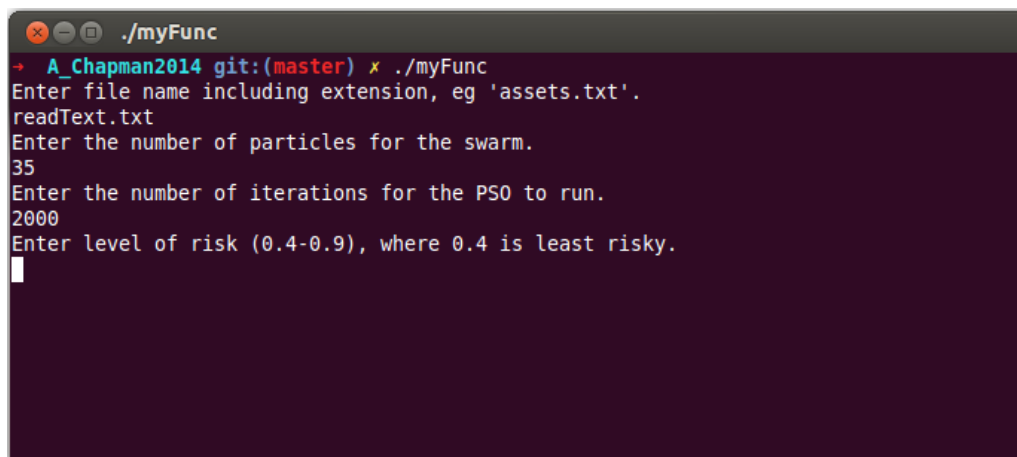
Then you will be asked to enter the number of iterations for the PSO to run for. Figure 3 shows what you should see.

Figure 3: Enter number of iterations
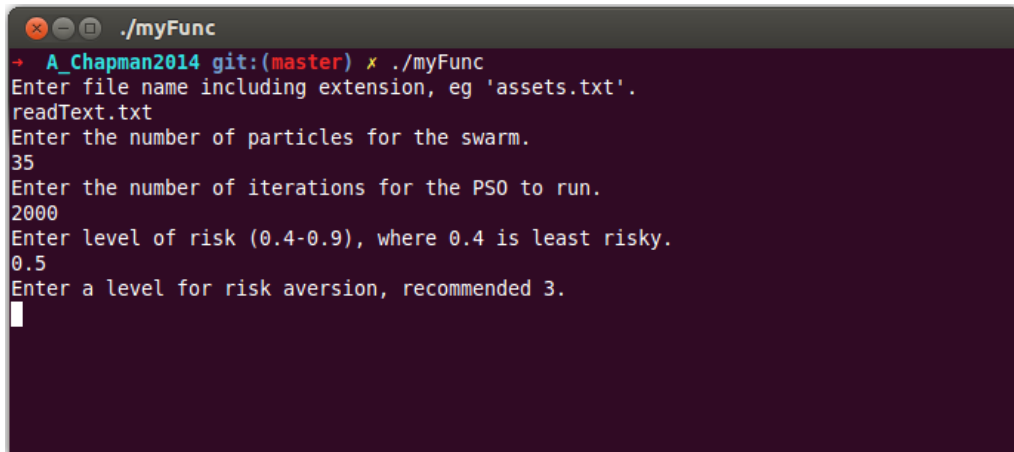
Followed the level of risk, 0.5 is neutral. Figure 4 shows what you should see.

Figure 4: Enter level of risk

Then input the lever of risk aversion, recommended is 3. Figure 5 shows what you should see.

Figure 5: Enter level of risk aversion

Finally, your expected required portfolio return. Figure 6 shows what you should see.

Figure 6: Enter expected required return

PSO is pretty quick so you will, if nothing was wrongly inputted, get an answer almost instantaneously. The system will display the "un-prettified" results on the window. Figure 7 shows what you should see.

Figure 7: Displaying results on command line

For the better looking results, open the folder containing the output file. The name will be "output-(date).txt" for example, output-(2014,5,1).txt. It will contain a file with the proportions of the optimal portfolio. Figure 8 shows what you should see, where it states that you should invest 11.37% on asset *Aa*, 5.35% on asset *Bb*, and so on.

Figure 8: Example output file.

Appendix B: Maintenance Manual

Instructions on how to compile and run the algorithm:

Makes sure that the source Haskell files (myFunc.hs, PSO.hs) are located in the same folder as the asset data files you wish to use for the portfolio.

1. Open a terminal and move to the folder containing the source files, ie the Haskell code.

2. Type the following command into the terminal and press enter:

   % ghc -O2 –make myFunc.hs -threaded -rtsopts

3. You have now created an executable file, now type into the terminal the following and press enter:

   % ./myFunc

4. The program will start in the same terminal window.

The User Manual provides a guide for using the program.

Organisation of System Files, including directory structures