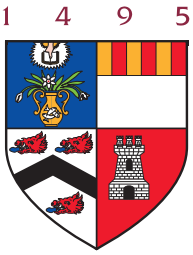

University of Aberdeen

Department of Computing Sciences

CS4526 Project Report

2013–2014

Particle Swarm Optimisation for the Portfolio Selection Problem in a Function Based Environment

Author : **Anthony S. Chapman**

Supervisor : **Dr. Wei Pang**

Declaration

I hereby declare that this report has been composed by me. I also declare that all sources of information have been specifically acknowledged and all quotations distinguished by quotation marks.

(signed) .....

Abstract

Abstract....

Acknowledgments

Acknowledgments....

Contents

CHAPTER 1

Introduction

1. Overview

2. Motivation

3. Primary Goals

4. Secondary Goals

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 fully

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 [**8, 18, 4, 14**]. It simulates the behavior 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.

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):

$$\begin{aligned} &V_i^t : \text{The velocity of particle i at time t.} \\ &X_i^t : \text{The current position of particle i at time t.} \\ &Pbest_i^t : \text{The personal best position of particle i at time t.} \\ &Gbest^t : \text{The global best position of particle i at time t.} \end{aligned} \tag{2.1}$$

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

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

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

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

ω : Inertia weight parameter.

c_1, c_2 : Acceleration coefficients.

r_1, r_2 : Random numbers between 0 and 1.

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

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

(2.2)

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].

```
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(r1, r2)
        updateVelocity(i, r1, r2)
        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. Portfolio Management

The money maker...

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 everything is a mathematical function, so everything has an input and an output, similarly to object orientated languages, where everything is an object. This is not exactly clear, to make it simpler, in Haskell, everything is a function or a variable, but something it can be both! By this I mean we can apply function composition. One can think of this by having a function and to that function you can input another function and the output could either be yet another function or a value.

"Real World Haskell" by Don Stewart, Bryan O'Sullivan, and John Goerzen is a brilliant book to quickly get your head around Haskell. 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. One I started learning about Haskell, there was no going back.

Back to our project, Haskell is one of the leading lazy evaluation languages in the functional programming community [**7**]. Lazy evaluation, if you did not know, is an evaluation 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.

CHAPTER 3

Related Work

1. Markowitz Model

One of the first contributions to the portfolio problem was made by Markowitz [**11**] which was later described in more detail in his book [**12**]. 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 $\sum_{i=1}^{N}\sum_{j=1}^{N} x_i x_j \sigma_{ij}$ 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:

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

$$\begin{aligned} \textit{Constraints:} \sum_{i=1}^{N} x_i r_i &= \pi \\ \sum_{i=1}^{N} x_i &= 1 \\ x_i \geq 1, i \in \{1, 2, \ldots, N\} & \end{aligned} \tag{3.1}$$

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 [**21**].

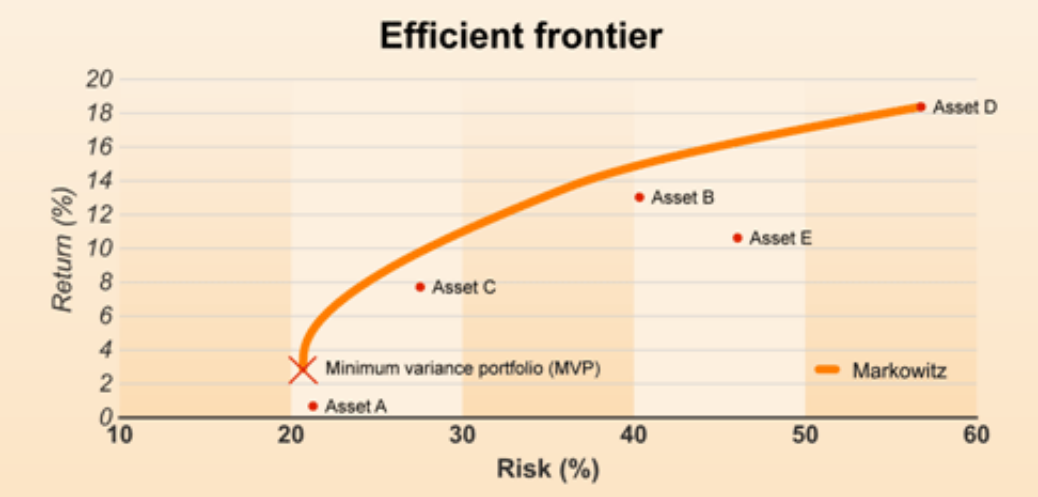

FIGURE 1. Example of an Efficient Frontier Without a Risk-Free Asset [**17**].

Each point on the line in Figure 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 1 Assets A-E represent what the portfolio will be made out of.

Markowitz's model is subject to serous criticisms as stated in [**20**], 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 [**20**].

This criteria should not be taken lightly since the assets' return does not follow normal distributions in real world situations [**10**]. 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 [**2, 6**].

2. Portfolio in Excel

3. Something PSO

4. PSO applied

CHAPTER 4

Problem Domain

1. Approach

CHAPTER 5

Requirements and Risk Assessment

This section describes the requirements for this project. Table 1 refers to the functional requirements from technical point of view. Section 2 focuses on the non-functional requirements of the system.

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 1. Functional requirements for system.

2. Non-functional

As this system is an extension on a PSO module [**16**], 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

Some possible risks which might affect the development of my project and some worries which must not be overlooked

1. Social

Stuff like joint-honours so have half the time

2. Project Based

2.1. Haskell. Nature of Haskell being difficult as the type settings so strong (transferring data, parsing) solved by partial functions.

2.2. PSO. Becoming stuck in local minima and not giving optimal solution

2.3. Portfolio. Company suddenly goes bust, application has no notion of background

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.

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 slightly longer than planned for whatever reason.

——Project timeline——

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.

2. Technology

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.

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 [**19**] and similarly for the choice of OS, I am happy with this editor.

CHAPTER 8

System Design and Architecture

1. Original PSO Implementation

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 [**13**] 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.

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 [**15**]. 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[**1**].

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.

1.4. Termination. An 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 it's position.

2. Expansion for Portfolio Optimisation

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 and they might not fully understand their purpose and therefore give un-optimised inputs.

2.2. Constriction Factor. Maurice Clerc in his study on stability and convergence of PSO [**3**] 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:

$$V_i^{t+1} = K\left[V_i^t + c_1 r_1 \times \left(Pbest_i^t - X_i^t\right) + c_2 r_2 \times \left(Gbest^t - X_i^t\right)\right] \tag{8.1}$$

where

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

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 [**9**].

2.3. 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 carring of. 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.

2.4. Fitness Function.

2.5. 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

Financial Data

1. Data Description

2. Problem Domain

3. Assets and their Weights

4. Analysis

5. PSO Parameters

6. Experimentation and Testing

7. Portfolio Constraints

8. Results

CHAPTER 10

Experimentation and Testing

1. Efficiency

Standard Deviation stuff, box plots blahh blahhh

2. Constriction Factors

In System Design and Architecture 2.2 the concept of a constriction factor was introduced.

2.1. 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 [**15**], 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 [**3**]. 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.

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

2.3. Results. This subsection shows the results and the following Table 1 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	1.61088×10^{-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 1. Results for Constriction Factors.

Table 2 shows what the acronyms in Table 1 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 2. 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 1 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 1 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.

2.4. 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"[**5**], 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.

3. Scalability

3.1. Testing Strategy. The application will be run

3.2. Hypothesis.

3.3. Results.

3.4. Conclusion. Number of assets

4. Relationships

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

4.1. Testing Strategy. I will run nine tests and record, the time it takes to finish and the results just to make sure their are consistent. Each of the nine tests will be run 20 times with a mean and standard deviation calculated and shown in the results. 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

4.2. Hypothesis.

4.3. Results. Sum of weights for all results Results mean 1. Results standard deviation $1.3735 * 10^{-}9, 0., 1.60689 * 10^{-}11, 6.4372 * 10^{-}9, 2.66467 * 10^{-}15, 1.29662 * 10^{-}11, 0.027735, 2.28702 * 10^{-}11, 2.54305 * 10^{-}10$

Time taken for PSO to finish for all tests Results mean 0.0463947, 0.107632, 0.0812466, 0.0555938, 0.1253, 0.106167, 0.089034, 0.181696, 0.17403 Results standard deviation 0.00487954, 0.00912289, 0.00542592, 0.00491575, 0.00699414, 0.00669427, 0.00856579, 0.00828043, 0.00557404

Expected Return Means: 0.0447, 0.0447, 0.0447, 0.0527, 0.0527, 0.0527, 0.101475, 0.1007, 0.1007 Standard Deviation: $6.13954*10^{-}11, 1.58041*10^{-}17, 7.18283*10^{-}13, 3.3924*10^{-}10, 1.41115 * 10^{-}16, 6.83323 * 10^{-}13, 0.00279292, 2.30303 * 10^{-}12, 2.56085 * 10^{-}11$

4.4. Conclusion.

5. Scalability Time

This test might seem a little peculiar at first but be assure, there is method in this madness (or madness in this method!). I want to see what the relationship is between time, number of particles, number of iterations and number of assets.

5.1. Testing Strategy. The application will be run

5.2. Hypothesis.

5.3. Results.

5.4. Conclusion. Number of assets

6. Penalty value

6.1. Testing Strategy. The application will be run

6.2. Hypothesis.

6.3. Results.

6.4. Conclusion. For fitness function

7. Asset percentage/Induced/Forced Diversification

7.1. Testing Strategy. The application will be run

7.2. Hypothesis.

7.3. Results.

7.4. Conclusion. Constraint that says you have to invest between 0.05 to 0.35 on each asset

8. Risk and Risk Aversion

8.1. Testing Strategy. The application will be run

8.2. Hypothesis.

8.3. Results.

8.4. Conclusion. Level of riskiness

CHAPTER 11

Future Work

1. PSO Parameters

More testing on WPG inertia weights Dunno...

2. Self-termination

Stops after some criteria is met

3. Diversification

One of the most interesting concepts in portfolio theory I found was that of diversification, 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.

4. Asset's Covariance

Risk in terms of

5. Market Relationships

Gold vs Money

This would be brilliant!!

6. Real-time processing

Wow!

CHAPTER 12

Discussion and Conclusion

1. Discussion

2. Future Work

3. Conclusion

Bibliography

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

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

[3] 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.

[4] 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.

[5] M. Clerc and J. Kennedy. The particle swarm: Explosion, stability, and convergencein a multidimensional complex space. 2002.

[6] 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.

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

[8] 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.

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

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

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

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

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

[14] 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.

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

[16] Pablo Rabanal, Ismael Rodguez, and Fernando Rubio. `http://antares.sip.ucm.es/prabanal/english/heuristics_library.html`. Accessed: February,2014.

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

[18] 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.

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

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

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