Evolutionary Algorithms and Computational Methods for Derivatives Pricing

Samuel Palmer

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

February 20, 2019

I, Samuel Palmer, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

"My CPU is a neural net processor; a learning computer"

- Arnold Schwarzenegger

Abstract

This work aims to provide novel computational solutions to the problem of derivative pricing. To achieve this, a novel hybrid evolutionary algorithm (EA) based on particle swarm optimisation (PSO) and differential evolution (DE) is introduced and applied, along with various other state-of-the-art variants of PSO and DE, to the problem of calibrating the Heston stochastic volatility model. It is found that state-of-the-art DEs provide excellent calibration performance, and that previous use of rudimentary DEs in the literature undervalued the use of these methods.

The use of neural networks with EAs for approximating the solution to derivatives pricing models is next investigated. A set of neural networks are trained from Monte Carlo (MC) simulation data to approximate the closed form solution for European, Asian and American style options. The results are comparable to MC pricing, but with offline evaluation of the price using the neural networks being orders of magnitudes faster and computationally more efficient.

Finally, the use of custom hardware for numerical pricing of derivatives is introduced. The solver presented here provides an energy efficient data-flow implementation for pricing derivatives, which has the potential to be incorporated into larger high-speed/low energy trading systems.

Impact Statement

The work presented in this thesis contributes to the applications and methodologies of evolutionary algorithms (EAs), neural networks, and other computational methods for problems of academic and commercial interest within financial engineering. As well as exploring novel ideas and improving current methods, this work highlights potential directions for further research.

The explicit focus on specific applications in this thesis highlights the value of this research outside of academia. The work presented shows how EAs, along with alternative heterogeneous computing platforms, can be successfully used to enhance the efficiency of frequent computational tasks faced by financial institutions. Apart from potentially reducing operating costs of financial institutions, the use of more efficient computational methods allows for less resources usage and lower power consumption, which can further aid in reducing the industry's ecological impact via its carbon footprint.

In conjunction with the work presented in this thesis, conference publications and presentations have also been leveraged to extend the reach of this research to both academic and industry experts. Work in this thesis has been presented at the following conferences: Computing in Economics and Finance (2015 & 2016); Engineering Applications of Neural Networks (2016;) and the European Symposium on Artificial Neural Networks (2017). Furthermore, this work has received active interest from current financial practitioners regarding further development of the methodologies presented for use in practical applications.

Acknowledgements

I am grateful for having the opportunity to study for my PhD, and this thesis would not have been possible without the unconditional support of my family and friends, and the invaluable contributions of Mitzy and Monarch. It would often seem that completing this thesis was always just out of reach, and even driving me to run off to the Amazon rainforest to catch butterflies. It provided an easy punchline for many of us, but despite the joking you have all been by my side, making this journey possible.

During my studies at UCL I have met many new people and made new friends from all over the world, all of whom have enriched my PhD experience. In particular I would like to thank those at Bloomsbury Fitness for their camaraderie that kept me sane throughout the years, and motivated me to better myself physically as well as academically.

I would like to thank my supervisor for over the years coping with my erratic trails of thought, and the many hours spent deciphering my cryptic grammar and spelling.

Finally, I am eternally thankful to my parents, brother and partner for believing in me every time I tell them “it’s almost there”, and for their love and support whilst writing the most expensive book they will ever buy.

Like father, like son.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Financial derivatives are contracts written allowing the holder of the contract to buy or sell an underlying asset for a specified price at a given time in the future. A popular type of derivative contract is an *option*. An option gives the holder the right, but not the obligation, to buy (a *call* option) or to sell (a *put* option) the underlying asset. Option contracts are not a modern day concept and have been around since the 16th century, and played an important part in speculative commodity trading by the Dutch East Indies Company and Dutch West Indies Company trading on the Amsterdam bourse during the 17th century [1].

In modern times option contracts play a major role in the finance industry, as well as other commodity heavy industries such as airlines and mining. Options are used to hedge exposure to currency exchange and interest rates, and in industry to protect against oil and commodity price fluctuations, as well as being speculative assets profiting from potential market mispricings and exploiting arbitrage opportunities. As of June 2018 the global average daily turnover of exchange traded options for the two largest markets, interest-rates and foreign exchange options, were $1,704 billion USD, and $16 billion USD respectively [2].

Given the vast scale of the option markets it is important to be able to properly understand and value such contracts; even in the days of trading on the Amsterdam bourse the basics of options pricing theory such as put-call parity were understood [3]. However, since then modern financial theory has come a long way, introducing more elaborate mathematical models of asset price behaviour as well as

the use of mathematical and computational tools to try and accurately value these contracts. Most notable is the derivation of the famous Black-Scholes solution used to price European style options, which led in a new era of financial mathematics and modelling. Increasingly complex models have been introduced to try and capture the observed price behaviour of observed exchange option prices, introducing multiple stochastic factors such as stochastic volatility [4] and interest rate [5], and jump diffusion dynamics [6]. In addition there is an increasing array of exotic option contracts, which often means that in many cases analytical forms of the price do not exists and pricing then relies heavily on numerical and computational methods.

Since the financial crash of 2007-2008 the volume of options traded has slightly fallen; this is due to new tighter regulations on derivatives trading being imposed. As part of these regulations financial institutions now have to calculate their daily exposure to manage cash reserves, and this is now a critical part of operations. This leads to an increasing pressure on efficient and accurate computation of option pricing models to measure the exposure of portfolios. To meet this demand many institutions are turning to the use of high-performance-computing architectures, in particularly GPUs, to make use of parallel computing. Monte Carlo pricing methods are well suited for GPU implementation and allow for an accessible approach to exotic option contracts and elaborate pricing models. However relying on high-performance-computing and its development to meet the ever increasing computational demands is not sustainable, and the performance is subject to Moore's Law; this approach also requires more hardware, more power, more space and overall higher costs. Instead an alternative approach would be to explore more efficient computational methods that could be used, the major topic of this thesis.

With a globally increasing interest in machine learning, this is now becoming an area of increasing interest for financial researchers. This thesis explores how machine learning methods, more specifically evolutionary algorithms and neural networks, can be used in options pricing. One important area explored in this thesis, which is of growing interest, is the use of neural networks to approximate

the solution of option pricing models; compared to popular Monte Carlo GPU implementations which take in the order of 10ms to price a single option, the neural network approach explored here can take a fraction of this time and also be run on regular desktop CPUs, with the potential of further speed-ups by then applying HPC architectures. In addition this thesis also looks at the application of evolutionary optimisation algorithms for the problem of model calibration, and the implementation of an HPC architecture for accelerating numerical methods.

1.1 Research Objectives

1) To develop a powerful problem-general evolutionary optimisation algorithm.

Although current EA algorithms show good results over many problems, the issue of local minima and trapping can slow optimisation progress down; there are also concerns that although current algorithms may show good behaviour on contrived mathematical benchmark functions these results may not be relevant to the landscape of applied problems. As an avenue for improving the performance of EAs, elements from popular and powerful evolutionary algorithms are here combined to produce a single hybrid algorithm. Self parameterisation mechanisms are also introduced to eliminate the need for parameter tuning to obtain optimal algorithmic performance and to increase the algorithm' reliability over all the optimisation problems considered.

2) To compare to use of the above and other candidate EAs for the problem of calibrating the Heston stochastic volatility model.

Current research literature using EAs for Heston model calibration has focused only on rudimentary versions of the algorithms. This work aims to compare the use of a number of more advanced EAs for the problem of Heston model calibration, as well as the use of the newly develop BrPSO algorithm in a practical application.

3) To use a combination of neural networks and EAs to deliver, after training, computationally efficient approximate solutions for exotic derivates pricing.

One issue with current numerical methods is that they only provide a one-off so-

lution for the specific model parameters, and hence have to be reran for every new parameterisation. This work aims to develop a methodology using neural networks trained by EAs to approximate the solution for option pricing models. The neural network solution offers a closed-form approximation of the ideal solution, and once the neural network has been trained it can be used for all model parameterisations and thus offer highly efficient price evaluations.

4) Investigate the use of custom hardware for acceleration of financial calculations.

As an alternative approach, efficient problem specific custom hardware can be used to accelerate existing numerical methods. Compared to GPU devices field programmable gate arrays (FPGAs) offer a platform for custom hardware design. By implementing designs on an FPGA low level parallelisation of an algorithm can be achieved; this can offer computational speed-ups as well as being more energy efficient. As an example a custom FPGA design for the solving tridiagonal systems of equations is designed and implemented, which can be used as part of an implicit finite difference solver for options pricing. The use of custom hardware has the potential that neural network models could also be implemented on FPGA devices.

1.2 Contributions to Research

Breeding Particle Swarm Optimisation - Chapter 3

1. The introduction of a novel hybrid particle swarm optimisation algorithm using discrete crossover and dynamic self-adaptive parameterisations.

2. An analytical and empirical analysis of discrete crossover operators for global convex optimisation.

Calibrating the Heston Model using Evolutionary Algorithms - Chapter 4

1. Improve upon current methods used in the financial literature for heuristic calibration methods using BrPSO and other state-of-the-art differential evolution and hybrid algorithms. This work also highlights important concepts with respect to the robustness of the heuristic calibration methodology and the numerical integration scheme used.

Options Pricing using Neural Networks and Evolutionary Optimisation - Chapter 5

1. Demonstration that the function approximation of the Black-Scholes solution can be reduced to requiring only 3 model inputs to cover the 5-dimensional parameter space.

2. Use of linear ensembles of neural networks to provide pseudo-analytical solutions to efficiently and accurately price European and exotic options.

Options Pricing using Hardware Acceleration - Chapter 6

1. A design is presented for a parallelised implementation of the Thomas algorithm for solving tri-diagonal systems of equations using field-programmable-gate-arrays.

2. Analytical bounds are provided for the range of values required for fixed-point arithmetic implementation of LU-decomposition.

3. Application of the presented hardware design and analysis for pricing European options using implicit finite difference schemes.

1.3 Thesis Structure

Chapter 1 introduces the problems addressed by the thesis. **Chapter 2** then gives necessary background to the thesis: evolutionary optimisation algorithms, introducing both Particle Swarm Optimisation (PSO) and Differential Evolution (DE) algorithms; neural networks and neural network training; financial mathematics for derivatives pricing. In **Chapter 3** Breeding Particle Swarm optimisation (BrPSO) is introduced, a novel PSO-Crossover hybrid algorithm for optimisation, and as a proof of concept BrPSO is successfully applied to a problem in naval engineering. **Chapter 4** reviews the currently limited applications of EAs to model calibration in finance before demonstrating that BrPSO and other state-of-the-art EAs can be used successfully for calibration of the Heston stochastic volatility model. In **Chapter 5** neural networks trained using BrPSO and other EAs are used for derivatives

pricing, providing accurate generalised price approximations for European, Asian, and American options. **Chapter 6** addresses a different approach to more efficient financial computation, showing how custom hardware can be utilised, with the potential to be combined in future with the EA-based approach of earlier chapters. **Chapter 7** concludes with a discussion of the work presented in the thesis together with some suggestions for future work.

1.4 Publications and Conference Presentations

The following publications and presentations were completed over the course of this thesis.

1. S. Palmer and D. Gorse. Pseudo-Analytical Solutions for Stochastic Options Pricing Using Monte Carlo Simulation and Breeding PSO-Trained Neural Networks. In *ESANN 2017 Proceedings, European Symposium on Artificial Neural Networks*, 2017.

2. S. Palmer, D. Gorse, and E. Muk-Pavic. Neural Networks and Particle Swarm Optimization for Function Approximation in Tri-SWACH Hull Design. In *Proceedings of the 16th International Conference on Engineering Applications of Neural Networks (INNS) - EANN '15*, pages 1–6, New York, New York, USA, 2015. ACM Press.

3. S. Palmer. Options Pricing using Monte Carlo Simulations and Neural Networks. *The Society for Computational Economics 22nd International Conference Computing in Economics and Finance, 2016* .

4. S. Palmer. Accelerating Implicit Finite Difference Schemes Using a Hardware Optimized Tridiagonal Solver for FPGAs. *The Society for Computational Economics 21st International Conference on Computing in Economics and Finance, 2015*.

Chapter 2

Background

2.1 Evolutionary Optimisation Algorithms

Evolutionary optimisation algorithms as a method of solving real minimisation problems

$$\mathbf{x}_g = \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}}\, f(\mathbf{x}) \tag{2.1}$$

where the algorithm finds the vector, $\mathbf{x}_g$, that minimises the objective function or otherwise known as the fitness function, $f()$. Evolutionary algorithms (EAs) are a class of heuristic search algorithms based upon natural phenomena. The three most popular types of EA are: genetic algorithms (GAs); differential evolution (DEs); and particle swarm optimisation (PSO) [1]. In this work the focus is primarily on the use and development of PSO, but with a sub-context on the use of DE.

2.1.1 Particle Swarm Optimisation

Particle swarm optimisation (PSO) is a heuristic search algorithm based upon the flocking/swarming behaviour observed in various species in nature. The routes of PSO extend back to using particle systems in graphics modelling of fuzzy objects. A noticeable example was Reynolds [7] boid flocking model that used a particle system to simulate the flocking behaviour of birds. PSO was a development of these particle flocking systems, devised by Eberhart and Kennedy [8]. Eberhart and

[1] although PSO is not technically 'evolutionary' as such, it is placed under this general umbrella of methods which moreover covers naturally inspired algorithms.

Kennedy used the term swarm instead of flock based on systems behaviour being coherent with the principles of swarm intelligence given by Millonas [9]. Kennedy describes the concept of PSO as "[i]ndividuals changing their beliefs to become more like their neighbours. Thus it is a social-psychological model of knowledge management" [10]. In essence, PSO uses a population of particles, each particle representing a candidate solution within the search space, each which then follow specified dynamics to move around the search space, with all particles finally converging towards the optimal solution.

An important difference between PSO and other naturally inspired/evolutionary algorithms, such as genetic algorithms (GAs) or differential evolution (DE), is that each particle has a memory component. Memory is an important part of PSO and allows the particles to move back towards previously known good positions, in comparison to GAs or DE, for instance, where members of the population cannot move back towards to a previously preferred solution; although some more recent DE developments such as SHADE [11] do incorporate a type of memory component.

PSO works by intelligently searching through the n-dimensional search space to minimise the objective function $f(\mathbf{x})$. PSO uses a population, a set of P number of particles collectively known as a *swarm*. Each particle, $\mathbf{p}_i$, is made up of three components: The position of each particle, $\mathbf{x}_i \in \mathbb{R}^n$, represents a candidate solution for the objective function within the search space; the velocity, $\mathbf{v}_i \in \mathbb{R}^n$, which is an n-dimensional vector that describes how the particles' position moves in the search space for each iteration; and the personal best location, $\mathbf{y}_i \in \mathbb{R}^n$, which is the particles' best known historical position that it has discovered within the search space. The in additional to the particles the swarm has the global-best component, $\mathbf{y}_{\text{gBest}} \in \mathbb{R}^n$, which keeps track of the overall best found position of all the particles; furthermore each particle, depending on the variant of PSO used, can have a local-best component $\hat{\mathbf{y}}_i \in \mathbb{R}^n$ that keeps track of the best found position of a selected subset of the swarm related to the particle $\mathbf{p}_i$. The use of $\mathbf{y}_{\text{gBest}}$ and $\hat{\mathbf{y}}_i$ will be described in more detail further on with respect to the velocity update equations. Algorithm 2.1 gives an outline of the basic PSO algorithm.

The four main stages of PSO are:

1. **Initialisation** : Firstly the particles of the swarm are initialised. This involves setting up their initial starting positions within the search space and defining an initial velocity. The simplest initialisation is to use a uniform distribution, used in the definition of the 2007 Standard PSO [12], though Clerc [13] suggests that although simple it is a poor choice to use. This is because on any one sample of uniform initialisation the search space won't be evenly covered, on average over many initialisations it would, but in the case of each individual initialisation it may not provide a reasonable representation. As such Clerc [13] suggests using a Hammersley distribution or Tessellations method to provide a more even coverage of the search space.

2. **Fitness evaluation** : At the beginning of each iteration the position of each particle is evaluated for the objective function; the value of the function at this position is the particles fitness value. The fitness evaluation is usually the most computationally expensive part of the PSO algorithm, and as such it is desired to have an algorithm that converges quickly to reduce the number of fitness evaluations required.

3. **Velocity update** : The velocity update is the crux of PSO and characterises the dynamics of the particle's movement through out the search space. The numerous improvements and developments of the velocity update equation are discussed in more detail later on.

4. **Position update** : The particle is moved in the search space by using the new updated velocity. Typically this is simply done by addition of the velocity

$$\mathbf{x}_i(t) = \mathbf{x}_i(t-1) + \mathbf{v}_i(t). \tag{2.2}$$

The most defining feature of PSO algorithms are the velocity update equations. The velocity vector of the particle is itself dynamic and the velocity update equation dictates how the velocity, $\mathbf{v}_i(t)$, is updated for every iteration (time step),

Algorithm 2.1 Pseudo code for an outline of the particle swarm optimisation algorithm.

For each particle, $\mathbf{p}_i$, initialise the position, $\mathbf{x}_i$, velocity, $\mathbf{v}_i$, and $\mathbf{y}_i = \mathbf{x}_i$
while Stopping criteria not met **do**
  for Each particle $\mathbf{p}_i$; $i \in \{1 \dots P\}$ **do**
    Calculate the current fitness, $\text{fit}_i = f(\mathbf{x}_i)$
    Update personal best, $\mathbf{y}_i = \mathbf{x}_i$ if $\text{fit}(\mathbf{x}_i) \leq \text{fit}(\mathbf{y}_i)$
    Update swarm global and/or local best, $\mathbf{y}_{\text{gBest}} = \mathbf{y}_i$ if $\text{fit}(\mathbf{y}_i) \leq \text{fit}(\mathbf{y}_{\text{gBest}})$
  end for
  for Each particle $\mathbf{p}_i$; $i \in \{1 \dots P\}$ **do**
    Update velocity, $\mathbf{v}_i(t)$, according to velocity update equation
    Update position, $\mathbf{x}_i(t) = \mathbf{x}_i(t-1) + \mathbf{v}_i(t)$;
  end for
end while
Output global best, $\mathbf{y}_{\text{gBest}} = \text{argmin } f(\mathbf{y}_i)$; $i = \{1 \dots P\}$

t. The velocity update controls the search ability of PSO by determining the degree of exploration and exploitation of the particles. The new velocity at time t is a linear combination of the previous velocity, $\mathbf{v}(t-1)$, which constitutes as the exploration inducing component, and two components encouraging movement towards currently known good solutions, exploitation. The original velocity update equation, referred to as original-PSO, given by Eberhart and Kennedy is

$$\mathbf{v}_i(t) = \mathbf{v}_i(t-1) + C_1\mathbf{r_1} \odot (\mathbf{x}_i - \mathbf{y}_i) + C_2\mathbf{r_2} \odot (\mathbf{x}_i - \hat{\mathbf{y}}_i) \tag{2.3}$$

where C_1 and C_2 are constants, $\mathbf{r_1}$ and $\mathbf{r_2}$ are vectors of uniformly distributed random numbers from the interval $[0,1]$ and $\hat{\mathbf{y}}_\mathbf{i}$ is the ith particles' neighbourhoods' best position. The neighbourhood, $\mathfrak{N}_i$, of particle i is defined as the network topology for which other particles within the swarm can share information with particle i. The concept of topology will be discussed in more detail later on. For now, the simplest topology is where the local best for all particles is the swarm global best, $\hat{\mathbf{y}}_\mathbf{i} = \mathbf{y}_{\text{gBest}}$, PSO-gBest. In this topology all the particles can share information with each other, each particle is fully connected, and $\hat{\mathbf{y}}_\mathbf{i}$ is simply the best known position

of all particles in the swarm

$$\mathbf{y}_{\text{gBest}} = \text{argmin } f(\mathbf{y}_i) \; ; \mathbf{y} \in \{\mathbf{y}_1 \dots \mathbf{y}_P\} \tag{2.4}$$

where then $\hat{\mathbf{y}}_i = \mathbf{y}_{\text{gBest}}$ for all i. The velocity update equation consists of three parts: memory; social; and cognitive. The social and cognitive parts move the particle in a weighted averaged direction between the particles personal best and the swarms global best. The social part causes the swarm to gradually contract towards the global best solution, $\mathbf{y}_{\text{gBest}}$. The cognitive part causes the particle to move towards its best-known position, $\mathbf{y}_i$. The cognitive and social parts are weighted each by the constants C_1 and C_2 respectively, and uniformly distributed random numbers, $\mathbf{r_1}$ and $\mathbf{r_2}$. The random numbers are an important component of the velocity update rule, Wilkes *et al* [14] show that without the randomness PSO results in performing a line search, whilst having the random components creates diversity in the population.

The main issue faced in the original PSO algorithm is stability/convergence due to the fact that it is possible for the velocity to tend towards infinity and particles to rapidly escape the search space. To try and control this the simplest method is to use velocity clamping, which imposes fixed bounds to the velocity values, $[-\mathbf{v}_{\max}, \mathbf{v}_{\max}]$, though this simple approach can still negatively effect swarm convergence [15]. Velocity clamping allows control of the maximum granularity of the search and is problem specific.Velocity clamping effects the oscillatory movements of the particles [16], too high and particles may still be too explorative, and too small results in a lack of exploration that limits the search.

Further improvements to PSO were soon made by adding an inertia weight w to the memory component [15], the modified velocity update equation is now

$$\mathbf{v}_i(t) = w\mathbf{v}_i(t-1) + C_1\mathbf{r_1} \odot (\mathbf{x}_i - \mathbf{y}_i) + C_2\mathbf{r_2} \odot (\mathbf{x}_i - \hat{\mathbf{y}}_i). \tag{2.5}$$

An inertia weight is used as a means of balancing the swarms' global and local search ability and allows the rate of swarm contraction to be controlled. A large inertia weight ($>$1.2) encourages a global search by carrying the particle further

in its current direction, whilst a small inertia (<0.8) encourages a local search. Although the second assumption is debatable, Chaun *et al* [17] show that a small w only encourages a local search under certain parameter conditions. In general a small inertia is nearly always preferred as this encourages convergence and stability of the swarm. Rather than remaining a constant value, the inertia is commonly a linearly decreasing functions of time, often from $0.9 \rightarrow 0.4$. Other approaches have used chaotic inertia weights, where the inertia decreases whilst oscillating [18].

Alternatively a constricted velocity update equation can be used [19], this introduces a constriction factor χ with the aim of increasing local-convergence. The constricted velocity update equation is given by

$$\begin{aligned} \mathbf{v}_i(t) &= \chi(\mathbf{v}_i(t-1) + C_1\mathbf{r_1} \odot (\mathbf{x}_i - \mathbf{y}_i) + C_2\mathbf{r_2} \odot (\mathbf{x}_i - \hat{\mathbf{y}}_i)), \\ \chi &= \frac{2}{2 - C - \sqrt{C^2 - 4C}} \end{aligned} \tag{2.6}$$

where $C = C_1 + C_2, C > 4$ is chosen for guaranteed convergence. Eberhart and Shi [19] show that the constricted and unconstricted inertia velocity update rules are algebraically equivalent. Thus swarm stability can be controlled by careful parameter selection, most commonly $\chi = 0.7298$ with $C_1 = C_2 = 2.05$, as it has been analytically shown that these settings lead to swarm stability [20]. Furthermore, Eberhart and Shi [19] suggest that constriction alone is not always the best method and can be improved by combining with velocity clamping, where $\mathbf{v}_{\max} = \mathbf{X}_{\max}$, where $\mathbf{X}_{\max}$ is the maximum range of the search space.

The values C_1 and C_2 affect swarm diversity and as such some approaches look at controlling these values rather than keeping them constant [21] [22]. In adaptive PSO (APSO) [21], C_1 and C_2 are increased or decreased depending on the state of the swarm to encourage increasing diversity for exploration and decreasing diversity for convergence, this approach is seen to have positive effect of search performance for multimodal problems with a good ability to avoid local minima. Although Ma *et al* [23] suggest that with group-decision-making-PSO (PSOGDM) keeping a small population diversity is better; this maybe so for unimodal problems but for multimodal problems, particularly with respect to the Schwefel function

their argument is not particularly compelling.

In essence PSO can be thought of as a intelligent way of sampling the search space, as such an alternative line of thought, and a more abstract view of PSO, is to treat it as the sampling of an evolving probability distribution. This trail of thought is expressed in Bare-Bones PSO (BBPSO) [24] [25], where instead of a velocity update the position of each particle is drawn from a probability distribution characteristic to the search properties of PSO.

The other important component of PSO which controls swarm diversity and convergence is the topology.

2.1.1.1 Topology

It has been aforementioned that the topology of the swarm defines the connectivity and how information is shared throughout the particles in the swarm. In PSO the topology determines how the neighbourhood best, $\hat{\mathbf{y}}_\mathbf{i}$, in the velocity update equation, Equation 2.3, is determined. In comparison to PSO-gBest for local-best PSO (PSO-lBest), each particle has a neighbourhood, $\mathfrak{N}_i \subset S$, limited to only a subset of the total swarm (it was seen that PSO-gBest is a special subset of PSO-lBest where $\mathfrak{N}_i = S$).The neighbourhoods for each particle are overlapping to allow for information to eventually propagate throughout the entire swarm. Neighbourhood topologies dictate the flow of information and have a significant effect on the swarms performance [26] [27] [28]. In general, greater connectivity speeds up convergence but increases susceptibility to being trapped in local minima [26], although cases for some test functions have shown that this is not a strict assumption and inverse is possible [29]. Approaches such as unified PSO [30] (UPSO) use a combination velocity component using different topologies.

The standard topology used in PSO-lBest is the ring topology [12], this connects particles to a neighbour on either side, and as the name suggests, forms a ring of particles.

$$\mathfrak{N}_i = \{\mathbf{y}_{i-1}, \mathbf{y}_i, \mathbf{y}_{i+1}\} \tag{2.7}$$

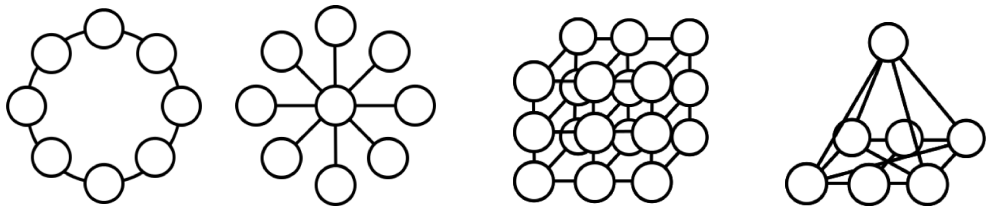


Figure 2.1: Commonly used particle swarm topologies.

The ring topology has been shown to perform well but exhibits the slowest convergence due to the slow propagation of information around the ring [26] [28]. Figure 2.1 depicts some of the other commonly used topologies. Out of these topologies it was found that the Von-Neumann(torus) topology was clearly superior, but that the pyramid topology also performs well. The strength of the Von-Neumann topology is due to the particles higher connectivity, and multiple paths for the distribution of information [26], although its disadvantage compared to the ring topology is the additional complexity in defining the neighbourhoods. For highly multimodal landscapes a wheel topology has also been shown to perform well [28], with the central particle acting as a filter for local minima and bad solutions. The worst performing topology with respect to finding the optima was the fully connected gBest. As a further improvement Mendes *et al* [31] suggests the fully informed particle methods (FIPs) where the particles take a weighted contribution from the neighbourhood; here they find the wFIPs variation with a ring topology is successful over all experimental runs, whilst using a fully connected, global-best, performing the worst. Another approach is to use random topologies, the advantage of these approaches is that these algorithms alleviate the need for parameter selection with respect to which topology to use.

Although there has been, and still is, and large ongoing debate about the preference of PSO-gBest or PSO-lBest. As seen, there is a lot of support for the preference of PSO-lBest for multimodal problems and PSO-gBest for unimodal problems. This is further illustrated later in this work in Section 4 were it is seen that PSO-gBest performs better than PSO-lBest for the given problem.

As a breakaway from idealisation of using a connective topology some other

PSO variations have taken the approach of exemplar learning [32] [33] [34]. The most noticeable example of this approach is Comprehensive-Learning PSO [32] (CLPSO), . In exemplar learning there is no fixed topology and instead $\hat{\mathbf{y}}_\mathbf{i}$ is defined as a combination of elements from selected 'example' particles. In CLPSO for the ith particle each element (dimension) in $\hat{\mathbf{y}}_\mathbf{i}$ is chosen probabilistically to be either from the particles' own personal best, $\mathbf{y}_\mathbf{i}$, or selected via some rule to be taken from a different particles', j, personal best, $\mathbf{y}_\mathbf{j}$. The elements of $\hat{\mathbf{y}}_\mathbf{i}$ are then refreshed periodically. As such there is no strictly defined topology within CLPSO. CLPSO also has the interesting property that it only has a single difference component in the velocity update, CLPSO will discussed in more detail further on in Section 2.1.1.3.

2.1.1.2 Standard PSO

In light of the historical developments of PSO since the original formulation in 1993 it is desirable to define a more relevant 'gold standard' of PSO algorithm [35]. This serves its purpose as suitable reference for development and comparison for other PSO algorithms. The first 'modern' standard was given in 2007, SPSO-2007, this algorithm follows the same structure as the original PSO algorithm given in Algorithm 2.1 and uses the constricted velocity update equation given in Equation 2.6. SPSO-2007 also suggests the use of PSO-lBest with a ring topology, and swarm size of 50 particles (although the authors do state that there was no difference observed with the range of 20-100 particles).

At the time of writing the most recent attempt of defining a standard is SPSO-2011 [36]. Though this standard lacks the grace and simplicity of the previously defined standards, which is arguably one of PSOs' most attractive traits for its implementation, and may explain its less widespread popularity within the literature as an actual standard [2]. SPSO-2011 looks to improve upon certain capabilities that were lacking in SPSO-2007, the main being rotational invariance. SPSO-2011 uses a hypersphere to generate the velocity update equation. It has been shown that SPSO-2011 is not affected by rotations of the search space [37] [38].

[2] this is assumption is based upon average citations per year at the time of writing since publication of the original articles, 93.8 for SPSO-2007 [12], and 46 from two papers for SPSO-2011 [35] [36].

These are attempts to create a standard PSO algorithm and are not designed with the intention of being the optimal PSO algorithm, but there are many interesting and exotic PSO variations that attempt to elaborate and improve upon the original PSO paradigm.

2.1.1.3 Further PSO Variations

There are numerous PSO variations discussed in reviews such as [39] [40] [41], a brief introduction to some of the more prominent and conceptually interesting PSO variants discussed in the literature and throughout this work is presented:

- **CLPSO** : Comprehensive-learning PSO (CLSPO) [32], is the most notable example of exemplar learning in PSO and is an extremely popular PSO variant with numerous applications within the literature. Its appeal is that it shows a significant improvement over previous PSO variants whilst still retaining a reasonably simple algorithmic structure. CLPSO has only two components in the velocity update

$$\mathbf{v}_i(t) = w\mathbf{v}_i(t-1) + C_1\mathbf{r_2} \odot (\mathbf{x}_i - \hat{\mathbf{y}}_i) \tag{2.8}$$

  where in this case $\hat{\mathbf{y}}_i$ is the exemplar vector rather than a topologically determined best known position.

  A variation of CLPSO is parallel-CLPSO [42] which uses CLPSO with distributed computing and subswarms and report significant improvements on the algorithms performance for multi-modal problems.

- **OLPSO** : Orthogonal Learning PSO (OLPSO) [33], uses the same exemplar based learning structure as CLPSO, Equation 2.8, but looks to improve upon the quality of exemplars used in learning. The issue addressed by introducing orthogonal learning is the concept of 'two steps forward, one step backwards'. Depending on the sensitivity of the objective function to a particular dimension the search may progress in a new direction that is beneficial for one dimension but detrimental in another whilst still minimising the objec-

tive function. This type of search progression can be a source of inefficiency.

- **ELPSO** : Example-based Learning (ELPSO) [34], is an extension of CLPSO and aims to provide and improved balance between swarm diversity and convergence speed. The premise of ELPSO is to use multiple elite particles to learn from and as such uses multiple 'global bests', $\hat{\mathbf{y}}_{\text{gBests}}$, as well an exemplar vector component

$$\mathbf{v}_i(t) = w\mathbf{v}_i(t-1) + C_1\mathbf{r_2} \odot (\mathbf{x}_i - \hat{\mathbf{y}}_i) + C_2\mathbf{r_2} \odot (\mathbf{x}_i - \hat{\mathbf{y}}_{\text{gBests}}). \tag{2.9}$$

  The exemplar vector, $\hat{\mathbf{y}}_i$, is generated in a similar fashion is CLPSO, where each dimension is taken from a personal best of a different particle; except the selection process is a lot simpler with the choice being made by a uniform random number $[1,N]$. The new vector, $\hat{\mathbf{y}}_{\text{gBests}}$, is created using the same selection procedure as previously described but with the selection being taken from a set, G of historical global bests from each iteration. Over the given test set ELPSO does show some statistically significant improvements over CLPSO, although the many of the cases the improvement is not of a notable magnitude.

- **CPSO** : Co-operative PSO (CPSO) [43], attempts to address the problem of 'two steps forward one step backwards' by suggesting that the objective function needs to be evaluated more frequently as values of each dimension change. It is an interesting approach as it decomposes the n-dimensional search space using $K = n$ number of sub-swarms; the type of PSO used to govern the sub-swarms can be any of the other PSO algorithms discussed, although the authors suggest using GCPSO. Each sub-swarm, k, searches in the nth dimension, d, where a context vector, $\mathbf{b}$, is used to provide the other $d-1$ dimensions to create the test vector for the objective function evaluations. The context vector is composed of the dimension d from each of the corresponding sub-swarms global best, $\hat{\mathbf{y}}^k_{\text{gbest}}$. i.e. $b_d = \hat{y}^k_{\text{gbest},d}$, where $k = d$. This original version of the algorithm is known as CPSO-S [44], CPSO-S_K is

then later introduced [43] which relaxes the search space decomposition so that $K \leq D$. This allows grouping of dimensions which maybe interrelated into the same subswarm. Though CPSO-S suffered from stagnation and trapping local minima. To overcome this CPSO-H_K is introduced [43]; CPSO-S_K is run in alternate iterations with another PSO algorithm, the two algorithms can then exchange information.

- **FIPS** : Fully informed PSO (FIPS) [31], changes how the the neighbourhood best vector is calculated, instead of selecting the single best position known in the particles neighbourhood the idea of FIPs is to use all the information of the neighbourhood, but with weighted contributions. Given that the particle now uses all available information to it the authors say that the particle is 'fully informed', but only within its local neighbourhood, and truly fully informed particles exist only when a fully connected topology is used. The authors introduce 5 different FIPs variations: FIPS; *w*FIPS; *wd*FIPS; Self; *w*Self. The two best performing were FIPS and *w*FIPS; FIPS uses a equally weighted sum of contributions from all neighbours, whilst *w*FIPS uses a weighting based on the particles fitness; with *w*FIPS being the best overall and successfully finding the optima in all the of the runs over all the test functions evaluated.

- **APSO** : Adapative PSO (APSO) [21], uses a classification procedure to determine the swarms' 'evolutionary state'. Using this determined state the PSO parameters, w, C_1 and C_2 are are adapted to aid the swarms performance in it current state. Four states and adaptive strategies are: Exploration state, increasing C_1 and decreasing C_2; slightly exploitation state, slightly increasing C_1 and slightly decreasing C_2; convergence state, slightly increasing C_1 and slightly increasing C_2; jumping-out state, decreasing C_1 and increasing C_2. This combined with a w taken as a function of the evolutionary state factor, f. The evolutionary state factor is calculated from the mean distance, dist_i, of

the particles from all other particles

$$\text{dist}_i = \frac{\sum_{j=1, j\neq i}^{N} \sqrt{\sum_{k=1}^{D} (x_i^k - x_j^k)^2}}{N-1} \tag{2.10}$$

this factor is then put into one of the four states via fuzzy clustering. The advantage of this methodology is that the swarm can appropriately increase and decrease it's diversity depending on its current state allowing it to effectively escape local minima and explore, but with faster convergence than normal PSO-lBest when in the vicinity of the global optima.

- **UPSO** : Unified PSO (UPSO) [30], is a simple attempt to try and exploit the different exploration and exploitation capabilities of PSO-gBest and PSO-lBest. In the UPSO scheme it uses both the constricted global best velocity update, $\mathbf{v}_i^{\text{gBest}}$, and the local best velocity update $\mathbf{v}_i^{\text{lBest}}$, by linearly combining to produce the final unified velocity update

$$\mathbf{v}_i^{\text{unified}} = u\mathbf{v}_i^{\text{gBest}} + (1-u)\mathbf{v}_i^{\text{lBest}} \tag{2.11}$$

where u is the unification factor and controls the balance between the global and local best updates. Out of the variations tested $u = 0.5$ gave the best performing results.

- **FDR** : Fitness-Distance-Ratio PSO (FDR) [45] aims to reduce particle oscillation and premature convergence by using an additional velocity term. The new additional velocity update term is based on maximising the fitness-distance-ratio for each dimension between the current ith particle and a selected particle j. For each dimension a new j^* is selected such that

$$j^* = \text{argmax} \frac{\text{fit}(\mathbf{x}_i) - \text{fit}(\mathbf{x}_j)}{|x_i^d - x_j^d|} \; ; \; j \neq i. \tag{2.12}$$

The additional velocity term for dimension d is then $C_3(x_i^d - x_{j^*}^d)$, where $C_3 = 2$. It should also be noted in FDR that for the original PSO velocity term

$C_1 = C_2 = 1$.

Other exotic variations look at improving swarm dynamics using techniques such sub swarms, speciation, attraction-repulsion and quantum particles, for example in Local Optima Avoidable PSO (LOAPSO) [46], two swarms are used are used to increase overall population diversity, the first swarm moves according to regular PSO while the second swarm moves in directions away from the current best of the first swarm.

2.1.1.4 Weaknesses in PSO

One problem that exists for all optimisation algorithms is the *no free lunch theorem*, this conjectures that there is no single algorithm that can perform well on one class of problems without compensating for performance elsewhere. For example in PSO it can often be seen that PSO-gBest is preferable for unimodal problems, whilst PSO-lBest is often preferred for multimodal problems.

As previously mentioned 'two steps forward one step backwards' [44] can result in an inefficient search. This is the case where to minimise the objective function the search progresses correctly in some dimensions whilst regressing with respect to others, this can create a zigzagging between the optimal solution vector. Methods such as CPSO and OLPSO heave been developed in an attempt to help resolve this issue by creating search processes that use information regarding the search with respect to each dimension and their relationships with each other.

Another known problem, which is well a documented phenomena of particle trajectory, is oscillation. Oscillations in the particles position occur due to the weighting of two different Euclidian distances in the traditional velocity update equation. The particle will zigzag between moving towards its own personal best and the global/local best positions. This can lead to inefficiency in the search progress and hinder convergence, this is in part the motivation for using a dampening inertia coefficient which can help reduce the particles velocity. Another way to address this element of the dynamics is to use exemplar learning [32] [34], used for example in CLPSO [32], in exemplar learning the velocity update equation is reduced to using a single Euclidian distance which removes the oscillation between

two points.

The appealing behaviour that PSO shows fast convergence towards the optimum can also lead to issues of stagnation and premature convergence. The diversity of the population can quickly shrink, reducing the explorative behaviour of the algorithm, this is part of the exploration-exploitation tradeoff. Stagnation occurs when all the particles have moved to the global best, and the global best does not change. In the case of multimodal problems this can lead the problem of converging towards local minima, algorithms such as PSO-lBest reduce the connectivity to retain more diversity and reduce the speed of global convergence. Other problems can occur for very flat functions where eventually the velocity decreases over time and the particles converge to sub-optimal value, this is seen in the optimisation of the Rosenbrock benchmark function. Algorithms such as CLPSO try to increase diversity by permuting the velocity, other algorithms such as APSO measure the diversity and use the jump-out state to allow the swarm to start exploring again. Other strategies such as simple restarts, and reinitlisation of particles [47] can be used.

Challenges also exist for solving functions that are non-separable, asymmetrical and have large number of local minima. These types of functions are represented by composite functions in the known CEC test suite. These problems have been observed to be extremely challenging for all different variants of PSO and even other types of algorithm [48] [49]. One problem is due to rotational invariance of the velocity update equation. Wilke *et al* [14] shows that the traditional PSO velocity update equation, Equation 2.3, is rotationally invariant due to the coupling between directional and magnitudinal diversity. The effects of rotating the search space can been seen in the results of Liang *et al* [32] which show that for all the PSO algorithms tested rotation leads to a large degradation in performance. Wilke *et al* [14] go on to suggest the DRI velocity which decouples the magnitude and direction by applying a new random rotation matrix to each the two velocity components, although this results in a very computationally slow method. The issue of rotational invariance has also be addressed in SPSO-2011 [37] and Locally convergent Rotationally invariant PSO (LcRiPSO) [50]. Rotational variance has been noted to also

cause a bias in the search behaviour of PSO [51] [52]. Spears *et al* [52] show that there a biases towards searching in directions parallel to axes, consequently PSO performs much better for search spaces where the trough or basin is parallel to the axes due to the fact the bias helps guides the particles along these features. Another search feature PSO has been shown to have problem with are where the global minima is situated far away from the local minima, this property is characteristic of the Schwefel function. In Liang *et al* [32] it was found for the Schwefel function that all the PSO variants considered, apart from CLPSO, failed to find the global minima and fell into the deep local minima. The relative lower degree connection between particle in CLPSO allowed particles to have greater exploration capabilities and avoid local convergence.

As a final note, one overlooked problem in the PSO literature is the lack of a clear direction and consensus within the literature, when compared to differential evolution, discussed next, there is no clear current best state-of-the-art PSO algorithm and a clear sense of research direction for improving and developing PSO.

As an alternative to PSO, differential evolution may be used.

2.1.2 Differential Evolution

Differential Evolution initially developed by Storn and Price [53] is based upon the process of genetic evolution found in nature. In population based algorithms it starts with a population of P n-dimensional vectors, $\mathbf{x}_i$, distributed over the search space, differential evolution then works by taking linear combinations of these vectors to try and create new better vectors; Algorithm 2.2 gives an outline of the Classic DE algorithm. The three main stages of DE are:

1. **Initialisation** : As with PSO the population in DE is initialised by randomly selecting a set of vectors from within the search space, most commonly a uniform distribution is used although as previously mentioned this may not be optimal for evenly covering the search space and other distributions or methods may be required [13] [54].

2. **Mutation** : In the mutation stage a donor vector, $\mathbf{v}_i$, is created to be used in the proceeding crossover procedure. The donor vector is created by taking linear combinations of scaled differences between other vectors in the population, there are various mutation strategies which are the most characteristic component of DE variations.

3. **Crossover** : Crossover creates a new trial vector, $\mathbf{u}_i$, by mixing the donor vector, $\mathbf{v}_i$, with the target vector, $\mathbf{x}_i$, using the crossover operator $C_{\otimes}(\mathbf{x}_i, \mathbf{v}_i)$. There are three common methods used for crossover: binomial; exponential; arithmetic.

4. **Selection** : The final step is selection, if the trial vectors objective function value is lower than the target vector objective function value, $f(\mathbf{u}_i) < f(\mathbf{x}_i)$, then the target vector, the existing member of the population, is replaced with the the trial vector, $\mathbf{x}_i = \mathbf{u}_i$, otherwise the target vector remains unchanged.

Algorithm 2.2 Pseudo code for an outline of the differential evolution.

```
For each x_i, and fit(x_i)
while Stopping criteria not met do
  for Each x_i ; i ∈ {1...P} do
    Create a donor vector v_i using chosen strategy
    Create trial vector u_i using crossover strategy, u_i = C_⊗(x_i, v_i)
    Calculate the trial vector fitness, fit_i = f(u_i)
    Replace x_i if trial vector is better, x_i = u_i if fit(u_i) < fit(x_i)
    Update global best, x_gBest = x_i if fit(x_i) ≤ fit(x_gBest)
  end for
end while
Output global best, x_gBest = argmin f(x_i) ; i = {1...P}
```

The mutation and crossover procedures are the defining stages of different DE algorithms, in the Classic DE formulation the mutation strategy combined with a crossover strategy produces the final DE variant and the standard nomenclature adopted to express the type of DE is *DE/mutation/number-of-difference-components/crossover*; for example DE/rand/1/bin is Differential Evolution using the *rand/1*, uses the random mutation strategy with one difference component, and

a binomial crossover. The mutation procedure for creating a donor vector $\mathbf{v}_i$ can be generalised

$$\mathbf{v}_i = \mathbf{x}_j + F \sum_{n=0}^{N-1} \mathbf{x}_{r_{2n}} - \mathbf{x}_{r_{2n+1}} \tag{2.13}$$

where j is the index of the parent vector selected by the mutation scheme, N is the number of difference components, r_x are mutually exclusive random numbers, $r_x \in [0, NP-1]$ which select random members of the population to generate the difference vectors and F is the mutation factor parameter. Three popular mutation strategies used to create the donor vector are:

- **DE/rand/1/** : The donor vector is created as linear combination of three random vectors selected from the current population, *rand/1* stands for random vector strategy with one difference component and $\mathbf{x}_j$ is selected as another mutually exclusive random vector from $j \in [0, NP-1]$.

- **DE/best/1/** : In this case $\mathbf{x}_j$ is selected such that it is the population member with the best fitness at the current iteration, $j = \text{argmin fit}(\mathbf{x}_j)$

- **DE/current-to-best/1/** : This strategy is slightly different to the others, the number of difference components is $N = 2$, but r_0 and r_1 are replaced by $r_0 = \text{argmin fit}(\mathbf{x}_j)$ and $r_1 = i$. In this instance the first difference component is the difference between the currently selected parent vector i and the current best population member. The perturbed vector $\mathbf{x}_j = \mathbf{x}_i$.

Other popular variations are DE/ * /2 strategies, where two equally weighted difference components are used in the above strategies. The mutation strategies can be divided into four classes [55]: 1) Rand, where no information of solution is quality is used in creating the donor vector and all the vectors are randomly selected, for example DE/rand/1; 2) rand/best, this is defined as a chaotic local search which uses information of a best known vector in the population, for example DE/current-to-best/1; 3) rand/dir, these strategies slightly differ from the previous two groups as they use values of the objective function to find a good direction in which to

mutate, an example is given in [55] where the population is separated in half based on objective function values, a difference component is then calculated using the centres of these two groups, or for example, in trigonometric mutation [56], or [57]; 4) rand/best/dir, which combines the use of a 'best' vector with a directional difference component.

An important property of the DE algorithm is contour matching [58]. This is a product of the mutation procedure and the difference components used. Contour matching is where the algorithm self-adapts to the fitness landscape, this is illustrated by how the distribution of the difference vectors produced clusters with different areas of the search space. The use of difference components also promotes basin to basin transfer, allowing members of the population to escape from local minima and move towards the global basin.

After mutation crossover is often introduced using the crossover operator $C_{\otimes}$, mutation only DE can be used but is not advised, the three main crossover strategies: binomial, exponential and arithmetic are defined as:

- **Binomial crossover** : This is the simplest crossover strategy. In binomial crossover for each dimension d the trial vector either takes on the value of the parent vector x_i^d or the donor vector v_i^d.

$$u_i^d = \begin{cases} x_i^d, & \text{if } r_i < \text{Cr} \\ v_i^d, & \text{otherwise} \end{cases} \tag{2.14}$$

  where Cr is the crossover rate parameter and r_i is a uniformly distributed random number $r_i \in [0,1]$. The Crossover Rate parameter, Cr, determines the degree of mixture and has an important effect on the search ability of the algorithm.

- **Exponential crossover** : In exponential crossover the crossover rate is used slightly differently compared to binomial crossover and determines the number of sequential elements l that are taken from the donor vector. First a random element d is chosen as the start point for crossover in the parent vec-

tor x_i^d, the elements $x_i^{(d+l)\%P}$ $l \in 1,2\ldots P$ are replaced by the donor vector $u_i^{(d+l)\%P}$ in a sequential and circular manner (where % is the modulus operator) until the crossover stopping criteria is met: either $l = P$ or $r_l >$ Cr, where $r_l \in \mathbf{U}[0,1]$ is a uniformly distributed number. This is shown is Algorithm 2.3. The advantage of exponential crossover is that is preserves some of the original structure of the two solutions.

Algorithm 2.3 Exponential crossover

$\mathbf{u_i} = \mathbf{x}_i$.
$d = \mathbf{U}[1\ldots P]$
$u_i^d = v_i^d$, $l = 1$
while $l < P || r_l >$ Cr **do**
  $u_i^{(d+l)\%P} = v_i^{(d+l)\%P}$
  $l++$
end while

- **Arithmetic crossover** : This is the least popular of the three strategies used in DE, although it will be seen in Section 3.1.1 that it is popular in EA hybrid methods. Arithmetic crossover takes a random linear combination of the parent and donor vector in each direction

$$u_i^d = r_d x_i^d + (1 - r_d) u_i^d \tag{2.15}$$

  where r_d is a uniformly distributed random number $r_d \in [0,1]$.

The crossover rate Cr is an important parameter for binomial and exponential strategies; although the same Cr value may be applied to different crossover strategies they have considerably different effects on the value of p_m [59]. Binomial crossover is the most popular crossover method, and was seen to perform better than exponential in the study by Montes [60]. Though, a more detailed comparative study of crossover has been done by Zaharie [59] [61]; the observed difference in performance for the crossover strategies for the same Cr values is due to the relationship of Cr to p_m. For binomial p_m varies linearly whilst non-linear behaviour occurs for exponential which results in a higher density of small p_m values for Cr $\in [0,1]$, and this behaviour should be taken into account when comparing Cr values for the

different strategies. When this is taken into account and the probability p_m is the same, both strategies were seen to perform very similarly, with perhaps exponential performing slightly better on rotated due to better preservation of solution structure but being harder to find a suitable set of parameters for high-dimensional problems.

Crossover can be thought to aid the search by introducing additional diversity into the population but when a high amount of crossover is used, i.e a low Cr value which means that there will be a larger mix of the donor and target vector, the important contour matching behaviour is in fact lost due to the additional randomness introduced into the trial vector from crossover. This introduces a few problems such as rotational variance and a search bias along the axes [62]. This can be useful for separable functions but not so in the case of non-separable and rotated problems. Despite this crossover is still used due to the benefit of the increased population diversity. Without so the DE algorithm can quickly stagnate due to only a limited set of potential donor vectors being created with mutation only.

The settings of the control parameter values, the mutation factor, F, crossover factor, Cr, and population size. N, have a significant effect on DE performance [63]. Originally in classic DE Storn and Price [64] suggest that F should be initially set to 0.5 and Cr should be tried first at $[0.9, 1]$ to see if quickly converging solution can be found, and if then to try 0.1. With respect to mutation only DE it has been observed that F is a function of the dimensionality [65] [66]. Control bounds of F have been derived by Zaharie [67] as a means of controlling population variance, where F is bounded by

$$F > \sqrt{\frac{\left(1 - \frac{\mathrm{Cr}}{2}\right)}{P}}. \tag{2.16}$$

It should also be noted that for exponential crossover a larger F has to be used to have the same effect on population variance [61]. Rather than useing fixed control parameter values, *dither* and *jitter* methods use F taken from a probability distribution [67] [58]. In dither F is applied as single value whilst dither F becomes a D-dimensional vector where a new random value is chosen for each component.

This removes the need for hard setting a parameter value, and results in better performance. A disadvantage is that jitter and dither causes mutation to become rotationally variant, with jitter introducing an additional rotation.

With respect to the previous discussion regarding crossover negatively affecting the contour matching property it can be assumed that a large Cr, resulting in a small amount of crossover is preferable especially when the problem is non-separable; this is in line with Ronkkonen *et al* [68] who suggest that Cr should be between [0,0.2] for separable functions and [0.9,1] for non-separable functions. Although the affect of Cr has also been studied by Penunuri *et al* [69], (for DE/rand/1/bin with dither) which seems to show that a Cr value of [0.2,0.4] is reasonable for most test functions used, separable or non-separable however this could be due to the fact that rotational invariance is lost when using dither and search success becomes more dependant on a large diversity to explore. In other self-adaptive methods, for example in SaDE [70], Cr is selected from an initial probability distribution which is then updated after a given number of generations with respect to better fitting the parameters values that have successfully generated good trial vectors.

As well as the two control parameters population size P can also have a large influence on DE performance, with a population too small there will be a severe lack of diversity resulting the population to be more likely trapped sub-optima, whilst too large will result in slower rates of convergence. Although there has been no general concensus on the ideal population size to use, in general the population is around 4-10D, though for high-dimensional real world problems optimal population size is highly dependent on the problem and algorithm [71]. Adapted population size methods are preferred [71] as they can allow for a large initial population to encourage initial exploration and then reduce the population size to aid convergence [72], this methodology has recently been applied to the already powerful SHADE algorithm [73]. Penunuri *et al* [69] suggest an interesting method of adapting the population with respect to the problem complexity using the Shannon entropy of the objective function, although interestingly conceptually the authors fail to take

into account the initial requirement of fitness evaluations to determine the problems complexity in their analysis.

The main issue with Classic DE presented here is that its performance is heavily dependant on finding optimal control parameters and also determining the best mutation strategy to use, as with PSO, there are many exotic and advanced variations of the classic algorithm in an attempt to develop a better all-round robust optimiser.

2.1.2.1 Advanced Variations

There are two main branches of research for advanced DE variations have been identified [74]. The first branch looks at improving the DE within the existing algorithmic framework, for example by improving the mutation or integration local search features. The second branch of advancement on Classic DE, which is currently the most fruitful, is the introduction of self adaptation mechanisms to create a robust optimiser to try and remove the weakness of parameter dependant performance. An overview of a few of the most important landmark variants in the development of DE are presented next.

- **DEGL** : Differential evolution global local (DEGL) [75] was developed not with the focus of self-adaptation but is worth mentioning as it offers a novel contribution by using a local topology similar to PSO. In the original DE algorithm all of the population is used, in terms of the previously discussed PSO literature it could be said that DE traditionally uses a global topology. It has been suggested that, as with PSO, a global topology may not be optimal in the case of multi-modal problems.

- **jDE** : The jDE algorithm [76] is a self-adaptive algorithm for the parameters F and Cr. It provides the novel contribution that the two control parameters F and Cr are incorporated into the search DE search process. The search space is extended by an extra two dimensions to accommodate the two control parameters, although they do not undergo the standard DE procedure. The parameters do not undergo mutation and during a binomial like crossover either

the parent control parameters carry over into the trial solution or with a probability of 0.1 are given new random values $F \in [0.1, 1]$ and $\text{Cr} \in [0, 1]$. The original jDE algorithm uses a rand/bin/1, jDE-2 uses two different mutation strategies [77].

- **JADE** : JADE [78] is an important step in DE progress and provides the template for the powerful L-SHADE algorithm. JADE introduces two new concepts: the first is using the strategy *DE/current-to-p-best/1*; and the second is using an external archive to hold some of the replaced vectors, this allows retention of old information. DE/current-to-p-best/1 without archive is similar to the DE/current-to-best strategy, but instead of using the global best solution, $\mathbf{x}_{\text{gBest}}$ in the first difference component it is randomly selected from the set of the top $p\%$ of solutions

$$\mathbf{v}_i = \mathbf{x}_i + F_i(\mathbf{x}^p_{\text{best}} - \mathbf{x}_{r_1}) + F_i(\mathbf{x}_{r_3} - \mathbf{x}_{r_4}). \tag{2.17}$$

  The archive works by storing the failed parent vectors from selection, once the acheive reaches a set size of P solutions, randomly selected members of the archive are deleted to keep it at size P. DE/current-to-p-best/1 with archive modifies the second difference term, where $\mathbf{x}_{r_4}$ is selected from the set $\mathbf{P} \cup \mathbf{A}$ where $\mathbf{P} = \{\mathbf{x_1} \dots \mathbf{x_P}\}$ is the current population and $\mathbf{A}$ is the set of archived solutions. Finally JADE uses an adaptive parameter strategy that draws the parameters F_i and Cr_i from a probability distribution with a weighted mean using previously successful parameter sets.

- **SHADE** : The Success-History based Adaptive DE (SHADE) variant is considered the current state-of-the-art DE, currently dominating the international CEC optimisation competition. In fact in the 2016 CEC competition there were five (out of a total of nine algorithms) SHADE variations entered for the CEC14' benchmark competition [79] with L-SHADE-epsin winning, and for the CEC15' benchmark problems [49] CCLSHADE and L-SHADE44 placing 2nd and 3rd, although first place was MVMO-PHM, but MVMO-SH was

beaten in 2015 by SPS-L-SHADE-EIG [48].

SHADE is an extension of the JADE algorithm and aims to increase the robustness of the parameter adaptation.The major contribution of SHADE is adding an additional memory archive for previously good parameter settings, the second contribution is that the greediness parameter p is initialised independently for all population members creating additional diversity.

The history based mechanism stores H sets of the successful mean Cr and F parameter values. For each individual the parameter values Cr^i and F^i and calculated by picking a random set from the history, and then drawing the parameter values for the respective distributions.

It is worth briefly mentioning the features of SHADE extensions. L-SHADE is a simple extension of SHADE using a linearly decreasing population [80], the population decreasing method of Brest *et al* [72] was also investigated in D-SHADE. L-SHADE was clearly superior to the basic SHADE algorithm and also superior to D-SHADE for lower dimensional problems (10-30D) whilst D-SHADE showed slightly better performance for 100D problems. Cooperative co-evolution L-SHADE (CCLSHADE) [81] uses the cooperative approach as seen in CPSO but using L-SHADE for both the cooperative search and sub-searches. L-SHADE44 [82] (a further development of L-SHADE4 [83]) uses the pooling of mutation strategies, similar to EPSDE. The four strategies [3] used in L-SHADE44 are current-to-pbest/1/bin, current-to-pbest/1/exp, randrl/1/bin, and randrl/1/exp.

Later on in Section 4 hybrid L-SHADE algorithm, PSO-LSHADE-NM , where NM is the Nelder-Mead search is introduced.

Within DE there has been a reasonably clear path of development building up from jDE→JADE→SHADE leading up to the current state-of-the-art L-SHADE algorithms.

[3] it is worth mentioning that the authors have taken into account the analysis of Zaharie [59] [61] and adapt p_m rather than Cr directly when using the two different crossover strategies

2.1.2.2 Weaknesses in DE

Several areas of development for DE have been highlighted in the recent review by Das *et al* [84]. The main weakness being rotational invariance which is inherently introduced by the use of crossover. This issue has often been overlooked due to the significant benefits. Even in the state-of-the-art L-SHADE rotational invariance is still an issue, one remedy has been the introduction of using a rotated crossover operator , this has been introduced into the SHADE algorithm in SPS-L-SHADE-EIG [85]. It has also been discussed that with high rates of crossover although this introduces population diversity it can lead to undesirable traits such as rotational variance and axes bias.

The main weakness of DE algorithms is in search spaces similar to the deceptive function [86]. The deceptive function is characterised by a relatively flat area area separated from the global optimum by a long and steep barrier. DEs are greedy algorithms and do not let the population move to areas of worse fitness, unlike PSO, this means that the algorithms can't step up this barrier to search around the area of the global optimum. It is found in the next section that even L-SHADE has a success rate of 0%, whilst PSO algorithms can have up to 100% success.

Although performance of self-adapting DE algorithms is impressive, analysis by Tanabe *et al* [87] use a Greedy Optimal Algorithm (GOADE) as a benchmark of optimal DE performance (using a by rand/1/bin strategy) to show that the mechanisms of the algorithms (jDE, EPSDE, JADE, MDE, and SHADE) still do not find the optimal parameters and performance that may be achievable.

2.1.3 DE vs PSO

Differential Evolution and Particle Swarm Optimisation are very much two competing methods in optimisation, this has led to the growth of a competitive community where sets of artificial benchmark functions are formulated to test and compare different behaviours and performance of the algorithms. Out of these benchmark studies yearly optimisation competitions at CEC and GECCO conferences have been established.

Within the specialist heuristic and EA optimisation community, both the CEC

and GECCO BBOB competitions PSO entries have wilted away over the years, in fact since 2008 for the CEC Real-Parameter Optimisation there have only been a handful of PSO entries, in 2013 there was SPSO2011 and fk-PSO and in 2015, there was SaDPSO and DMSPSO, but in all cases the PSO algorithms ranking mid to near the bottom in final results when compared the other algorithms. In the GECCO BBOB the only PSO entries made were in 2009, and since then the field has been dominated by variants of DE, CMA and MVMO. This begs to ask the question of why there has been such a decline in the popularity of PSO compared to DE within the EA research community. Even within more general literature searches on ScienceDirect or GoogleScholar for the terms 'particle swarm optimisation' and 'differential evolution optimisation' show a significantly greater interest in DE, for example from ScienceDirect in 2016 there were 3,362 and 4,575 results respectively, confirming a strong preference towards DE which has been an ongoing trend since their introductions in the early 90's.

One of the main factors is performance, and as mentioned in these renowned competitions PSO has failed to place favourably in any, whilst in CEC variants of DE continues to dominant the top ranks. For the BBOB problems DE is still beaten by CMA algorithms, although Tanabe *et al* [87] has shown that using GOADE, DE has the potential to further maximise performance and and produce results that are similar if not better than other heuristic optimisation methods such as top performing CMA and IPOP variations. Comparing results of L-SHADE, which here is considered to be state-of-the-art DE, with some of more recent competitive PSO algorithms, APSO, OLPSO and CLPSO on the CEC'14 benchmarks [86] [73] L-SHADE is clearly the superior algorithm on all the functions considered and also achieving the absolute minimum (mean fitness value of zero over 50 independent runs) on a handful of the easier functions, whilst only OLPSO achieved this for $F8$. Although none of the algorithms perform particularly well on the hybrid and composite functions, and still prove to be the most challenging. Other studies directly comparing PSO and DE variants on benchmark functions.

It is clear then that state-of-the-art DE is currently leagues above what might

be considered state-of-the-art PSO with respect to performance in regards to the well known benchmarking functions, but conceptually PSO may have advantages for certain search space landscapes when compared to DE. Langdon *et al* [88] show that PSO is superior when the landscape is comprised of multiple local optima and the global optima is located near the boundaries,this is particularly true when the initialisation space is asymmetrical and may not include the location of the global optima, although in this study they are only comparing Original PSO and Classic DE. These observations are further confirmed by the Deceptive Function [89] and Linear Deceptive Function [86]. The difference is that PSO does not use an elitist selection strategy, this allows for PSO to temporarily move and explore through areas of bad fitness, whilst elitist selection does not allow for this behaviour. The Deceptive Function and Linear Deceptive Function are generalisations of features found in the rotated-shifted Ackely function (CEC'14 $F5$), for which in 30-dimensions , CoDE, jDE, ESPDE, SaDE, JADE and L-SHADE found challenging [86] [73], but at the same time similar poor results were also achieved for APSO, OLPSO and CLPSO [90], suggesting that although there is preferential behaviour for PSO in the simplified settings, the features in a more complex landscape generate an overall challenging function for optimisation algorithms.

To better illustrate this differentiating behaviour the Linear Deceptive Function has been implemented here and used to compare the advanced Differential Evolution SHADE and L-SHADE variants, and PSO variants: lbest-PSO; gbest-PSO; and CLPSO. Success is measured if the algorithm manages to enter the well of the global optimum which is located at $-0.2 < x < 0$, this is repeated for 200 independent runs with 5000 FE. The results are very enlightening, with L-SHADE scoring 0% success rate and SHADE 0.5%, which is inline with the majority of the results found for the classic DE strategies by Hu *et al* [86] (the best was 33% success by DE/rand/2/bin), showing that in these cases even the state-of-the-art still suffer from this weakness, albeit very specific. Running the experiments for PSO though lbest-PSO achieves 100% success although gbest-PSO only achieving 1% and CLPSO only 10%.

2.1.3.1 Structural Similarities

With respect to simplicity of implementation of Original PSO and Classic DE both algorithms are very similar and easy to implement, they only vary in how the population is updated. The two major differences between the paradigms of DE and PSO is the elitist selection in DE and the inertia weighting w in PSO. As discussed with respect to the Linear Deceptive Function, elitist selection of solutions may aid in speeding up convergence but this strategy limits its explorative nature whilst PSO allows for some leeway in searching through bad areas.

By removing inertia weighting from PSO velocity, the position update of a particle essentially becomes the addition of two randomly weighted difference components which in essence are similar to the DE strategies current-to-best and current-to-p-best. In the case of exemplar learning by removing the inertia weight the update becomes similar to current-to-rand/1. Hierarchical-PSO (HPSO) [91] is in-fact a PSO variant which does not use an inertia, $w = 0$, this relies upon the difference vectors alone. In HPSO-FAC, where $C_1 = C_2 = 2$ are constant the results on the numerical benchmarks is reportedly poor, this is unsurprising if the similarity to DE and the results regarding high a mutation parameter$F$ are considered; once time varying coefficients are used, HPSO-TVAC, which is now similar to a linearly reducing F in DE, performance is considerably improved.

With respect to simplicity DE has a slight advantage over PSO. DE requires only 3 control parameters, population size, mutation factor, F, and crossover rate, Cr, whilst PSO requires, population size, w, C_1, C_2 and a topology. There is also the issue of debate over the best PSO topology and its effects on performance, whilst for DE, apart from the case of DEGL, it has been accepted that a global topology is as standard. There is also less distinctive research in self-adapting PSO compared to self-adapting DE which has taken off as the next major research direction in the field. In PSO the research is still more orientated towards novel methods of improving particle interactions [92].

With respect to structural bias it has previously been discussed, along with rotational invariance, as an issue present in most variants of the algorithms, although

in general DE algorithms show this to a lesser effect than PSO. It has been discussed that only crossover introduces rotational invariance but this can be controlled by using low rates of crossover, whilst in PSO the source of invariance is due to the inherent nature to the velocity update equation and the random perturbations introduced. Although in both cases it is possible to derive rotationally invariant formations of both algorithms.

It is interesting though that a lot of the successful developments of DE have used elements inspired by the PSO paradigm. The first is the use of the current-to-best mutation which of course is the same as the PSO velocity update. The most compelling is in the recent state-of-the-art SHADE algorithms which use a current-to-p-best mutation strategy, which is very similar to PSOs' velocity update, and DEGL is inspired by the idea of local neighbourhoods seen in PSO-lBest. It is clear that although DE may have some performance advantages over PSO this can't be said without taking into consideration the inspiration PSO has contributed to the DE literature and the difference in their respective search behaviours, as such promising developments for both fields may be possible with hybridisation [93].

2.2 Neural Networks

Artificial neural networks (ANNs) are known as a class of universal approximators inspired by the connectivity of the brain. ANNs consist of simple computing units, known as the 'neurons', connected in a layered structure via numeric weights. The simplest neural network is a feed forward multi-layer-perceptron (MLP) network. Traditionally an MLP consists of an input layer, hidden layer/s, and an output layer. Mathematically the output of a individual neuron in a feed forward MLP can be given by

$$y_i^l = \phi\left(\sum_i \sum_j w_{i,j}^l y_j^{l-1}\right) \tag{2.18}$$

where y_i^l is the output of neuron i in the layer l, when $l = L$ this represents the network output layer and when $l = 0$ this represents the network inputs. Each neuron also has an additional input, $i = 0$, which is known as the bias this input stays

constants for all neurons in all layers and is equal to one. $f(x)$ is known as the activation or transfer function of the neuron, the activation function defines the mapping of the weighted neuron inputs to the neuron output. A number of different activation functions can be used, three popular choices are: the logistic sigmoid

$$\phi(x) = \frac{1}{1+e^{-x}}, \tag{2.19}$$

hyperbolic tangent

$$\phi(x) = \tanh(x), \tag{2.20}$$

and the soft-plus function

$$\phi(x) = \log(1+e^{x}). \tag{2.21}$$

The sigmoid is the most popular choice as it has the appealing property that the derivative $\phi'(x) = 1+\phi(x)$, the sigmoid and hyperbolic tangent also met the criteria of the universal approximation theorem that states functions should be monotonically increasing and bounded, the soft-plus function is not bounded. The soft-plus function and the soft-max function are popular in deep learning architectures for pattern and image recognition and classification, being an unbounded function makes the networks less susceptible to saturation problems (saturation occurs when the neurons become heavily weighted at the asymptotic limits) [94] [95] in classification problems.

2.2.1 Universal Approximation Theorem

The most important result relied upon in this thesis is the *universal approximation theorem* [96]. The universal approximation theorem states that given a monotonically increasing bounded function, $\phi(.)$, there exists a set of weights such that

$$F(x) = \sum_{i=1}^{N} v_i \phi(w_i^x + b_i) \tag{2.22}$$

can approximate a function $f(x)$ arbitrarily well as $\lim_{N\to\infty} |F(x) - f(x)| \to 0$. This implies that for function approximation tasks, such as the ones approached in this thesis, simple shallow neural networks using one or two hidden layers should be capable. The theorem states that the approximation exists, but it does not state how to find the optimal set of weights. Finding the optimal weight set relies on training procedures for which there are numerous methods and extensive research on the topic. One such method is using a metaheuristic based approach with evolutionary algorithms (EAs).

2.2.2 Training with Evolutionary Algorithms

Using EAs is not new concept used for training neural networks and originally GAs [97] were used; more recently apart from using PSO and DE other EAs such ABC [98], Cuckoo search [99] [100] and Ant colony [101] have been used for training. When training neural networks using EAs the objective function

$$g(\mathbf{W}) = \|F(\mathbf{y}, \mathbf{W}) - f(\mathbf{y})\| + \lambda \tag{2.23}$$

is minimised, where W represents the weight matrix of the neural network, $\mathbf{y}$ are the set of training patterns, $\|.\|$ is defined as some norm, and λ is a regularisation term. The heuristic nature of the fitness function makes it appealing in cases where for instance: the error norm defined is non-differentiable, when using more complex regulaziation terms or the activation functions are non-differentiable [102] [103].

Compared to back-propagation methods (BP), EAs have been shown to be better on simple problems; for a quadratic function approximation problem PSO was seen to achieve the same level of accuracy of BP in up to 6x less computation [104]. Compared to more advanced gradient based methods such as Levenberg-Marquadt (LM) the performance of EAs is debatable in some cases they have been observed to be worse, whilst DE can often be as good as LM, Ilonen *et al* [103] state that DE never performs worse than the initial optimum found by LM, but it is highly probable that it can find better if it exists. In studies by Piotrowski *et al* [105], [106] [107] using more advanced variants DEGL outperforms ALC-

PSO and CLPSO, EPUS-PSO was seen to be only slightly worse but performed better on larger networks. Relative performance between EAs and gradient based methods can be problem dependant, EAs tend to perform better on problems with prominent local minima [108] [109]; for PSO, using PSO-lBest with a pyramid topology showed the best compromise between convergence speed and avoiding minima [108]. One final advantage is that EAs are not sensitive to the initial guess [109] which possibly makes them more robust and again less susceptible to falling into local minima.

Some issues when using EAs for training neural networks have been highlighted. Although they can not directly susceptible to the problem of vanishing gradients [94] some issues with saturation have been observed [110] [111], this can also explain stagnation of DE populations during training [107], DEGL showed this to a lesser degree than other EAs. When high weights are achieved and the transfer function are at the asymptotic limits i.e. saturated, changes in the weight space have little effect on population fitness, and although diversity can remain large the population moves very little [107]. Saturation can occur because of the large space EAs can explore, compared to the areas of most change in the transfer function the area of the asymptotic limits is much larger and hence EAs are more likely to spend time in these areas. Contrary to this for PSO it was found that larger initialisation range [-1 1] compared to [-0.01 0.01] showed less saturation [111], this can be explained by when the initialisation range is [-1 1] the EAs are initialised over a larger region where the transfer function still has high gradients. It was found that reducing the velocity clamping $\mathbf{v}_{\max}$ parameter further helped to reduce saturation [111] . Other issues observed is the scalability with the number of neurons, as the size of the hidden layer increases the performance of the EAs quickly degraded; smaller population sizes seem to be more scalable [103]. It has also be seen for function approximation that robustness is sensitive to the algorithm parameters [103], although effect has not been tested for self-adaptive DEs.

Overall EAs provide a viable alternative to gradient-descent decent based methods for neural network training; EAs are preferable in situations when the

objective function is complicated or non-differentialable, and when training is susceptible to local-minima.

2.3 Financial Derivatives

This section assumes little previous knowledge of finance and aims to introduce the basic concepts of options pricing in an approachable and generalised manner, the interested reader is referred to [112] [113] for a more in-depth introduction. An *option* is a financial derivative contract that gives the holder the right but not the obligation to purchase/sell the underlying asset [113], denoted as S, once the contract expires or is executed at a time, T. Option contracts can either be *calls* or *puts*; when a call option is purchased the holder is effectively long on the underlying asset, whilst when a put is purchased the holder is short on the underlying asset. The option is then valid until the time of contracts expiry known as maturity, and in some special cases it maybe possible to execute the option before maturity which is known as early exercise.

When an option is exercised it allows the party to either purchase/sell the underlying asset at a pre-specified price determined in the contract. For some types of options this is price is a fixed value K, known as the strike price. Exotic options may employ more elaborate functions over the contracts life time for determining the price at which the asset, S_T, is purchased/sold at, this is generalised as function $g(S_T)$. The *payoff* of an option is how much the contract is worth when executed, if the contract is profitable then it has value $g(S_T) - K$, otherwise the contract would not be executed and it becomes worthless. The function $g(S_T) - K$ which defines the value of an option when executed is known as the *payoff function* denoted here as $I_{\mathrm{c}}(S,K)$ for call options and $I_{\mathrm{p}}(S,K)$ for put options

$$I_{\mathrm{c}}(S_T,K) = \max(g(S_T) - K, 0) = (g(S_T) - K)^+ \tag{2.24}$$

$$I_{\mathrm{p}}(S_T,K) = \max(K - g(S_T) - K, 0) = (K - g(S_T)^+ \tag{2.25}$$

where x^+ is introduced as a cleaner notation for the max function. The simplest type of option is the European option (sometimes referred to as vanilla options) and only

allows the option to be exercised at maturity, T. The payoff functions for European options where $g(S) = S$ are

$$I_{\mathrm{c}}((S_T, K) = (S_T - K)^+ I_{\mathrm{p}}((S, T) = (K - S_T)^+. \tag{2.26}$$

This defines the value of the option at the time of execution T, which does not equal the current value of a option at current time, t, with asset price S_t, due to the movement of the asset price between $t \to T$. This now defines the option valuation/pricing problem, given the current asset price S_t what will be the future value of the option at the time of execution given the behaviour of the asset price. The future asset price at time T can be given as its expected value $\mathbb{E}[S_T|S_t]$, from this the current price of an option, V, can be defined as *the discounted expected value at execution*

$$V = e^{-r(T-t)}\mathbb{E}[I(S_T, K)|S_t]. \tag{2.27}$$

The discounting factor $e^{-r(T-t)}$ is required to take into consideration the risk-free growth of an asset due to interest rates r. It can now be seen that the value of the option depends on the possible future values of S_T, it is the characterisation of the behaviour of the asset price, S, that defines the future expected value of an option, and defines the different models used to derive the option price V.

Three important terms when discussing option prices are: *in-the-money*, this refers to the case that if the option was executed at the current time, t, with underlying asset price, S_t, the option would be profitable; *out-the-money*, refers to the case when if executing the option at the current time it would not be profitable and hence worthless; *at-the-money*, refers to the final case when $S_t = g(S)$, at this point in time the option is worthless but it is on the boundary.

Exotic Payoff Functions

There are numerous other types of option contracts and important relationships between them, the interested reader is referred to [113], two popular path-dependant option contracts of interest are:

American : These have the same payoff function as European options but allow the holder to execute any time before maturity, t^*.

$$V_{\text{Am}} = e^{-r(t^*-t)}\mathbb{E}[I(S_t^*, K)|S_t]. \tag{2.28}$$

This corresponding to an optimal stopping stopping time problem to find the bet t^* to maximise the value of an option, for non-dividend paying assets, $q = 0$, the value of an American options is the same as a European, $V_{\text{Am}} = V$; $q = 0$; in addition a lower bound for the price of American options is that they are always at least as valuable as a European option $V \leq V_{\text{Am}}$. The additional complexity of the early exercise problem means that these options have no known analytical solution but can be modelled as a free-boundary problem. The valuation of American options is found either by approximate solutions [114] or numerical methods [115].

Asian : The payoff function for an Asian option, I^{Asn}, uses the average $A(T)$ of the underlying asset prices, $S_t \to S_T$, during the options lifetime,

$$I_c^{\text{Asn}} = (A(T) - K)^+. \tag{2.29}$$

Two types of averaging are considered *geometric* (gmc) and *arithmetic* (art),

$$A(T)_{\text{gmc}} = \exp\left(\frac{1}{T}\sum_{t=0}^{T}\ln(S_t)\right), \tag{2.30}$$

$$A(T)_{\text{art}} = \frac{1}{T}\sum_{t=0}^{T} S_t. \tag{2.31}$$

In practice the averages are sampled discretely but theoretically can be treated in the continuos case. Geometric averaging initially seems less intuitive but as will be seen in Section 2.3.1.2 under the assumption of geometric Brownian motion this corresponds to compounding the return series and has a analytical solution. For arithmetic averaging there currently exists no analytical solution and approximations or numerical methods are used.

2.3.1 Option Pricing Models

When modelling options prices many different approaches can be taken depending of the branch of mathematics used, but are characterised by the behaviour defined for the underlying asset price. The behaviour of an asset price can be thought of as a random process, where the price fluctuates up and down. The simplest model for the asset price behaviour is assuming it follows a Brownian motion. Rather than modelling the asset price directly, the behaviour of the asset price can be modelled implicitly by modelling the return series. Consider the log-return

$$X(u,t) = \ln\left(\frac{S(t)}{S(u)}\right) \tag{2.32}$$

where $X(0,t)$ is random variable with mean μt and variance $\sigma^2 t$. From this the stock price can be written as

$$S(t) = \ln(S(0))e^{\mu t + \sigma W(t)} \tag{2.33}$$

where $W(t)$ is known as a Wiener process and has distribution $\mathcal{N}(0,t)$.This defines the *geometric Brownian motion* model (GBM) of the stock price, an important property of GBM corresponding with asset prices is that it is non-negative. Using Itos Lemmas from stochastic calculus this can be written as the stochastic differential equation (SDE)

$$dS_t = \mu S_t dt + \sqrt{\sigma} S_t dW \tag{2.34}$$

where μ is known as the drift, σ is the volatility, and dW is Brownian noise $\mathcal{N}(0,t)$. This forms the fundamental understanding of the famous Black-Scholes options pricing model [116].

2.3.1.1 Black-Scholes Equation

The Black-Scholes equation [116], Equation 2.36, gives an analytical price for European call and put options given an initial asset price S_0 and parameters μ and σ.

The first assumption is that the asset price follows geometric brownian motion.

The second important assumption is there is no arbitrage in the market, this assumes that no-risk free profit can be made. In options pricing to satisfy no arbitrage conditions the asset price is changed from measure $\mathbb{P}$ to the risk neutral measure $\mathbb{Q}$. Risk neutrality essentially means that the stock price grows with interests rate so that a risk free profit can not be made from selling the asset and investing the proceeds at interest rate r.

Based on the GBM model of the asset price, Equation 2.34, using stochastic calculus and no arbitrage arguments the Black-Scholes PDE for an option can be derived as

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial V^2}{\partial S^2} + rS\frac{\partial V}{\partial t} - rV = 0. \tag{2.35}$$

Combined with the payoff function at maturity, $V(T) = I(S_T, K)$, this forms an *initial boundary* (IB) problem. The Black-Scholes IB problem can be solved analytically for the case of European options, the detailed are omitted here but as part of the analytical solution Equation 2.35 can be reduced down via similarity transforms to the heat equation found in fluid dynamics [112], which is a class of equations known as diffusion equations. For most other types of exotic options the solution to the PDE must be found numerically. The analytical Black-Scholes price, C_{bs}, of a European call option is given as

$$C_{\text{bs}}(S,t) = S\phi(d_1) - K\exp -r(\tau)\phi(d_2) \tag{2.36}$$

where $\phi(.)$ is the cumulative normal distribution, and d_1 and d_2 are given by:

$$d_1 = \frac{\ln(S/K) + (r - \frac{1}{2}\sigma^2)}{\sigma\sqrt{\tau}}$$
$$d_2 = d_1 - \sigma\sqrt{\tau}$$

For a set of asset prices, S, and time-to-maturities, $\tau = T - t$, Figure 2.2 shows the price surface for a European call option. The tick shape at $\tau = 0$ is the payoff

function, $(S-K)^+$, at maturity, and the surface shows how the value of the option changes over the contracts lifetime and converges towards the final payoff function as time gets closer towards maturity. An important point to note is the behaviour of at-the-money-options and near out-the-money-options which become more valuable at the time-to-maturity increases, this is because the the uncertainty of the final asset price S_T increases and there is more chance that it will become in-the-money when executed at expiry. The price of a European call option can be related to the price of a European put via the Put-Call parity relationship

$$C_{\text{bs}}(S,t) - P_{\text{bs}}(S,t) = S - Ke^{-r\tau}. \tag{2.37}$$

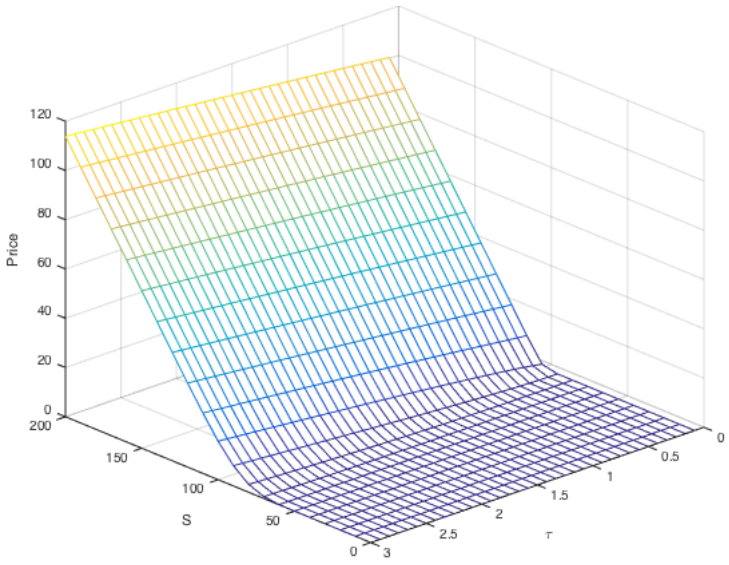


Figure 2.2: Black-Scholes option pricing surface for a European call option with $K = 100$, $\sigma = 0.1$ and $r = 0.05$.

2.3.1.2 Asian Averaging Options

Asian options are a popular example of an exotic path dependant option, for Asian options the payoff function, Equation 2.29, is a function of the average of the path values over the option's lifetime using either the geometric, Equation 2.30, or arithmetic average, Equation 2.31. In the continuous case the geometric average can be

given by

$$A(T)_{\text{gmc}} = \exp\left(\frac{1}{T}\int_0^T \ln(S(t))dt\right). \tag{2.38}$$

It can be seen that in the case of lognormal stock prices this corresponds to a compounding of the return series. Under the assumption of GBM it is possible to derive an analytical solution for geometric Asian options, given by Kemna-Vorst [117] (KV) in Equation 2.39. As it currently stands there are no known analytical solutions for arithmetic Asian options and pricing relies upon numerical methods to calculate E[Av]. The KV solution for Asian options are

$$C_{\text{kv}}(S,t) = e^{-r\tau}(\mathbb{E}[\text{Av}]\phi(d_1) - K\phi(d_2)) \tag{2.39}$$

where $\mathbb{E}[\text{Av}]$ is the expected path average, ϕ is the cumulative normal distribution. For the case of geometric averaging d_1 and d_2 are given by:

$$\begin{aligned} d_1 &= \frac{\ln(Se^{a\tau}/K) + \frac{1}{2}\sigma_G^2\tau}{\sigma_G\sqrt{\tau}} \\ d_2 &= d_1 - \sigma_G\sqrt{\tau} \\ a &= \frac{1}{2}(r - \frac{\sigma^2}{g}) \\ E[\text{Av}] &= Se^{0.5(r-\frac{\sigma_G^2}{6}\tau)} \\ \sigma_G &= \frac{\sigma}{\sqrt{3}}. \end{aligned}$$

Due to the averaging behaviour Asian options tend to be cheaper than their European equivalents.

2.3.1.3 Stochastic Volatility

In the Black-Scholes framework the asset price is taken to be a stochastic process, geometric Brownian motion, which has a constant volatility parameter, σ. In the stochastic volatility approach this assumption is removed and in line with real world observations the volatility of the asset price is itself a stochastic process. The

stochastic volatility is required to be introduced into the option pricing model as it helps explains certain pricing phenomenon known as the *volatility smile* [118]. The volatility smile is the observation that the implied Black-Scholes volatility, finding σ from option market prices, varies with time-to-maturity and strike price, where the implied volatility increases as $|S_t - K|$ increases. To derive the general stochastic volatility pricing equation the two SDEs for the asset price process and volatility, v, process are defined as

$$dS_t = \mu_t S_t dt + \sqrt{v} S_t dW_1, \tag{2.40}$$

$$dv_t = \alpha(S_t, v_t, t)dt + \nu\beta(S_t, v_t, t)\sqrt{v_t}dW_2. \tag{2.41}$$

The two processes are correlated by ρ, and where α and β are general functions which can be parameterised in accordance with given models. In a similar way to how the BSE is derived the pricing PDE for stochastic volatility models can be obtained [118]

$$\hat{L} = \frac{\partial}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2}{\partial S^2} + \rho\nu v\beta S\frac{\partial^2}{\partial v \partial S} + \frac{1}{2}\mu^2 v\beta^2\frac{\partial^2}{\partial v^2} + rS\frac{\partial}{\partial S} - r, \tag{2.42}$$

$$\hat{L}V = -(\alpha - \phi\beta\sqrt{v})\frac{\partial V}{\partial v} \tag{2.43}$$

where L is the differential operator and V is the value of the option.

2.3.1.4 Heston Model

A popular parameterisation of Equation 2.42 is the Heston model [4] which gives rise to a semi-analytical solution for European options. The Heston model assumes the volatility is a mean reverting process, the parameterisation used in the Heston model is $\alpha(S_t, v_t, t) = \kappa(\theta - v_t) - \lambda(S, v, t)$, where λ is the price of volatility risk, and $\beta(S_t, v_t, t) = 1$, these can be substituted into Equation 2.42 to get the Heston

PDE

$$\frac{\partial V}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho vS\frac{\partial^2 V}{\partial v \partial S} - \frac{1}{2}\mu^2 v\frac{\partial^V 2}{\partial v^2} + rS\frac{\partial V}{\partial S} - rV + [\kappa(\theta - v) - \lambda(S,v,t)]\frac{\partial V}{\partial S} = 0. \tag{2.44}$$

It can be shown that the Black-Scholes PDE is a special case of the the Heston PDE [119]. Setting this up for as an initial boundary problem for European options the price of a Heston Euronpean options can be given in a semi-analytical form that involves the evaluation of the complex integral. For the price of a European call option

$$C(S,K,\tau)_{\text{hstn}} = SP_1 - e^{-r\tau}KP_2 \tag{2.45}$$

where P_1 and P_2 are the in-the-money probabilities $P_j = \Pr(\ln(S) > \ln(K)$ but obtained under different measures. Writing P_j as a function of the characteristic function f_j the semi-analytical price of call option is

$$C(S,K,\tau)_{\text{hstn}} = \frac{1}{2}S_t e^{-q\tau} - \frac{1}{2}Ke^{-r\tau} + \frac{1}{\pi}\int_0^\infty \frac{e^{\mathrm{i}\omega \ln K}}{\mathrm{i}\omega}\left(S_t e^{-q\tau} f_1(\omega;x,v) - Ke^{-r\tau} f_2(\omega;x,v)\right) \tag{2.46}$$

where $x = \ln(S)$ and ω is the domain of integration. There exists an analytical form of the characteristic function f_j, meaning only the integral needs to be numerically evaluated. The main issue with the original form of f_j given by Heston [4] is that it leads to an unstable numerical integration with jumps occurring at discontinuities due to complex-logarithms [120]. To improve the stability of the integral many different forms of the characteristic function have been proposed [121] [122] [123] [124] [125], additionally the form given by Rollin *et al* [126] allows for an efficient evaluation of the derivatives with respect to the five model parameters which allows for fast calibration [127] methods. For a detailed and accessible discussion on numerical implementation of the Heston model the interested reader is referred

to [119].

2.3.2 Numerical Methods For Options Pricing

It has been seen that only under a very limited set of conditions can analytical solutions be found for the price of an options and instead numerical methods have to be used. Many different numerical methods have been used to solve the resulting pricing equations and initial boundary problems. Each have their own advantages and disadvantages based on simplicity, accuracy and speed. Here only a brief overview of the most common methods used will be given.

- Tree Methods : These are the simplest methods used and are rarely used in practice. The most common tree method is the binomial tree which is the starting point for most courses on options pricing [112], and models the asset price process as either rising by probability p to price u, or falling down to price d with probability $1-p$. The pricing starts at the final option prices based on the payoff and is then discounted backwards through the branches of the tree with calculated probabilities of price movements. Taking the continuous time limit of the binomial tree gives the BSE.

- Monte-Carlo : Monte-Carlo methods are one of the most popular methods used in options and derivatives pricing, with a vast amount of literature on the subject [128]. The basic concept of Monte-Carlo is that asset-prices are simulated using the underlying SDE over the contracts lifetime. From the simulated path the value of the option can b determined using the payoff function; this is then repeated many thousands of times and averaged to give a final expected payoff of the contract. The appeal of Monte-Carlo methods, apart from the name, is the simplicity in that no complex PDEs need to be solved or used and that only the final payoff function is needed, this allows complex asset price models to be used very easily. The major disadvantage is the accuracy, the accuracy of the resulting price is proportional to the number of asset price simulations and this number can grow very large and can result in a very computational expensive procedure.

- Finite Difference Methods : Finite difference (FD) methods [129] [130] are another type of discretisation similar to the tree methods. Though FD methods look at a larger range and finer granularity of asset prices and are used to approximate the solution of the discretised pricing equations. These methods are used due to the similarity between pricing equations and PDEs found in computational fluid dynamics. Like tree methods FD methods are backwards time stepping starting at the final payoff functions, known as the initial conditions. These methods can become very accurate given the appropriate level of discretization is used but can result in computational expensive linear algebra and complex discretisation equations which may then favour the use of Monte Carlo. These methods are the focus of this report and will be discussed in more detail later.

In this work 'less traditional' computational methods using evolutionary algorithms and customised hardware that could be used within this domain are explored.

Chapter 3

Breeding Particle Swarm Optimisation

In this chapter mechanisms and directions of research for improving the search capability of Particle Swarm Optimisation (PSO) are investigated. A self-adapting hybrid PSO using crossover and mutation is introduced. This hybrid PSO algorithm, dubbed Breeding PSO (BrPSO), mimics a breeding population of competing partners, and incorporates a breeding operator with mutation functions into the PSO framework to further aid population diversity and search exploration. The utility of BrPSO is demonstrated by comparing it against optimisation results of other powerful PSO algorithms over sets of well known benchmark test functions. In Chapter 4 BrPSO is then introduced as a tool for solving a challenging real world optimisation problem in finance.

3.1 Introduction

In light of the *no free lunch theorem* [131] and from previous discussions in Sections 2.1.1 and 2.1.2, neither particle swarm optimisation (PSO) or differential evolution (DE) offer the perfect all-round solution with each algorithm having different search properties and preferences for search space landscapes [88].

3.1.1 Hybrid Particle Swarm Optimisation

DE-PSO hybridisation is an active area of research which looks to combine the strengths of both families of algorithms to create a more robust optimiser. An in-

depth review of DE-PSO hybridisation is given by Xin *et al* [132] which defines three families of hybridisation:

Collaboration : In this model of hybridisation the DE and PSO algorithms are run separately retaining their original algorithmic structures but co-operate by sharing information through the population. In the literature reviewed by Xin *e*t al [132] sequential collaboration using bi-directional information (i.e. information is shared equally between both algorithms), was the most popular form of hybridisation. A typical collaboration strategy is where PSO and DE operations are performed on the population on different iterations, for example in one of the first DE-PSO hybrids [133] a combination parameter is used to determine the probability of PSO or DE occurring, in other cases they maybe applied in alternate iterations [134] or using a time dependant combination parameter. Other methods of collaboration involve sharing information between sub-populations via elitist selection of population members, such as sharing the global best with sub-populations being evolved by different algorithms [135], or in the co-operative framework proposed by Epitropakis *et al* [136] where DE is used to evolve a population taken as the set of personal bests from the PSO population.

Assistance : In the assistance methodology the focus is on using a second algorithm to aid in improving elements of the base algorithm, the key difference compared to embedded hybridisation is that the components of the assisting algorithm do not directly contribute to the fitness of the population. Typically the assisting algorithm is used to improve parameters of the base algorithm, this type of approach is seen in the adaptive PSO method DEAPSO proposed by Kannan *et al* [137], where DE is used as a sub-search to dynamically optimise the PSO velocity parameters, other examples of assisted adaptive behaviour will be discussed in Section 3.4.1.

Embedded : In embedded models components of the other algorithm are incorporated into the mechanisms of the base algorithm which influence the population fitness. Using this definition DEGL could be considered an embedded hybrid with PSO topology embedded inside DE. Other PSO inspired approaches discussed

in the DE literature, such as current-to-pbest mutation could also be considered weakly embedded approaches. Though most typically of embedded approaches DE is used as a perturbation for either the position [138] or the velocity values [139]. A noticeable example is Differentially Perturbed Velocity PSO (PSO-DV) [139], and PSO-DV has been further developed in Ageing-Leaders and Challengers PSO (ALC-PSO) [140]. PSO-DV applies DE to the particles' velocity, which is used as a base vector for mutation and donor vector for binomial crossover. Mutation uses two difference components, a random difference and current-to-p-best difference, and then finally selection is applied to pick the new trial particle position.

On the other hand although hybridisation appears to be a popular area of research it could be viewed that the literature with respect to hybridisation lacks a level of depth compared other other areas of EA research [141]. It is not uncommon for hybrids to feature as an additional 'novelty' to aid publication in application areas, consequently the performance of hybrid algorithms are often only tested on a limited set of very simple benchmark problems, they are not extensively compared with other algorithms, and their behaviour is not properly analysed. This makes it hard to gain proper insight into successful mechanisms of hybridisation and the additional benefits that can be produced. Furthermore, in these cases DEPSO hybrids are usually comprised of standard PSO and DE which questions the relevance of these hybrid methods with respect to the breadth and development of state-of-the-art PSO and DE algorithms.

This is not to say that there is not a body of respectable and relevant research within the topic of hybridisation. State-of-the-art algorithms have been used in the hybrid algorithms such as jDE [142], HPSO-TVAC [139], ALC-PSO [140] and HPSO-TVAC/SaDE [143]. A thorough analysis has been provided for the collaborative framework proposed by Epitropakis *et al* [136] and have comprehensively tested a variety of combinations of advanced PSO and DE variants over an extensive set of numerical benchmarks problems. In this proposed framework it was found that CLPSO provided the best PSO algorithm and was best combined with DEGL or JADE, however BBPSO/DE/rand/1 and FIPS:TDE/rand/1 also showed good per-

formance improvements. Apart from hybridising PSO with DE using both mutation and crossover operations a popular hybrid variation is to use only the crossover operation embedded within PSO.

3.1.2 PSO with Crossover

PSO with crossover only hybrids could also be considered a PSO-GA hybrid as well as DE-PSO, and in general fall into the embedded family of hybridisation. Crossover is embedded into PSO such that it is used to update a particles position. This can typically be applied similarly to when used in DE where a new trial vector is generated via crossover of members of the population to generate a new child particle. Three popular types of crossover used within PSO hybrids are Arithmetic, Discrete and Parent Centric.

Definition 3.1. The crossover operator, $C_{\otimes}(P) \rightarrow \mathbf{p_c}$, takes a set of selected parent vectors from the swarm, $P \subset S$, $|P| \geq 2$, with parent vectors $\mathbf{p}_i \in P$ and produces a child vector $\mathbf{p}_c$ as a mixture of the parent vectors defined by the type of crossover used.

Arithmetic : Arithmetic crossover (PSO-AX) uses a linear combination of two parent particles, for dimensions $d = \{1....D\}$

$$p_c^d = r_d p_1^d + (1 - r_d)\, p_2^d \tag{3.1}$$

where r_d is a uniformly distributed random number $r_d = \mathbf{U}[0,1]$. The most noticeable example, and often noted as the first crossover-PSO hybrid is given by Løvbjerg *et al* [144]. Arithmetic crossover is applied to introduce a breeding operation into the PSO algorithm, a new child particle replaces one of the parents is created by the arithmetic combination of two particles taken from a pool of randomly selected 'breeding' particles determined by the particles breeding probability (crossover probability). Settles *et al* [145] also employs the idea of a 'breeding' swarm and unlike other arithmetic crossovers utilises both the velocity and position information, dubbed VPAC. VPAC creates two children as an average of both parents minus a random weighting of either of the parents velocity vector, by us-

ing a negative contribution of the velocity vector it is aimed to increase population diversity. Unlike the 'breeding' population introduced by Løvbjerg *et al* [144] the population in [145] is kept constant by removing the worst $N\Phi$ particles from the population, where Φ is a parameter called the 'breeding ratio', and replaced with children bred using VPAC from the current remaining population. Chen [146] applies an arithmetic crossover where the new trial position is the mid-point between the particles personal best and a randomly selected particles' personal best position, crossover is used to replace both the velocity and update operations on every xth iteration. In the analysis some parallels are drawn with DMS-PSO. Zhang *et al* [147] uses arithmetic crossover to assist the PSO velocity update equation by generating a new trial position $\bar{\mathbf{x}}$ to use in place of the particles' position $\mathbf{x}$ in the PSO velocity update equation. In a dynamic population PSO Miao *et al* [148] uses arithmetic crossover of two current best population members to generate new particles if the population size needs increasing.

Discrete : Discrete crossover (PSO-DX) is synonymous with binomial crossover seen in DE, where for a dimension d the child vector is given as

$$p_c^d = \begin{cases} p_1^d, & \text{if } r_d < \text{Cr} \\ p_2^d, & \text{otherwise} \end{cases} \tag{3.2}$$

where Cr is the crossover rate parameter and r_d is a uniformly distributed random number $r_d = \mathbf{U}[0,1]$. The Crossover Rate parameter, Cr, determines the degree of mixture of parents in crossover. Discrete crossover has been applied in few different ways, Park *et al* [18] uses a discrete crossover with the particles' updated position and current personal best to try and create a better personal best position for the particle, i.e. only the personal best position is used in selection and replacement. Dong *et al* [149] applies a discrete crossover to update the particle positions after the position update, and uses the particles current position and global best as the parents. Extending on this Engelbrecht [29] [150] resets the particle velocity and personal best if crossover is successful, but also investigates using three different second parents: global best; personal best; an arithmetic combination of global and

personal best. Two different recombination schemes are also used, either equally weighted, or one point recombination.

Parent Centric : Parent Centric Crossover (PSO-PCX) [151], could be considered similar to the operations in DE because the final offspring is actually produced from a mutation procedure using a difference operator. The difference component is calculated as the distance, $\mathbf{d}_p = \mathbf{x}_p - \mathbf{g}$, where $\mathbf{g}$ is the centre of mass of a set of μ randomly chosen parent vectors and $\mathbf{x}_p$ is an individual parent vector randomly selected from the set. The vector $\mathbf{x}_p$ is then mutated as follows

$$\mathbf{x_c} = \mathbf{x}_p + w_\zeta |\mathbf{d}_p| + \sum_{i=1, i\neq p}^{\mu} w_\eta \bar{D} \mathbf{e}_i \tag{3.3}$$

where $\bar{D}$ is the average of the perpendicular distance of each of the $\mu - 1$ parent vectors to the the distance vector $\mathbf{d}_p$, and $\vec{e}$ are the orthonormal bases that span the perpendicular subspace. The resultant child vector $\mathbf{x_c}$ is centred around one of the parents, in comparison to other crossover operators which create a more uniformly distributed child vector in the subspace enclosed by the parents [151]. The PCX operations have some similarities to the modified velocity update equation introduced in LcRiPSO [50]. Parent centric crossover has been applied by Deb *et al* [152] [153] to replace the PSO update procedure rather than as an additional operation afterwards.

3.1.2.1 Embedding Crossover within PSO

Crossover can be embedded into PSO in different ways, in the first crossover hybrid by Løvbjerg *et al* [144] the crossover occurs after all the particle position and velocity updates have occurred; each particle is first marked if it is viable for crossover using the crossover/breeding probability $\mathbb{P}^i_{\mathrm{Br}}$.

Definition 3.2. The crossover/breeding probability (not to be confused with the crossover rate Cr) is the probability assigned to a particle, $\mathbb{P}^i_{\mathrm{Br}}$; $i = \{1...N\}$, that crossover will occur.

Algorithm 3.1 Pseudo code for PSO with embedded crossover

```
Initialise swarm of particles
Assign each particle with a crossover/breeding probability (ℙ^i_Br)
while not stopping citera do
  Update swarm
  for Each particle i do
    Particle position and velocity update
    if U[0,1] < ℙ^i_Br then
      C_⊗(P) → p_c
      if fit(p_c) < fit(p_i) then
        Replacement, p_i = p_c
      end if
    end if
  end for
end while
```

All of the marked particles are then entered into a pool of breeding particles, until the pool is empty two particles are selected and removed from the pool and arithmetic crossover is applied twice (using the same random numbers r_i but the order of the parents is switched) creating two children both children then replace the first parent of their respective crossover, i.e. $\mathbf{p}_1 = \mathbf{p}_{c1}$,$\mathbf{p}_2 = \mathbf{p}_{c2}$.

A more typical algorithmic template for embedded crossover in PSO is shown in Algorithm 3.1 [154], this can also be applied asynchronously with respect to the swarm update. The crossover operation occurs probabilistically with respect to $\mathbb{P}^i_{\text{Br}}$ after each of the particle position and velocity update, if the new particle has a better fitness then it is selected and replaces the current particle, although as has been discussed there are many different variations as to how particles can be selected and replaced. Another important variation is how the child velocity is handled in this select and replace model, Løvbjerg *et al* [144] creates a crossover velocity vector with respect to the two parents velocity vectors given as

$$\mathbf{v}_c = \frac{\mathbf{v}_1 + \mathbf{v}_2}{|\mathbf{v}_1 + \mathbf{v}_2|}\mathbf{v}_1 \tag{3.4}$$

whilst in the approach by Engelbrecht the velocity is reset to 0 and in Park [18] the velocity remain unchanged.

Arithmetic and discrete crossover are the most popular crossover operators embedded into PSO, part of their appeal could be due to their simplicity and efficiency for implementation; more objectively when compared to other crossover methodologies within Algorithm 3.1 discrete crossover with the current position and personal best (PSO-DX$_y$) was found to be the overall most robust hybrid [154] [141]. Next a new variant of a PSO-X hybrid is introduced.

3.2 Breeding Particle Swarm Optimisation

A hybrid particle swarm optimisation which incorporates a novel breeding operator using a discrete crossover operator is presented, this hybrid PSO algorithm is dubbed Breeding PSO (BrPSO) because biologically it imitates, to a certain degree, the competitive breeding of alpha males/females often seen in nature. The utility of BrPSO is then demonstrated and compared to other state-of-the-art PSO algorithms over a sets of well known benchmark test functions.

BrPSO is inspired by the performance improvements seen in other PSO-DX hybrids and elements of the CLPSO [32] algorithm. Compared to the PSO crossover algorithm described in Algorithm 3.1 and PSO-DX hybrids such as [18] [149] [29] [150] the main differences are:

1. A tournament procedure is used to select the second parent for crossover, the tournament procedure is between two randomly selected particles' personal best positions.

2. A mutation operator is additionally applied to perturb the child particle positions.

3. The set of breeding parameters are given a dynamic behaviour using a self-adaptive mechanism.

4. Crossover occurs before particle updates , the importance of this will be discussed in an analysis of particle behaviour.

Remark 3.1. Before proceeding it is worthwhile clarifying the notation used in the proceeding discussion. The notation used is consistent with the PSO notation used

in Section 2.1.1; $\mathbf{x}_i$ is the current position of the ith particle, $\mathbf{y}_i$ is the historical best position of the ith particle. When discussing crossover the notation introduced previously is used; $\mathbf{p}_i$ is the position vector of the ith parent and $\mathbf{p}_c$ is the resultant child position of the crossover operation.

BrPSO uses the idea of a tournament procedure, seen in CLPSO [32], for selecting a suitable parent particle from the population to be bred with the global best using discrete crossover. Tournament selection has also previously been used in GA crossover to apply selection pressure for the best solutions [155]. In the tournament two mutually exclusive particles are randomly chosen from the population excluding the global best particle, the particle with the best personal best position measured by the fitness function is then chosen for crossover

$$\mathbf{p}_1 = \min\left(\text{fit}\left(\mathbf{y}_{r_1}\right), \text{fit}\left(\mathbf{y}_{r_2}\right)\right) \tag{3.5}$$

where the two random particle indexes are $r_1 \in [1,N]$, $r_2 \in [1,N]$; $r_1, r_2 \neq \text{gBest}$, $r_2 \neq r_1$. The tournament procedure is beneficial as it filters out the potential for breeding with a bad solution and completely removes the probability that the worst particle in the swarm will be used for breeding and it will be seen later on in Section 3.3.5 that it increases the probability of a better child particle being created. The child particle is then created by using the discrete crossover operator $C_{\otimes \text{D}}$,

$$C_{\otimes \text{D}}\left(\{\mathbf{p}_1, \mathbf{y}_{\text{gBest}}\}\right) \rightarrow \mathbf{p_c}. \tag{3.6}$$

Once a child particle has been created mutations can additionally be applied as a perturbation to the new child particle position, these act to extend the potential search range of the child particle and are seen later to be an important factor in determining the algorithms performance. Finally replacement takes place, each particle in the swarm is assigned a crossover/breeding probability, see Definition 3.2, in the event that crossover occurs for a particle i, in BrPSO the particle is not directly used in the crossover (unless it is randomly selected as a parent in the crossover procedure) but it is then potentially replaced by the child particle

created using the crossover procedure in Equation 3.6, if the child particle has a better fitness than the current position of the particle, $\mathbf{x}_i$, then the position of the particle is replaced by the child, $\mathbf{x}_i = \mathbf{p}_c$ if $\text{fit}(\mathbf{p}_c) \leq \text{fit}(\mathbf{x}_i)$. It should be noted that if replacement is successful the velocity of the particle remains unchanged so as to act as a random perturbation when position updates occur. If the particle is replaced then the particle's personal best position is updated as required, the global best can either be updated synchronously, after all breeding has occurred, or asynchronously after replacement, in general asynchronous updating is used. The BrPSO algorithm is given in pseudo code shown in Algorithm 3.2, the base PSO algorithm is PSO-gBest with position and velocity updates occurring accordingly, although the base algorithm can easily be changed if desired.

Algorithm 3.2 Pseudo code for BrPSO

```
Initialise swarm as array of particles, set each particle with a random breeding probability (ℙ^i_Br)
swarm = new ParticleArray[N]
while not stopping citera do
    for every particle, particle do
        if U[0,1] < ℙ^i_Br then
            Select two random particles, p1, p2
            Compare the two random particles and select the particle with the best personal fitness
            for each dimension, d do
                if U[0,1] < Cr then
                    trial[d] = swarm[globalBest].bestPos[d]
                else
                    trial[d] = swarm[selectedBest].bestPos[d]
                end if
            end for
            if fitness(trial) < fitness(particle) then
                particle.pos = trial
                Update particle personal bests and swarm global best indexes
            end if
        end if
    end for
    for each Particle particle do
        Update particle velocity and position
        Evaluate particle fitness, update personal best and global bests if applicable
    end for
end while
```

The use of crossover means that a set of two additional parameters, the crossover/breeding probability ,$\mathbb{P}^i_{\text{Br}}$ see Definition 3.2, and the crossover rate, Cr, for each particle needs to be set. These parameters control the rate of additional

exploration as a result of crossover occurring and the formation of newly bred solutions. The crossover/breeding probability for each particle $\mathbb{P}^i_{\text{Br}}$ $i \in 1...N$ is an i.i.d random number from the uniform distribution $\mathbf{U}[0.05, 0.5]$. This is again another element borrowed from CLPSO for which the probability of an event is randomly distributed over all the particles. This allows the algorithm to retain a mixture of behaviours, for particles with a low $\mathbb{P}^i_{\text{Br}}$ they will behave more inline with regular PSO behaviour whilst those with higher probabilities will be more susceptible to crossover based behaviour. Under the condition that crossover/breeding probability is 0 for all particles BrPSO reduces down to PSO-gBest.

It has been observed that PSO-DX is particularly sensitive to the value of Cr [141]. The discrete crossover operator used by Dong and Yang [149], which is the most similar to the crossover used in BrPSO, has a preference for low Cr = 0.2 and 0.4 [141], throughout this work unless mentioned otherwise Cr in BrPSO is set fixed to 0.5 for all particles. Further analysis in Section 3.3 will shed some light as to why low values of Cr are favoured when using gBest based crossover and a value of 0.5 offers the best level of diversity.

3.2.1 Mutation

It is well known that mutations are an important part of DEs and genetic algorithms (GAs) as a means of extending the search range of the population and avoiding states of equilibrium, even a small mutation can significantly improve the quality of results. Mutations are introduced into BrPSO, with the mutated algorithms denoted as BrPSO($M_{\text{P}}, M_{\text{F}}$), where M_{P} and M_{F} are the two mutation parameters. The basic BrPSO algorithm with no mutation is therefore denoted as BrPSO(0,0). The mutation operation used in BrPSO differs from that used in the mutation stage of DE and does not require donor vectors from the population. The scheme used here is more analogous to the mutation functions found in GAs which use an underlying probability distribution. The mutation functions, denoted as $M(.)$, are embedded into BrPSO by setting a probability, given as the mutation probability parameter M_{P}, that after crossover produces the child position for each dimension that the position can be perturbed by applying the mutation function. Algorithm 3.3 presents

how the mutation function is added to the crossover procedure in BrPSO.

Algorithm 3.3 Pseudo code for mutation added to crossover in BrPSO

```
for each dimension, d do
    if U[0,1] < Cr then
        trial[d] = swarm[globalBest].bestPos[d]
    else
        trial[d] = swarm[selectedBest].bestPos[d]
    end if
    if U[0,1] < M_P then
        trial[d] = M(trial[d])
    end if
end for
```

The mutation function used is a random perturbation of the particle position. Considering a linear mutation function $M(.)_L$ given as

$$M_L\left(p_c^d\right) = p_c^d + p_c^d M_{\mathrm{F}} r_d \tag{3.7}$$

where M_{F} is the mutation factor and controls the scaling of the perturbation, $r_d = \mathbf{U}[-1,1]$ is a random uniformly distributed number, and p_c^d is the original position of the child particle in the d^{th} dimension. Having a similar effect Gaussian mutation could also be used where the random variable r_d is replaced by a symmetric Normal or Cauchy distributed random variable, as long as $\mathbb{E}[r_d] = 0$. In Equation 3.7 it can be observed that the mutation is centred around the current position with the expectation and variance (assuming $r_d = \mathbf{U}[-1,1]$)

$$\mathbb{E}\left(M_L\left(p_c^d\right)\right) = p_c^d \tag{3.8}$$

$$\mathbb{V}\left(M_L\left(p_c^d\right)\right) = \frac{1}{3}\left(p_c^d\right)^2 M_{\mathrm{F}}^2 \tag{3.9}$$

The mutation factor, M_{F} provides an additional degree of control over the range of the mutation area, and can allow it to be more finely controlled. By using the position p_c^d as part of the variance means that the size of the range can be scaled appropriately with respect to the current position, this assumes that in general the magnitude of the position is representative of the scale of the search area, this assumption and it's consequences will be discussed later on. Though, it is determined that mutation in the form of Equation 3.7 is only generally beneficial for an optimisation problem if M_{F} is either a decreasing function of time or a time adaptive

value rather than a fixed value throughout the whole optimisation. The introduction of mutation means large jumps in the particles' position may occur, although this can be controlled by controlling the variance using M_{F}, this behaviour may not be desirable in the case where convergence is required, therefore it is suggested that the mutation probability should also be considered as a time adaptive value.

The effectiveness of mutation parameters was also observed to vary dynamically with the problem landscape. Hence a final extension to BrPSO uses an adaptive mechanism governed by an independent internal PSO to dynamically learn the optimal mutation parameters for each of the particles; this method is dubbed BrPSO-Self-Adaptive-Mutation (BrPSO-SAM) and produces the overall best results compared to BrPSO(0,0) and BrPSO($M_{\mathrm{P}}, M_{\mathrm{F}}$).

3.3 Analysis of Crossover and Particle Behaviour

Particle swarm optimisation has many moving parts and stochastic components making it hard to derive a complete rigorous analysis, therefore simplifications are often introduced [156]. This section will attempt to provide some insight into the behaviour of BrPSO subject to certain assumptions and simplifications ascertaining to the probability distributions of velocity and position vectors. Examples will be given for a 2D search space that can easily be visualised and then generalised for n-dimensional space, this is accompanied by an empirical analysis.

The analysis will be introduced by considering Figure 3.1 which shows a simple 2D example of how BrPSO with and without mutation operates. For simplicity the domain is given as the unit n-ball, $\mathbf{S}_1^n$, it is then assumed that the fitness $F()$ is linearly proportional to the distance from the centre of the space, point O, $F(\mathbf{p}_i) = \alpha r_i$, where $r_i = \|\mathbf{p}_i\|_2$ is the distance of the particle from the optimum and α is a constant determining the gradient. Therefore it can be seen that each particle lies on the circumference of a circle with radius r_i, therefore for any particle to be better than another it must lie within the disc defined by r_i, $\mathbf{S}_{\mathbf{p_i}}{}^n$.

In this examples it can be seen that $F(\mathbf{p}_1) < F(\mathbf{p}_3) < F(\mathbf{p}_2) < F(\mathbf{p}_4)$, making $\mathbf{p}_1$ the global best particle. In Figure 3.1 particle 3, $\mathbf{p}_3$, has been selected for re-

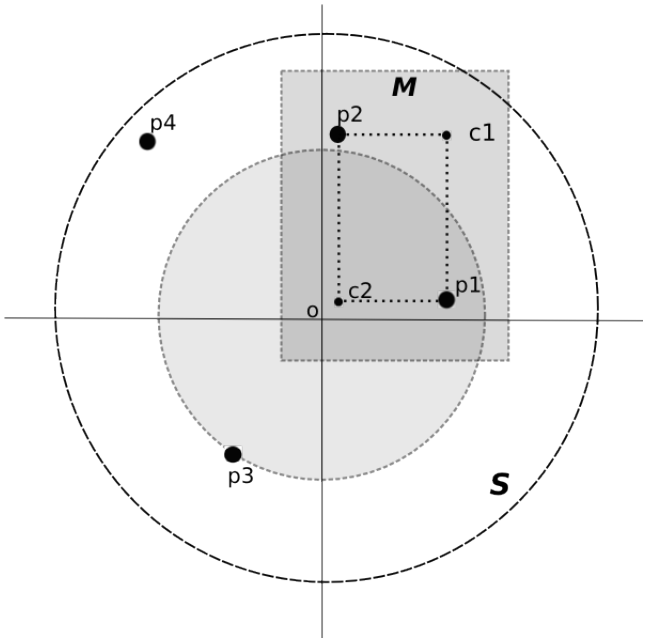


Figure 3.1: Example particle system of breeding particle swarm optimisation.

placement, therefore the child particle, $\mathbf{p}_c$ must be created within the shaded disc, $\alpha\|\mathbf{p}_c\| < \alpha\|\mathbf{p}_3\|$ for it to be successful. Two particles are selected to be entered into the tournament procedure, $\mathbf{p}_2$ and $\mathbf{p}_4$, to be selected for crossover with the global best. Given the better fitness (it shall be assumed here that current positions are also the best historical positions) $\mathbf{p}_2$ is selected.

Without mutation in this example it can be seen that the child particle created by breeding will either take on position of one of the points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_{c_1}, \mathbf{p}_{c_2}$. In the depicted scenario the parent $\mathbf{p}_2$ lies outside of the region enclosed by the replacement particle, $F(\mathbf{p}_3) < F(\mathbf{p}_2)$, this gives a 50% chance of improvement over $\mathbf{p}_3$s current position. If $\mathbf{p}_2$ was inside the region enclosed by $\mathbf{p}_3$ then there would be at least a 75% chance of improving the particles position.

This mechanism greatly increases the rate of convergence towards the minima as it can be seen the child particle has a good chance of resulting in an improved position. However using only discrete crossover means exploration is limited due to the limited possibilities of the child. Therefore mutation is introduced as a perturbation on the child particle's position, this extends the child's possible positions to a continuous space, defined in Figure 3.1 as **M**. This is advantageous as it encourages

additional exploration. For example it can be seen that an area of $\mathbf{M}$ is actually outside of the space $\mathbf{S}$ which demonstrates how mutation can aid in particles escaping local minima. In this case for the child to be successful it must be created within the darker shaded sector where $\mathbf{M} \cap \mathbf{S}_{\mathbf{p_3}}$. It can be seen that in the case of mutation it is possible to explore better areas of search space than the c_2 position created for crossover only breeding.

In the proceeding analysis the behaviour described in this example is generalised for the setting of an n-dimensional space, providing bounds for the determining the probability of crossover improving particle positions, as well as showing how crossover effects population dynamics.

3.3.1 Initial Definitions

The example presented in Figure 3.1 can be more formally described and generalised using the following definitions.

Definition 3.3. The position of the ith parent particle is denoted as the vector $\mathbf{p}_i$, p_i^d denotes the the dth component of $\mathbf{p}_i$ vector. $\mathbf{p}_{\text{gBest}}$ denotes the parent using the global best position of the swarm. $\mathbf{p}_c$ denotes the child vector that is the output of the crossover operator.

Definition 3.4. The n-ball with radius r is denoted as $\mathbf{S}_r^n$.

Definition 3.5. The n-dimensional hypercube with side length a is denoted as $\mathbf{H}_a^n$.

To begin with the domain of the analysis Ω^n is defined such that it allows for a simple geometric representation of a convex optimisation problem.

Definition 3.6. The domain of the search space Ω^n is a convex subset of the Euclidean space $\mathbb{R}^n$ and is the n-dimensional closed unit ball such that

$$\|\mathbf{p}_i\| \leq 1 \; ; \forall \mathbf{p}_i \in \Omega^n \tag{3.10}$$

$\Omega^n = \mathbf{S}_1^n$. Consider a global or local optimum is located at the center of a well, assuming the well is symmetric and centred at the origin the fitness function, $F(.)$, can simply be defined as a function of the particle's Euclidean distance.

Definition 3.7. The fitness function, $F(\mathbf{p}_i)$, of a parent particle $\mathbf{p}_i \in \Omega^D$, inside a well, W, $\mathbf{p}_i \in \Omega^n$ is calculated as the distance from the center of the well $\mathbf{w}$.

$$F(\mathbf{p}) = \alpha \sqrt{\sum_{1\ldots D}^{i} (x_i - w_i)^2} \tag{3.11}$$

where α is a constant defining the constant gradient of the well, from hereon it is assumed that $\alpha = 1$. Considering this fitness function it is equivalent to minimising the n-dimensional spherical benchmark function $f1$, with the optimum at the origin. It can also be viewed that the position of a particle $\mathbf{p}_i$ in Ω^n represents the contour of a function in $\Omega^{(n+1)}$.

Now assume there is a set Pop of N number of i.i.d particles, Pop$= \{\mathbf{x}_1 \ldots \mathbf{x}_N\}$ that are uniformly distributed over Ω^n.

Lemma 3.1. . *The initial global best position is bounded by the contour* $\sum x^2 = r_g^2$ *such that* $F(\mathbf{x}_{gbest}) < r_g$ *and* $\Pr[F(\mathbf{x}_i) \leq r_g] = \frac{1}{N}$. *Given that the particles are initially uniformly distributed over* Ω^D *and that* Ω^D *is the n-dimensional unit ball*

$$r_g = \sqrt{\frac{1}{N}}^D \tag{3.12}$$

This ensures that probabilistically only one particle will exist on or inside this contour and hence be the singular global best.

Defining Crossover

From the set Pop three particles are chosen to take part in crossover, where $\mathbf{p}_1 = \mathbf{x}_j$ and $\mathbf{p}_2 = \mathbf{x}_k$, $j \neq k$ are the two parent particles for crossover, and $\mathbf{p}_R = \mathbf{x}_R$ is the particle selected for replacement by $\mathbf{p}_c$.

The success of crossover is measured as the event in which the child particle has a fitness less than the replacement particle $F(\mathbf{p}_c) < F(\mathbf{p}_R)$. Define $r = F(\mathbf{p}_R)$, the probability of crossover success is given as $\Pr[F(\mathbf{p}_c) < r]$.

This analysis will focus on the used of the discrete crossover operator.

3.3.2 General Properties of Discrete Crossover

For discrete crossover between two n-dimensional vectors there is a limited set of possible child vectors.

Lemma 3.2. *Using discrete crossover with two particles in Ω^n results in 2^n possible child positions, these positions form the vertices of a n-dimensional hyperrectangle.*

Lemma 3.3. *With a crossover rate, Cr, the probability of a child position is*

$$\Pr[\mathbf{p}_c] = \prod_{1\ldots D}^{d} \mathbf{1}_{p_1^d}\left(p_c^d\right)(1-Cr) + \mathbf{1}_{p_2^d}\left(p_c^d\right)Cr \tag{3.13}$$

where $\mathbf{1}_A(B)$ is the indicator function that $A = B$. If z is the number of dimensions that are taken from $\mathbf{p}_2$ in the child vector, then

$$\Pr[\mathbf{p}_c] = (1-Cr)^z\, Cr^{n-z}. \tag{3.14}$$

The limiting behaviour of crossover with respect to the choice of Cr and the expected child vector.

$$\lim_{\mathrm{Cr}\to 1} \mathbb{E}[\mathbf{p}_\mathrm{c}] \to \mathbf{p}_2 \tag{3.15}$$

$$\lim_{\mathrm{Cr}\to 0} \mathbb{E}[\mathbf{p}_\mathrm{c}] \to \mathbf{p}_1 \tag{3.16}$$

finally if $\mathrm{Cr} = 0.5$ then $\Pr[\mathbf{p}_\mathrm{c}] = \frac{1}{\mathrm{Cr}}^n = 2^n$ i.e. all the possible outcomes are evenly distributed.

Lemma 3.4. *With no selection criteria if $\mathbf{p}_1$,$\mathbf{p}_2$ and $\mathbf{p}_R$ are uniformly i.i.d in Ω^n then $\Pr[F(\mathbf{p}_c) < F(\mathbf{p}_R)] = 0.5$ and is independent of the crossover rate .*

Next, the probability of breeding a successful child will be considered for two cases, firstly discrete crossover with two improved parents where $F(\mathbf{p}_2) < F(\mathbf{p}_1) < F(\mathbf{p}_R)$ and secondly discrete crossover with one improved parent $F(\mathbf{p}_2) < F(\mathbf{p}_R) < F(\mathbf{p}_1)$. These two scenarios are important given the selection criteria of parents used in BrPSO crossover, if there was no selection criteria the success probability can be simply inferred by Lemma 3.4.

3.3.3 Bounds For Crossover With Two Improved Parents

Here bounds and limits for the success of crossover given that $F(\mathbf{p}_2) < F(\mathbf{p}_1) < F(\mathbf{p}_R)$ are derived.

Lemma 3.5. *Given $F(\mathbf{p}_2) < F(\mathbf{p}_1) < F(\mathbf{p}_R)$ the child particle $\mathbf{p}_c$ is guaranteed to be better than the replacement candidate particle $\mathbf{p}_R$ if both $\mathbf{p}_1$ and $\mathbf{p}_2$ exists inside the an n-hypercube inscribed inside the hypersphere $\mathbf{S}_r^n$; $r = F(\mathbf{p}_R)$.*

Proof. Given both parent particles are inside a hypercube the child particle is also bounded by the hypercube given that vertices of the hypercube correspond to the maximum and minimum points that can be formed by crossover. Therefore inscribing an n-hypercube inside of $\mathbf{S}_r^n$ gives $\Pr[F(\mathbf{p}_c) < F(\mathbf{p}_R)] = 1$; $\mathbf{p}_1, \mathbf{p}_2 \in \mathbf{H}_r^n$ ■

From this a loose lower bound for the probability of successful crossover can be derived given that both parent particles need to be inside the inscribed n-hypercube.

Lemma 3.6. *The lower bound of the probability that crossover is guaranteed to be successful given $F(\mathbf{p}_2) < F(\mathbf{p}_1) < F(\mathbf{p}_R)$ is*

$$\Pr[F(\mathbf{p}_c) < F(\mathbf{p}_R)] \geq \frac{V_n(\mathbf{H}_r^n)^2}{V_n(\mathbf{S}_r^n)^2} = \frac{2^{2n}\Gamma\left(\frac{n}{2}+1\right)^2}{n^{\frac{2n}{2}}\pi^{\frac{2n}{2}}} \tag{3.17}$$

where $\mathbf{H}_r^n$ is the maximum fully inscribed hypercube inside of the n-ball $\mathbf{S}_r^n$, and $V_n(.)$ is the n-dimensional volume operator (see Section A).

Proof. Using Lemma 3.5 the child particle will always be bound by $\mathbf{S}_R$ if both parents are inside a hypercube fully inscribed in $\mathbf{S}_r^n$. The probability of $\Pr[\mathbf{p}_i \in \mathbf{H}_r^n]$ is the ratio between the n-volumes of the maximum n-hypercube inscribed within the n-ball, this can be derived using Corollaries A.2 and A.1 which give the volume of the n-ball and the maximum fully inscribed hypercube respectively. The ratio is then squared given that both particles must be inside the hypercube. ■

A nice aspect seen in this lower bound is that it is completely independent of the radius of the n-ball and only depends on the dimensionality of the problem. It is

worth mentioning the limiting behaviour of Equation 3.17 with respect to dimensionality. Using Stirlings formula for the growth of the Gamma function it can be shown that $\frac{\partial}{\partial D}D^{\frac{D}{2}} > \frac{\partial}{\partial D}\Gamma(\frac{D}{2}+1)$, as such

$$\lim_{D\to\infty}\frac{V_D(H)}{V_D(S)} \to 0. \tag{3.18}$$

Equation 3.17 provides an extreme lower bound of specific behaviour where crossover is guaranteed to be successful, it is possible to define a tighter more general lower bound.

Definition 3.8. Let $K(n)$ be defined as the minimum proportion of all possible child particles that satisfy $F(\mathbf{p}_c) < r$, then

$$\Pr[F(\mathbf{p}_c) < r] > K(n) \tag{3.19}$$

Theorem 3.1. *Given $F(\mathbf{p}_2) < F(\mathbf{p}_1) < F(\mathbf{p}_R)$ the minimum proportion, $K(n)$, of all possible $\mathbf{p}_c$ that satisfy $F(\mathbf{p}_c) < r$ is given by*

$$K(n) = 1 - \sum_{z=1}^{n}\binom{z}{n} \tag{3.20}$$

Proof. Consider the case where $\mathbf{p}_1 = \{|r|, 0, 0 \ldots 0\}$ and define $\mathbf{p}_2$ such that $|p_2^d| > 0$ $\forall\ d = \{1 \ldots n\}$, then no matter what element $d > 1$ from $\mathbf{p}_2$ is combined with $\mathbf{p}_1$, the child particle will always lie outside of $\mathbf{S}_r^n$. The number of permutations of replacing $n-1 > z > 1$ elements can be defined by the binomial coefficient, but due to symmetry of the problem if an element of $\mathbf{p}_1$ replaces and element in $\mathbf{p}_2$ then the crossover will be successful, hence the number of permutations has to be halved. ■

Taking the limits with respect to the number of dimensions n

$$\lim_{n\to\infty} K(n) \to 0.5, \tag{3.21}$$

and the upper lower bound can be given in the case of n=2,

$$0.5 < K(n) \leq K(2) = 0.75. \tag{3.22}$$

Theorem 3.2. *Given $F(\mathbf{p}_2) < F(\mathbf{p}_1) < F(\mathbf{p}_R)$ the absolute lower bound for the crossover success rate is*

$$\Pr[F(\mathbf{p}_c) < r] > 0.5 \tag{3.23}$$

Combing the results of Theorem 3.1and Lemma 3.6 the form of a tighter lower bound is proposed.

Proposition 3.1. *A tighter lower bound of successful crossover can be given in the form of*

$$\Pr[F(\mathbf{p}_c) < r] > (1 - K(n)) \frac{V_n\left(\mathbf{H}_1^n\right)^2}{V_n\left(\mathbf{S}_1^n\right)^2} + K(n) > 0.5 \tag{3.24}$$

Proof. From the condition of Lemma 3.6 crossover is guaranteed to be successful and from Theorem 3.1the lower bound of successful crossover is given by $K(n)$, therefore when both parent particles are within the maximum inscribed hypercube the probability must go to one. ■

Furthermore, taking the lower bound of $K(n) > 0.5$ a lower bound of success rate can be given as in Theorem 3.3.

Theorem 3.3. *The lower bound of crossover being successful given $F(\mathbf{p}_2), F(\mathbf{p}_1) < F(\mathbf{p}_R)$ is*

$$\Pr[F(\mathbf{p}_c) < F(\mathbf{p}_R)] > 0.5 \frac{V_n\left(\mathbf{H}_1^n\right)^2}{V_n\left(\mathbf{S}_1^n\right)^2} + 0.5 \tag{3.25}$$

Using the limits in Equation 3.18 and 3.21 it can be seen that the limit of the proposed lower bound, Equation 3.24, with respect to dimensionality bound decays to 0.5. Hence for high-dimensional problems $n > 10$, crossover has at least a 50% chance of success when both parents are better than the replacement candidate.

A tighter approximate lower bound can be considered by assuming that the volume for which if only one particle is inside the of $\mathbf{H}_r^n$ the region for which the other particle must occupy such that the child particle lies outside of $\mathbf{S}_r^n$ is proportionally very small and could be consider negligible and as such the following approximation is acceptable

$$\Pr[F(\mathbf{p}_c) < F(\mathbf{p}_R)] \gtrapprox (1 - K(n)) \frac{V_n (\mathbf{H}_r^n)^2}{2V_n (\mathbf{S}_r^n)^2} + K(n). \tag{3.26}$$

Empirical Tests of Bounds

To test the validity of the proposed lower bound in Equation 3.26 this has been compared to Monte-Carlo simulations for the crossover success probability given $F(\mathbf{p}_2), F(\mathbf{p}_1) < F(\mathbf{p}_R)$. Figure 3.2 shows the empirical probabilities (for a fixed $\mathrm{Cr} = 0.5$) and theoretical bounds of Equation 3.26 using $K(n)$, the lower bound $K = 0.5$ and the upper lower bound $K = 0.75$. When $K = 0.5$ Equation 3.26 provides a suitable lower bound for all dimensionality, although the rate of decay towards the asymptote is faster than observed for the empirical results, when $K = 0.75$ it can be seen that Equation 3.26 provides a tighter bound for low dimensions but being asymptotic to 0.75 means this becomes unsuitable for higher dimensional problems. Using $K(n)$ as defined in Equation 3.20 shows a similar suitable bound for when $K = 0.75$ for lower dimensions, but still decays towards the lower bound of 0.5 to quickly.

This bound only takes into consideration one region where success is guaranteed, but by definition the volume of this region quickly decays to 0 with respect to dimensionality, this bound does not consider all the intermediate regions where $0.5 < K(n) < 1$. Further analysis within the domain of hypergeometry and combinatorics that are beyond the scope of this work is required to provide tighter lower bound estimates by considering more regions of crossover behaviour.

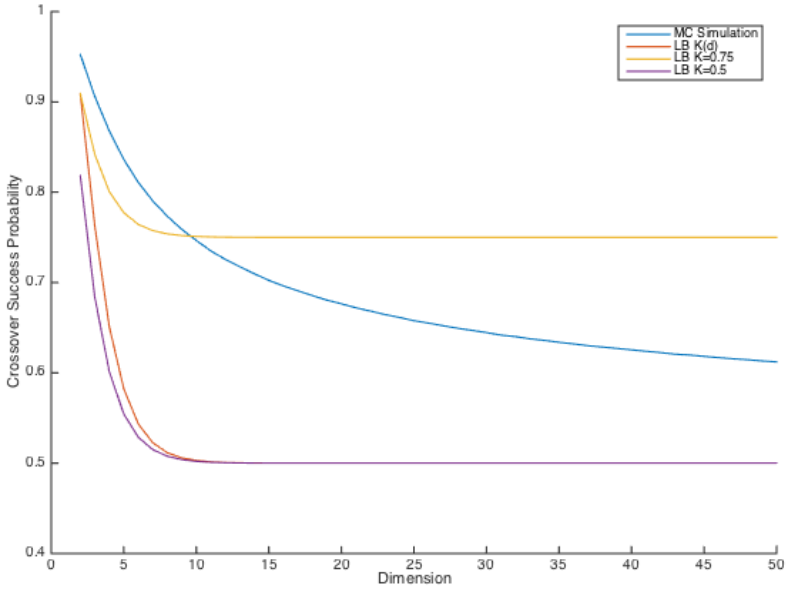


Figure 3.2: Crossover success probability estimation for $F(\mathbf{p}_2), F(\mathbf{p}_1) < F(\mathbf{p}_R)$ using Monte-Carlo simulation (10^5 samples, 50 replications, $\mathrm{Cr} = 0.5$) compared to lower bounds given by Equation 3.26 with $K = 0.5$ and $K = 0.75$.

Generalising for Crossover Rate

The above bounds consider the case when $\mathrm{Cr} = 0.5$ and all possible permutations of the child vector are equally as likely. Generalising this for when $\mathrm{Cr} \neq 0.5$ means that it shall be observed that by the definitions of crossover in Equations 3.15 and 3.16, and given that $F(\mathbf{p}_2), F(\mathbf{p}_1) < F(\mathbf{p}_R)$ then it can be seen that

$$\lim_{\mathrm{Cr}\to 1} K(n) = \lim_{\mathrm{Cr}\to 0} K(n) \to 1, \tag{3.27}$$

as the child converges towards either one of the parent vectors. As a result of this symmetry it can be observed that the minimum of $K(n, \mathrm{Cr})$ must occur for when $\mathrm{Cr} = 0.5$.

Overall it can be concluded that a suitable theoretical lower bound can be derived for the crossover success probability when $F(\mathbf{p}_2), F(\mathbf{p}_1) < F(\mathbf{p}_R)$ and a very accurate estimation of the behaviour can be given by using Equation 3.26 with a $K(n)$ given by Equation 3.20, this also provides a lower bound for all parameterisations of the Crossover Rate, $\mathrm{Cr} \in [0, 1]$. To complete the analysis the second

scenario for crossover now needs to be considered.

3.3.4 Bounds For Crossover With One Improved Parent

In this scenario $F(\mathbf{p}_1) > F(\mathbf{p}_R) > F(\mathbf{p}_2)$, the behaviour cannot be as well defined as the previous scenario due to the fact that $F(\mathbf{p}_1)$ is now unbounded within the domain. To analyse this behaviour Monte-Carlo simulations have been used. The crossover success probabilities with respect to the dimensionality and crossover rate are shown in Figure 3.3.

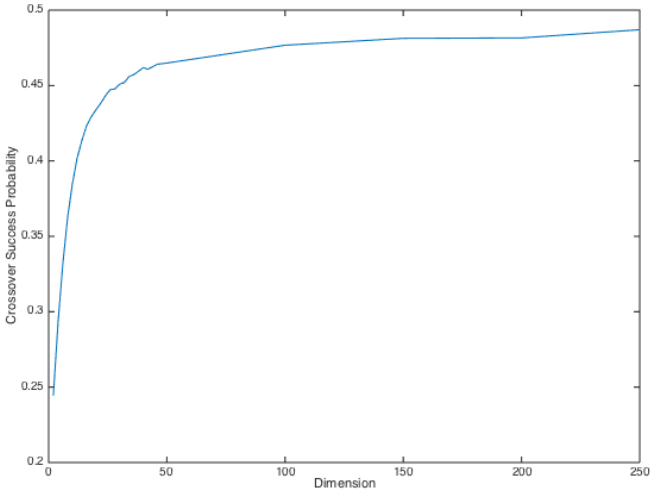
It can be seen in Figure 3.3 that when Cr=0.5 the variance of the crossover success probability is very small and remains close to 0.5 for all dimensions, it is interesting to see that in this case the success probability peaks for $d = 7$ which suggests that the probability is partially a function of the volume. However for Cr=025 and Cr=0.75 it can be seen that the functions tend to being more monotonic. For Cr=025 and Cr=0.75 the crossover success probability is more sensitive with respect to dimensionality, but as dimensionality increases for all values of the crossover rate the crossover success probability tends towards what is seen to be an asymptotic limit of 0.5.

The asymptotic limit can be explained that as the dimensionality increases the majority of the mass of the n-ball moves towards the surface and lies at the radius $O(\frac{1}{d})$, therefore the range of radii becomes more condensed. As the two points become closer to the surface the limiting behaviour becomes the same as seen in Theorem 3.1.

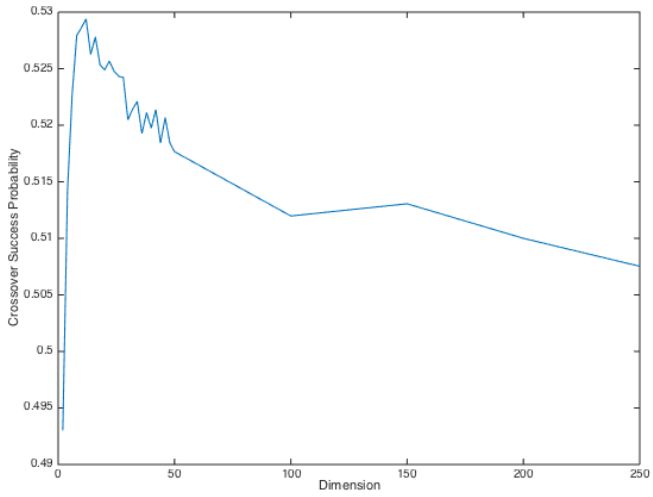
3.3.5 Crossover Success Rate Estimations

The results from the two preceding discussions regarding the scenarios for when $F(\mathbf{p}_2) < F(\mathbf{p}_1) < F(\mathbf{p}_R)$ and $F(\mathbf{p}_2) < F(\mathbf{p}_R) < F(\mathbf{p}_1)$ can now be combined to give estimations for the overall probability of successful crossover given three particles $\mathbf{p}_2 < \mathbf{p}_R, \mathbf{p}_1$, this corresponds to using gBest crossover.

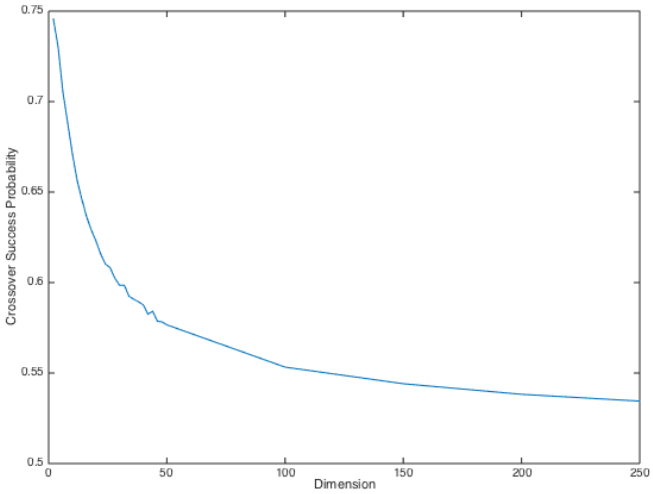
Theorem 3.4. *Assuming $\mathbf{p}_i$ are uniform i.i.d the probability of successful crossover*



(a) Cr $= 0.25$



(b) Cr $= 0.5$



(c) Cr $= 0.75$

Figure 3.3: Crossover success probability for $F(\mathbf{p}_1) > F(\mathbf{p}_R) > F(\mathbf{p}_2)$ with three different crossover rates.

of a random particle with the gBest has the asymptotic behaviour:

$$\lim_{n\to\infty} Pr[F(\mathbf{p}_c) < F(\mathbf{p}_R)] \to 0.5 \quad \forall Cr \in [0,1]. \tag{3.28}$$

Proof. This can be seen from the limiting case of crossover success probability in both scenarios, $F(\mathbf{p}_2) < F(\mathbf{p}_1) < F(\mathbf{p}_2)$ and $F(\mathbf{p}_2) < F(\mathbf{p}_R) < F(\mathbf{p}_1)$, tending towards the limit of 0.5 as the concentration of mass of the hypersphere moves towards the surface. ■

When the tournament procedure is used for picking the first parent the probability of the the first parent being better than the replacement particle becomes

$$\Pr[F(\mathbf{p}_1) < F(\mathbf{p}_R)] = \frac{2}{3}. \tag{3.29}$$

and

$$\begin{aligned}\Pr[F(\mathbf{p}_c) < F(\mathbf{p}_R)] = \frac{2}{3}\Pr[F(\mathbf{p}_c) < F(\mathbf{p}_R) \,|F(\mathbf{p}_2) < F(\mathbf{p}_1) < F(\mathbf{p}_R)] \\ +\frac{1}{3}\Pr[F(\mathbf{p}_c) < F(\mathbf{p}_R) \,|F(\mathbf{p}_2) < F(\mathbf{p}_R) < F(\mathbf{p}_1)]\end{aligned} \tag{3.30}$$

This therefore means that the tournament procedure increases the probability of successful crossover by increasing the probability that $F(\mathbf{p}_2) < F(\mathbf{p}_1) < F(\mathbf{p}_R)$. A lower bound of crossover success using the tournament procedure considering only the first scenario of parents is

$$Pr[F(\mathbf{p}_c) < F(\mathbf{p}_R)] > 0.33 \quad \forall \mathrm{Cr} \in [0,1], \tag{3.31}$$

compared to a bound of > 0.25 without a tournament procedure. For high dimensional problems, as previously seen, the crossover success probability tends towards an asymptotic limit of 0.5, regardless of the crossover rate. For the case of Cr=0.5 it can be seen from previous results that

$$Pr[F(\mathbf{p}_c) < F(\mathbf{p}_R)] > 0.5 \quad ; \mathrm{Cr} = 0.5. \tag{3.32}$$

Using a crossover rate of Cr=0.5 with a tournament procedure offers the best combination of child diversity and at least a 50% chance of being successful. Higher crossover rates will increase the probability of a successful child, but the diversity of the solutions is reduced and becomes more bias towards the best parent, in this case the global best, this can lead to a cluster of particles around the global best being formed.

3.3.5.1 Global Cluster

When using a gBest based discrete crossover a cluster forms around the global best location due to the probability that some child particles will take on the gBest particle position with probability

$$\Pr[\mathbf{p}_c = \mathbf{p}_{\text{gBest}}] = \text{Cr}^n, \tag{3.33}$$

where crossover is defined such that $\lim_{\text{Cr}\to 0} \mathbf{p}_c \to \mathbf{p}_1$ and $\lim_{\text{Cr}\to 1} \mathbf{p}_c \to \mathbf{p}_{\text{gBest}}$ with probability 1. For each iteration of the algorithm the expected size of the global best cluster, N_{gbc}, is

$$\mathbb{E}[N_{\text{gbc}}] = N\mathbb{P}_{\text{Br}}\text{Cr}^n \tag{3.34}$$

where N is the total population of the swarm, and $\mathbb{P}_{\text{Br}}$ is the probability that breeding occurs (breeding probability).

Due to the velocity component of the particle the particle then is subject to random perturbation around the global best location. This also highlights the importance of having the breeding operation occurring before the velocity update, otherwise there would be no perturbation to the particle position and the particle would remain at the global best location, which would significantly impact swarm diversity and the exploration capabilities. However, given that the particle, i, moves to the global best location the velocity update applied to the particle reduces to

$$\mathbf{v}_i(t) = w\mathbf{v}_i(t-1), \tag{3.35}$$

where w is either the inertia weight or constriction coefficient. If it is assumed that the velocity of each particle over time is a decreasing function such that $\mathbf{v}_i(t-1) \rightarrow 0$, then it can be seen that all particles will eventually end up at the global best location. In fact, it can be shown that when using gBest crossover the algorithm is globally convergent, given enough iterations, regardless of particle movement. This can be shown be observing that after a certain number of iterations all particles would have move to the global best cluster; once within the global best cluster the velocity update equation becomes

$$\mathbf{v}_i(t) = w\mathbf{v}_i(t-1) + C_2\mathbf{r}_2\left(\mathbf{p} - \mathbf{p}_{\text{gBest}}\right) \tag{3.36}$$

given that the personal best location will be the particles current location. It can then been seen how the particle will move towards the global best location and that $\mathbf{v}_i(t-1) \rightarrow 0$ given that $|w| < 1$. It can now also be seen why adding mutation is important, even as $\mathbf{v}_i(t-1) \rightarrow 0$, the mutation ensures that some exploration will persist and preventing the swarm from stagnating.

3.3.5.2 Impact on Choice of Cr Value

From the previous analysis it has been seen that the value of Cr can impact the PSO search dynamics, although Engelbrecht [141] provides an empirically sensitivity analysis for the crossover parameters used in PSO with discrete crossover (PSO-DX) hybrids, there lacks a discussion as to why different PSO-DX have a preference for different Cr values.

It can now be seen as to why in the cases where gBest crossover is used there is a preference for moderate/low Cr (0.2, 0.4). From the analysis presented it can be seen that given too a large Cr a cluster will form around the global best location and can lead to premature convergence, limiting the exploration abilities of the PSO. A high Cr in these cases can also lead to a higher chance of falling into local minima, if the incorrect minima is initially located and the global best cluster begins to form around this point. The choice of Cr is also impacted by the dimensionality of the problem and the size of the population. For a low dimensional problem and

small population size there is a higher probability that a larger percentage of the population will move to the global cluster. This can have a negative impact on the search capabilities of the algorithm and therefore a lower choice of Cr may instead be preferable. Based on the findings of this analysis Cr=0.5 offers the best compromise for retaining population diversity and maximising the chance of successful crossover.

Other PSO-DX using personal best crossover have a preference for high Cr (0.6, 0.8). This is due to the fact that crossover has a smaller effect on global convergence and acts only to improve the particle position within a localised area around the particle. A higher Cr in this case encourages more search around a known better area, and also increases the probability of crossover success in this localised area based on the finding in Section 3.3.5 where the personal best is used instead of the global best. A higher crossover rate will skew the child particle towards forming around the personal best, however this is a known area of better fitness and as such should have a high probability of creating a successful child. In this case it not possible for a global cluster to form, as all the parents are different for each particle. One alternative may be to use a local best crossover, this has the potential of combing the best of both elements seen for global best crossover and personal best crossover. Local best crossover would be able to maintain population diversity whilst creating small local clusters to more throughly explore promising areas.

3.3.5.3 Mutation Bias

Although mutation is useful for retain population diversity, one issue that can occur due to the choice of mutation parameters is a bias towards the axes. When the mutation function using Equation 3.7 is applied to the particles' position it can be moved anywhere within the linear range

$$p_c^d\,(1-M_{\mathrm{F}}) \leq M_L\left(p_c^d\right) \leq p_c^d\,(1+M_{\mathrm{F}}) \tag{3.37}$$

considering $M_{\mathrm{F}} < 1$ it can be seen that for all $p_c^d > 0$ the range is bounded $[0, p_c^d]$, given p_c^d is uniformly distributed around this axis it can be seen that as $M_{\mathrm{F}} \geq 1$ a

bias forms around the axis. For $M_F = 1$ the bias is formed at 0, this can be shown by observing that for a set of N points $p_i^d \in \mathbb{R}$ and define the set $E_i = [p_i^d(1 - M_F), p_i^d(1 + M_F)] = [0, 2p_c^d]$, the intersection of all sets E_i is

$$\bigcap_{i=1..N} E_i = 0. \tag{3.38}$$

More generally for any $M_F \geq 1$ the bias region, B, of the search space can be defined as

$$B = \bigcap_{i=1..N} E_i. \tag{3.39}$$

This is also true for any type of random mutation where the range is unconditionally extended over the axis. For functions where the optimum is centered somewhere around the origin this bias can be beneficial to the search progress of the algorithm, with a higher probability that the global minimum will be encountered. Whilst if a local minimum is located at the origin or close to an axis this bias results in a higher probability of finding and converging towards a sub-optimal solution. To help resolve this bias it is suggested that in Equation 3.7 the condition $M_F < 1$ is applied, although this implies a separate issue that particle remains bounded to their quadrant of its position and limits the particle from exploring regions across the axis. A better resolution may be to define that the variance of the mutation is dependant on the overall swarm state as a means of keeping the scale relative to the search. For example the swarm variance of positions or other diversity measures.

3.3.6 Empirical Analysis of Mutation Parameters

Following the introduction of BrPSO with mutation, BrPSO(M_P, M_F), the algorithm is initially tested on the eight basic DeJong benchmark functions [157] [158]. The effect of different mutation parameter pairs, mutation probability and mutation factor, are shown in Figure 3.4 that visualises the mean value of the minima found for a range of parameter pairs over the eight test functions considered. From Figure 3.4 it is possible to get a visual representation of how the mutation parameters can

effect BrPSO(M_P, M_F)s performance and identify regions of favourable behaviour. When mutation is added the algorithm's performance is significantly improved and in the best cases observed for each of the functions BrPSO(M_P, M_F) can produce competitive optimisation results on the DeJong benchmark functions.

It can be observed that BrPSO(M_P, M_F) with very small/negligible values of the mutation parameters, (the top left region of each heat-map), shows inherently very poor performance on the benchmark functions used here, although interestingly later on in this work, in Section 3.5 and Chapter 4, it is found that BrPSO(0,0) actually works very well in practical applications. This observation therefore throws some questions about the nature and applicability of using artificial benchmark functions as a valid method for measuring an algorithms respective performance.

From all of the test functions used here, the mutation factor of ≈ 0.5 seems to be the most robust, when combined with a suitable mutation probability. For the majority of functions,$f_{dj}1, f_{dj}3, f_{dj}4, f_{dj}5$ and $f_{dj}6$ the best combination is $M_F = 0.5, M_P = 1$, and based on the previous analytical analysis this corresponds to crossover dominant behaviour with extremely fast rates of global convergence, this indicates the minima is quickly found and then isolated. On the other hand for $f_{dj}7$ a more moderate approach seems to give the best performance where crossover is only occurring occasionally, based on this it can be speculated than when crossover occurs too often the rate of local convergence is too high and disrupts the global search.

The effects of mutation bias, Section 3.3.5.3, are also visible, with the combination of high mutation factor and high mutation probability finding the absolute minimum at $\hat{\mathbf{x}} = \mathbf{0}$ the majority of times, although it would be expected that a high mutation factor and probability would cause too much exploration, probabilistically it increases the chances of finding the minimum at $\mathbf{0}$, which is why this behaviour is not observed for $f_{dj}2$ and $f_{dj}8$.

All the PSO algorithms used by Liang *et al* [32] and including BrPSO(M_P, M_F) seemed to have trouble with the Rosenbrock $(f_{dj}2)$ function, and all but CLPSO had problems with the Schwefel $(f_{dj}8)$ function. The Schwefel function [159] is

characterised by the large separation between the deep local and global minimum and Rosenbrock function has a large shallow valley that can lead to stagnation of the population. The poor performance for BrPSO$(M_{\mathrm{P}}, M_{\mathrm{F}})$ for both the Rosenbrock ($f_{\mathrm{dj}}2$) and Schwefel ($f_{\mathrm{dj}}8$) can be explained by the limited behaviour of the mutation function used, Equation 3.7, when the mutation parameters are fixed. This is due to the scaling factor used in the linear mutation function that controls the mutation's variance.

In Equation 3.7 the scaling factor is a combination of the particles positions and the mutation factor, $p_c^d M_{\mathrm{F}}$. In most cases for the DeJong test functions the scaling used is appropriate for functions where the global optima, $\hat{\mathbf{x}}$, is located at $\|\hat{\mathbf{x}}\|_\infty < \mathbf{1}$; as particles move towards the optima the scaling factor reduces proportionally with the fitness, reducing the variance of the mutation to a similar magnitude as the positions and allowing for effective exploration around this smaller area. In cases where $\|\hat{\mathbf{x}}\|_\infty > \mathbf{1}$, initially the large scaled variance is good for exploration, however exploitation is not effective as the value of scaled variance stays respectively large around the global optimum hindering convergence. The assumption that the position and M_{F} produce a representative scaling factor is only true if the optimum lies within $\|\hat{\mathbf{x}}\|_\infty < \mathbf{1}$. For the Schwefel function, the optimum lies at $\hat{\mathbf{x}} = \mathbf{420.96}$, to allow for convergence the mutation must occur in the contracting area around this point, but under the assumption that the position and M_{F} is representative of the scale means that in this instance the variance of the mutation does not contract and remains large around this point. This can further be illustrated in the results of both the Rosenbrock ($f_{\mathrm{dj}}2$) and Schwefel ($f_{\mathrm{dj}}8$) functions; although non of the results could be considered significantly good the worst means are seen for the large mutation factor and probability values, whilst the best values are seen for the smaller mutation factors which allow for a smaller mutation search area relative to the scaling by the position.

In the case that M_{F} is static there is a tradeoff, a small M_{F} during the initial search stages means that mutation provides very little benefit in terms of the size of the relative search area it can extend, whilst a large M_{F} limits exploitation of good

solutions and convergence. Therefore it can been see that M_{F} needs to have some sort of dynamic behaviour that allows the mutation variance to be scaled and contract appropriately during the stages of convergence and expand during exploration.

For all cases though compared to BrPSO($M_{\mathrm{P}}, M_{\mathrm{F}}$) with a very small mutation factor and mutation probability adding a moderate amount of appropriately scaled mutation to crossover produces beneficial results. From the range of mutation parameters tested in Figure 3.4 it can be seen that performance of BrPSO($M_{\mathrm{P}}, M_{\mathrm{F}}$) is sensitive to the combination of mutation parameters used. In general over the eight functions tested the bottom right quartile is preferable, this corresponds to parameterisations with high mutation probability and mutation factor, although using static mutation parameters leads to tradeoffs between the exploration and exploitation stages. Overall BrPSO shows potential to be a powerful optimiser but it is still subject to the no free lunch theorem with respect to mutation parameterisation.

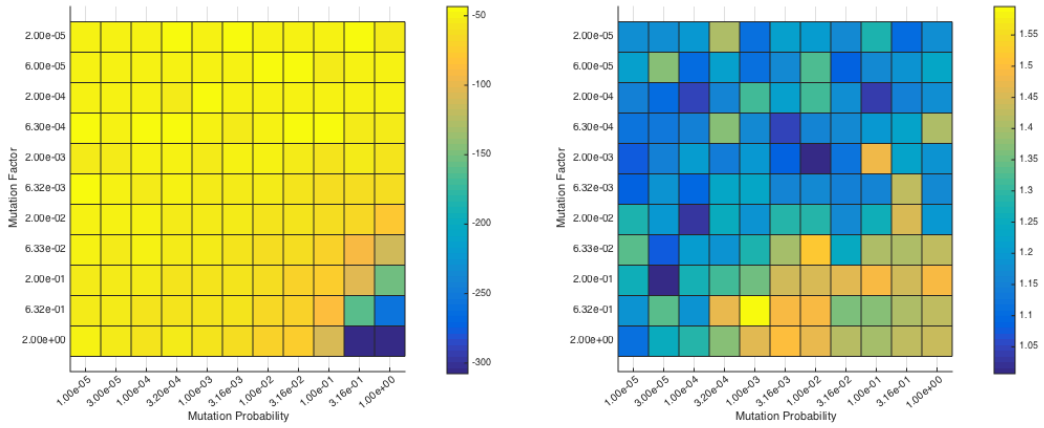
3.3.7 Conclusion

A discrete global best crossover operator has been embedded into the PSO algorithm, BrPSO, compared to previous approaches a tournament procedure and perturbations in the form of a mutation function are introduced.

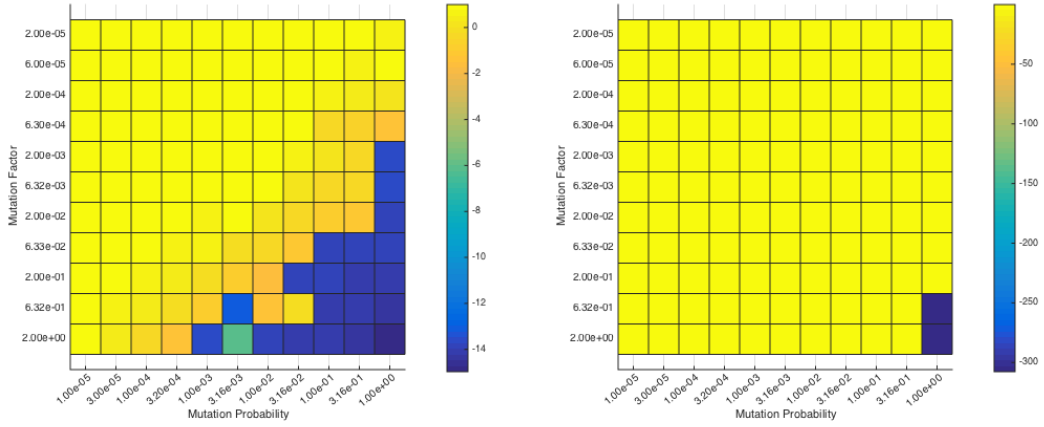
Analytical analysis of the discrete crossover operator and torment procedure show that this mechanism increases the probability of improving a particle's position. The discrete crossover operator also effects the swarm dynamics by introducing the behaviour of global clustering, where a subset of the population moves very close around the global best position. The crossover rate has a significant effect on this behaviour and in combination with the velocity inertia influences the swarms overall rate of convergence.

Though, the analysis introduced can be further improved as currently the bounds of the behaviour introduced are extremely loose when compared to empirical simulations. More accurate bounds need to take into consideration a wider range of crossover behaviour and the success rate probability in regions where different numbers of elements of the two parent vectors of exchanged.

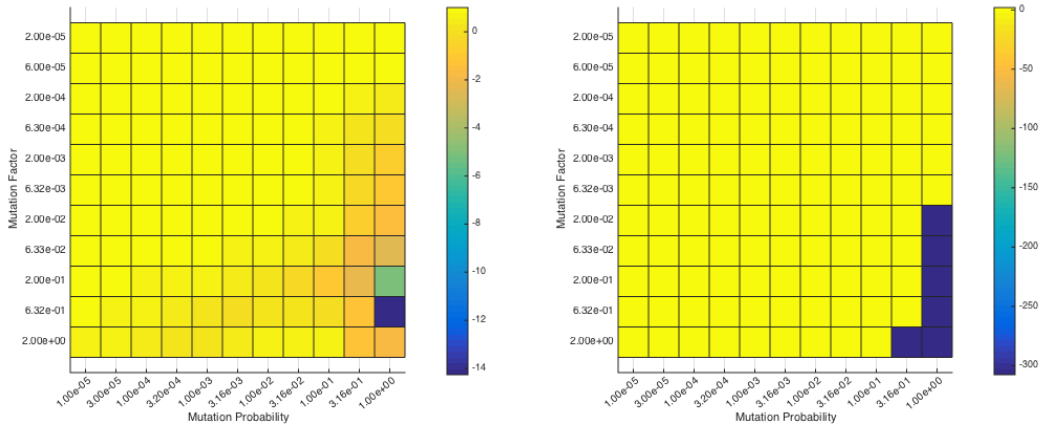
Further work should also aim to analyse how different swarm topologies effect



(a) $f_{\mathrm{dj}}1$ - Sphere **(b)** $f_{\mathrm{dj}}2$ - Rosenbrock

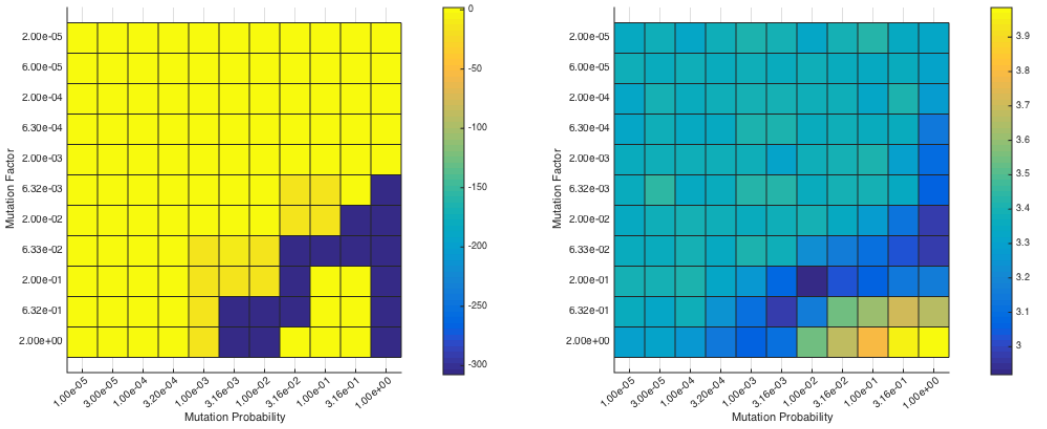


(c) $f_{\mathrm{dj}}3$ - Ackley **(d)** $f_{\mathrm{dj}}4$ - Griewank



(e) $f_{\mathrm{dj}}5$ - Rastrigin **(f)** $f_{\mathrm{dj}}6$ - Rastrigin Integer

Figure 3.4: Heatmap plot for BrPSO($M_{\mathrm{P}}, M_{\mathrm{F}}$) mutation parameter sets on the 8 DeJong functions, showing the logarithm of the mean value of the fitness found (50 runs).



(g) $f_{\mathrm{dj}}7$ - Rosenbrock **(h)** $f_{\mathrm{dj}}8$ - Schwefel

Figure 3.4: cont. Heatmap plot for BrPSO($M_{\mathrm{P}}, M_{\mathrm{F}}$) mutation parameter sets on the 8 DeJong functions, showing the logarithm of the mean value of the fitness found (50 runs).

the behaviours of crossover and global clustering, for example if using a local best topology the probability of a local cluster and the size of the cluster forming around a point and how this effected by the connectivity. One topology which may work better is overlapping fully connected subswarms, this allows the subswarms to focus on the local minima and locally converge, but with the overlapping particles given the ability of contour hopping by the crossover.

The initial version of BrPSO with mutation has been tested on the set of DeJong benchmark functions, it was found that the addition of mutation significantly increased the performance of the BrPSO algorithm and produced a competitive optimisation algorithm. However, it was also seen that the fixed mutation parameters used introduce issue with search bias and a tradeoff between exploration and exploitation behaviour, as such it is proposed that a dynamic set of mutation parameters can produce a most robust optimisation algorithm.

3.4 Self-Adaptive Mutation

The search ability of BrPSO(M_P, M_F) was sensitive to the values of the mutation probability and mutation factor. The manner in which performance (the minima found) was affected by the mutation probability and probability factor varied with uni-modal and multi-modal problems, as well as with the dimensions of the search space. This supports the 'no free lunch theorem' which states that there is no 'one size fits all' algorithm, and that for optimal performance tuning may be necessary. To overcome this self-adapting mechanism for the parameters is introduced into BrPSO(M_P, M_F), dubbed BrPSO-SAM, which allows the algorithm to learn the best values for the mutation probability and mutation factor, removing the issue of parameter tuning and maximising the algorithm's performance.

3.4.1 Self-Adaptive PSO

Self-adaptive methods in PSO are commonly used for adapting the three velocity parameters, a common method is linearly decreasing or increasing the parameter values throughout the search duration, such as in the linearly decreeing inertia in PSO-TVIW, or the linearly changing C_1 and C_2 values in HPSO-TVAC [91] and LPSO-TVAC [160]. This method assumes the swarm is following a linear progression of changing state, and works by aiming to increase the swarm's exploitation characteristic near the end of the optimisation, the risk of this assumption is that the swarm can perform an efficient initial search locating the global basin of attraction otherwise it may bury itself deep into local minima as exploitation increases. As such it may be desirable to allow the swarm to change its characteristics more dynamically.

A popular dynamic approach is to monitor the swarm's state and adapt parameters accordingly. A simple method by Arumugam *et al* [161] adapts the three parameters as functions of the ratio of the average personal best fitness and the global best fitness. This ratio can serve as an estimate to the degree of diversity. Using a slightly modified velocity update (the two acceleration components share the same random number and C value), the inertia is decreased and the acceleration coeffi-

cient is increased to accelerate convergence as the swarms diversity decreases. The most recognised approach is APSO [21], which, as previously discussed in Section 2.1.1, uses a measure of the swarm's state to dictate the acceleration parameter settings using a fuzzy classification method; this allows the swarm to switch between explorative and exploitative behaviours throughout the optimisation process.

A different approach which does not require monitoring the swarm's state given by Kannan *et al* [137] introduces an adaptive method for all three velocity parameters using a composite method involving DE as an internal search mechanism. Montalvo *et al* [162] uses a similar method with PSO being applied to the three parameters by extending the existing particles' search space to include three additional dimensions for the parameter space. This approach is again later adopted by Ismail *et al* [22], where two swarms are used, one for the optimisation problem and the other for searching the three velocity parameter space. The parameter values for each particle in swarm one are selected using the weighted probability selection method used by Wang *et al* [163], this allows the same particle to sample different types of parameter settings compared to the integrated optimisation approaches used by Kannan *et al* [137] and Montalvo *et al* [162] but has the disadvantage of requiring more overall fitness evaluations.

Although it is not within the scope of this chapter, additional work has been presented in the appendix which further investigates the concept of self-adapting PSO algorithms with respect to the original velocity parameters. In this work an approximation to the optimal adaptation process is presented as Greedy Self-Adapting PSO.

The previously discussed methods focus on adapting the parameters of the standard PSO velocity update equation. In heterogeneous PSO (HTPSO) model [164] particles are allowed to have different individual behaviours randomly chosen from a pool, an adaptive HTPSO has been introduced by Wang *et al* [163] which allows for adaptation of the velocity strategies themselves, although parameters for the strategies remain fixed. This is based on the observation that different PSO variants have different strengths and weaknesses for different problem types. They

use the four strategies CLPSO, PSO-CL-pbest, DbV, and EbV. Each strategy has an operating probability which is updated via a fitness based scoring system which is evaluated at the end of each iteration. As an alternative method Nepomuceno *et al* [165] use a hybrid with Ant Colony Optimisation (ACO) to select strategies from a pool a pool of 7 strategies: Cog-PSO; Soc-PSO; BB-PSO; BBMod-PSO; QSO; TVIW-PSO; and TVAC-PSO.

3.4.2 BrPSO with Self-Adaptive Mutation

The approach for self-adaptation used here is most similar to the extension approach of Kannan *et al* [137], which is also used in DEGL-SA [75]. The search space of PSO is extended by two dimensions which correspond to the BrPSO mutation parameters M_{P} and M_{F}.

Compared to previous approaches which use an extension of the search space it is considered that the parameter search space is a dynamic search as the optimum parameter values change as the swarm moves and it's behaviour evolves throughout the main PSO search. The dynamic changing search landscape of the parameter space as the main optimisation in the $\mathbf{x}$ space proceeds means that it is not desirable to have the mutation parameters converging towards one set of values. Similar to other dynamic PSO approaches [166] [167] it is important to retain population diversity and to stop the swarm converging to a stationary point but also to retain some tracking of the best current solution.

As such a PSO-augmented random search with a slightly modified velocity update equation is designed for the mutation parameters. In this, the particles are, as usual, initialised to random positions in the parameter space. The change is in the PSO velocity update, where instead of using the particle's personal best, the random initial position $\mathbf{x}(0)_i$ is used,

$$\mathbf{v}_i(t) = \mathbf{v}_i(t-1) + C_1\mathbf{r_1} \odot (\mathbf{x}_i - \mathbf{x}(0)_i) + C_2\mathbf{r_2} \odot (\mathbf{x}_i - \hat{\mathbf{y}}_i). \tag{3.40}$$

This allows the parameter search to move in the direction of the global/local best, but due to the random element of the initial position this prevents convergence, re-

taining the desired diversity, this is similar to the idea of jitter applied in DE. It should be emphasised that this variant of the PSO velocity is used for the optimisation of the mutation parameters only.

Algorithm 3.4 Pseudo code for BrPSO-SAM

```
Initialise swarm as array of particles, set each particle with a random breeding probability (P_Br^particle)
swarm = new ParticleArray[N]
particleParameters = new ParameterArray[N]
while not stopping citera do
    for every particle, particle do
        if U[0,1] < P_Br^particle then
            Select two random particles, p1, p2
            Compare the two random particles and select the particle with the best personal fitness
            for each dimension, d do
                if U[0,1] < P_CO then
                    trial[d] = swarm[globalBest].bestPos[d]
                else
                    trial[d] = swarm[selectedBest].bestPos[d]
                end if
                if U[0,1] < P_Mut then
                    trial[d] = M(trial[d])
                end if
            end for
            if fitness(trial) < fitness(particle) then
                particle.pos = trial
                Update particle personal bests and swarm global best indexes
            end if
        end if
    end for
    for each Particle particle do
        Update particle velocity and position
        Update particle parameter velocity and position (using the swarm global best particle values)
        Evaluate particle fitness, update personal best and global bests if applicable
    end for
end while
```

3.4.3 Benchmark Performance

The newly introduced self-adapting mutation variant of BrPSO(M^P, M^F), BrPSO-SAM, is tested on the set of CEC'05 benchmarking functions [168]. This set of 25 benchmark functions provides a more robust and challenging environment than the original 8 DeJong functions and is widely used in the literature for comparison of EA algorithms [169]. The fundamental functions are the same as those used in the DeJong test set but are elaborated on by applying shift, rotation and composite operations. Shifting and rotation are key for capturing algorithms that would otherwise rely on search biases around the axis and towards the origin, these benchmark functions provide a good test to see if the mutation bias issues initially present in BrPSO(M^P, M^F)is resolved by using self-adapting mutation. The CEC'05 functions can be divided into three main sub groups of increasing landscape complexity: unimodal functions $f_{'05}1 - f_{'05}5$; basic multimodal functions $f_{'05}6 - f_{'05}14$; and composite functions $f_{'05}15 - f_{'05}25$.

BrPSO-SAM has been run using the standard CEC'05 experimental setup,

$D = 30$, max fitness evaluations = 300,000, number of particles = 40 and with 30 independent runs for each test function. The results of the runs are shown in Figure 3.5, the results of BrPSO-SAM obtained here are then compared to results from the literature for other state-of-the-art PSO algorithms [90], APSO, OPSO and CLPSO in Table 3.1, and compared against other PSO-Crossover hybrids [154] in Table 3.2.

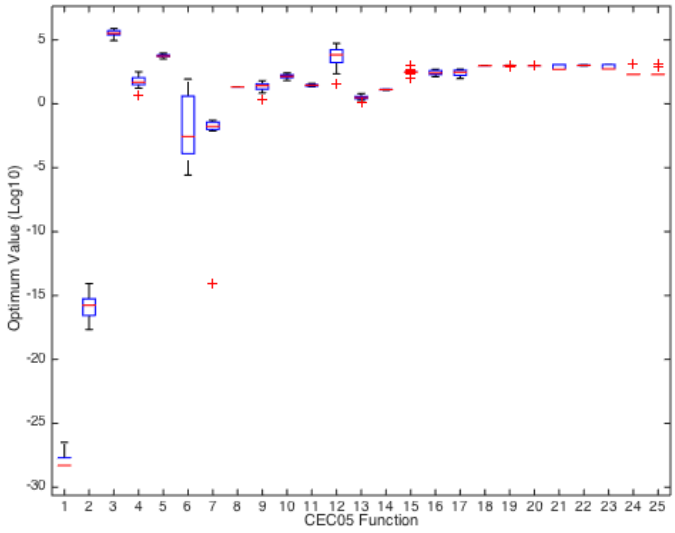


Figure 3.5: Box plot of 30 independent optimisation runs using BrPSO-SAM on the CEC'05 functions.

Compared against APSO, OLPSO and CLPSO, BrPSO-SAM ranks well for the first 14 functions, and has comparable performance for the composite functions. Most of the results show very similar performance across all the algorithms. It can be seen from the mean ranks that in particular APSO,CLPSO and BrPSO-SAM are very similar, but with OLPSO ranking the overall best. The most significant results for OLPSO are for $f_{05}1$ and $f_{05}9$, although BrPSO-SAM is very close for $f_{05}1$, the results on $f_{05}9$ shows that the OLPSO has a significant advantage for this search landscape. Overall the PSO algorithms all have very similar performance for the majority of the test functions considered here and BrPSO-SAM can be considered a competitive PSO algorithm.

When compared against other PSO-DX algorithms, BrPSO-SAM ranks overall the best for the first 14 functions, although again for the majority of functions the

	APSO, OPSO,CLPSO			BrPSO-SAM		
	Best	Mean	Std	Mean	Std	Rank
$f_{05}1$	OLPSO	0.00e+00	0.00e+00	1.91e-28	5.83e-28	2
$f_{05}2$	APSO	9.97e-13	1.79e-12	8.63e-16	2.08e-15	1
$f_{05}3$	APSO	3.96e+05	1.59e+05	3.82e+05	1.83e+05	1
$f_{05}4$	APSO	7.23e+01	6.02e+01	9.19e+01	9.37e+01	2
$f_{05}5$	OLPSO	3.28e+03	5.54e+02	5.79e+03	1.38e+03	3
$f_{05}6$	CLPSO	5.10e+00	5.43e+00	4.29e+00	1.57e+01	1
$f_{05}7$	APSO	4.70e+03	2.34e-04	2.16e-02	1.47e-02	1
$f_{05}8$	APSO	2.00e+01	2.97e-02	2.06e+01	4.89e-01	2
$f_{05}9$	OLPSO	0.00e+00	0.00e+00	2.74e+01	1.57e+01	4
$f_{05}10$	OLPSO	1.10e+02	3.12e+01	1.48e+02	4.83e+01	3
$f_{05}11$	OLPSO	2.55e+01	2.95e+00	2.87e+01	4.16e+00	4
$f_{05}12$	APSO	1.27e+04	1.70e+04	1.01e+04	1.15e+04	1
$f_{05}13$	APSO	1.54e+00	4.05e-01	3.27e+00	1.09e+00	4
$f_{05}14$	CLPSO	1.29e+01	1.72e-01	1.29e+01	6.24e-01	2
$f_{05}15$	CLPSO	1.06e+02	5.34e+01	3.34e+02	1.33e+02	3
$f_{05}16$	OLPSO	1.32e+02	3.74e+01	3.10e+02	1.31e+02	3
$f_{05}17$	OLPSO	1.89e+02	3.25e+01	2.93e+02	1.36e+02	3
$f_{05}18$	OLPSO	9.10e+02	1.82e+00	9.55e+02	2.32e+01	4
$f_{05}19$	OLPSO	9.07e+02	2.03e+01	9.54e+02	4.27e+01	4
$f_{05}20$	OLPSO	9.07e+02	2.03e+01	9.51e+02	2.93e+01	4
$f_{05}21$	OLPSO	5.00e+02	2.86e-13	7.68e+02	3.23e+02	4
$f_{05}22$	OLPSO	9.43e+02	1.35e+01	1.02e+03	4.97e+01	4
$f_{05}23$	OLPSO	5.34e+02	3.59e-04	8.08e+02	3.22e+02	3
$f_{05}24$	OLPSO	2.00e+02	2.89e-14	2.68e+02	2.59e+02	3
$f_{05}25$	OLPSO	1.64e+03	5.59e+00	3.92e+02	3.97e+02	1

Mean Rank			
APSO	OPSO	CLPSO	BrPSO-SAM
2.92	1.84	2.28	2.68

Table 3.1: Comparing BrPSO-SAM on the CEC'05 benchmarking suite ($D = 30$) with state-of-the-art PSO algorithms tested by Li *et al* [90]

difference between the mean minima found by all the algorithms is often very small so as to be insignificant. The results of BrPSO-SAM show that using mutation could be beneficial for all PSO-CX algorithms. PSO-DX_y^1 shows significant results for $f_{05}4$ compared to all other algorithms, this is the shifted Schwefel function with noise.

BrPSO-SAM showed significantly good relative performance when compared to the other algorithms on $f_{05}2$, 7 and 25. These functions are the unimodal shifted Schwefel problem without bounds, the multimodal shifted-rotated Griewanks func-

	PSO-PX, DX_y^u, DX_y^1			BrPSO-SAM		
	Best	Mean	Std	Mean	Std	Rank
$f_{05}1$	PCX	1.00e-14	3.35e-14	1.91e-28	5.83e-28	1
$f_{05}2$	DX1Y	1.00e-13	3.67e-13	8.63e-16	2.08e-15	1
$f_{05}3$	DXUY	1.15e+03	1.03e+03	3.82e+05	1.83e+05	4
$f_{05}4$	DX1Y	0.00e+00	1.55e-07	9.19e+01	9.37e+01	3
$f_{05}5$	DX1Y	2.43e+04	8.26e+03	5.79e+03	1.38e+03	1
$f_{05}6$	DXUY	1.10e+01	1.40e+01	4.29e+00	1.57e+01	1
$f_{05}7$	DX1Y	9.59e+01	2.03e+00	2.16e-02	1.47e-02	1
$f_{05}8$	PCX	2.10e+01	7.85e-02	2.06e+01	4.89e-01	1
$f_{05}9$	DXUY	3.90e+01	1.27e+01	2.74e+01	1.57e+01	1
$f_{05}10$	DXUY	1.13e+02	3.60e+01	1.48e+02	4.83e+01	4
$f_{05}11$	DXUY	2.70e+01	3.09e+00	2.87e+01	4.16e+00	2
$f_{05}12$	DXUY	1.81e+04	1.76e+04	1.01e+04	1.15e+04	1
$f_{05}13$	DXUY	3.00e+00	1.16e+00	3.27e+00	1.09e+00	2
$f_{05}14$	DXUY	3.06e+02	8.00e-01	1.29e+01	6.24e-01	1
$f_{05}15$	DXUY	4.11e+02	2.63e+01	3.34e+02	1.33e+02	1
$f_{05}16$	PCX	4.20e+02	1.45e+02	3.10e+02	1.31e+02	1
$f_{05}17$	DX1Y	3.70e+01	8.10e+01	2.93e+02	1.36e+02	3
$f_{05}18$	PCX	3.52e+02	2.42e+02	9.55e+02	2.32e+01	4
$f_{05}19$	PCX	3.85e+02	1.79e+02	9.54e+02	4.27e+01	4
$f_{05}20$	DX1Y	2.29e+02	5.69e-03	9.51e+02	2.93e+01	4
$f_{05}21$	DXUY	1.73e+02	1.41e+02	7.68e+02	3.23e+02	4
$f_{05}22$	DX1Y	4.22e+02	4.39e+02	1.02e+03	4.97e+01	4
$f_{05}23$	DXUY	1.94e+02	7.17e+01	8.08e+02	3.22e+02	4
$f_{05}24$	PCX	5.30e+01	2.20e+01	2.68e+02	2.59e+02	4
$f_{05}25$	PCX	1.26e+02	2.95e+01	3.92e+02	3.97e+02	4

Table 3.2: Comparing BrPSO-SAM on the CEC'05 benchmarking suite ($D = 30$) with other PSO-Crossover hybrids Engelbrecht *et al* [154]

tion without bounds and the rotated hybrid composition function without bounds respectively. The most noticeable of these results is for $f_{05}7$ where the average minima found by BrPSO-SAM is three magnitudes smaller than the best other PSO. In the two functions where BrPSO-SAM outperforms the other PSO algorithms without crossover (APSO, OPSO and CLPSO) both functions, $f_{05}7$ and $f_{05}25$ have the property that the global optima is located outside of the initialisation bounds. The positive results on these two functions support the idea that the additional breeding and mutation introduced in BrPSO aids additional exploration for the PSO algorithm and allows the algorithm to more widely explore the search space.

Good performance is also seen for $f_{05}1$ and $f_{05}6$. Compared to all the other al-

gorithms except OLPSO, BrPSO-SAM shows a significantly better minimum found for $f_{'05}1$, which is the unimodal shifted sphere function. Being a unimodal function means that the rapid convergent properties seen for gBest crossover can be exploited, although the absolute minimum was not found, which may suggest that the local convergence towards a point was too fast. This could suggest that a lower crossover rate or larger population may be beneficial. From Figure 3.5 it can be seen that for $f_{'05}6$, the shifted Rosenbrock function, there is a large range in values found but with a low median indicating a relatively good overall performance on this function. The large range in minima found can be explained by observations that although the Rosenbrock function is classed as a unimodal function there is evidence suggesting that in high dimensions it becomes multimodal [170]. The multimodal Rosenbrock function is characterised by a narrow valley separating the local and global optima. This is similar to the characteristic shown in the Deceptive function [86] which has been seen to be challenging for many EAs, particularly DE algorithms.

Relative to the other PSO algorithms BrPSO-SAM performed the worst on $f_{'05}9$, the shifted Rastrigin's function. This is otherwise known as the egg box function, which has a steep local optima, from the respectively high value of the optima found by BrPSO-SAM and from the small interquartile range over all of the runs shown in Figure 3.5 it can be inferred that BrPSO-SAM often got caught in this local minima. This is possibly due to the effects of global clustering encouraging global convergence to local minima. Furthermore, poor performance is also shown on this function for the other PSO-CX algorithms, which further supports the proposition that global clustering due to crossover hinders the algorithms search abilities by encouraging premature convergence.

When comparing the results for the composite functions, $f_{'05}15 - f_{'05}25$ show very little significance and there is only a very small variation observed. This is due to the inherent difficulty of these problems and consequently generally poor performance is seen across the board. From Figure 3.5 it can be seen that there is little variation between the magnitude of the optimum found for these functions which

shows that the mean values used for comparison are not majorly skewed by one or two bad runs. Two exceptions maybe for $f_{'05}25$ and $f_{'05}24$where BrPSO-SAM and PSO-PCX respectively show one magnitude better. For $f_{'05}25$ BrPSO-SAM and other PSO-CX algorithms show better performance compared to the other PSO algorithms without crossover. As previously mentioned crossover allows for an enhanced exploration which is advantageous for functions with unbounded domains.

Overall it can be seen that when compared to other PSO algorithms with and without crossover BrPSO-SAM shows competitive search performance compared to the other algorithms considered here. When comparing BrPSO-SAM to other PSO algorithms on the CEC'05 benchmark functions the performance is very much inline with the other PSO algorithms considered here. The results on the CEC'05 functions further confirm the validity of 'the no free lunch theorem' with certain functions highlighting advantages and disadvantages of BrPSO-SAM with respect to different search space landscapes. The main disadvantage seen is that it is susceptible to steep local optima, due to the effects of global clustering, this suggest further research is required into the effects of the crossover rate on multi-modal landscapes and optimising the crossover rate to reduce clustering when required.The main advantages seen for BrPSO-SAM is an increased exploratory ability due to crossover and mutation which allows it to search unbounded domains efficiently. This property supports its particular use in applications such as neural network training, where the initialisation area is generally a guess and the algorithm needs to be able search a large unbounded search space to potentially find the best set of network weights.

3.5 BrPSO for Function Approximation using Neural Networks

This section will describe how BrPSO can be used with neural networks to approximate complicated functions of many inputs; it also demonstrates the practical value of BrPSO for neural network training. As an example a challenging function approximation problem in naval architecture has been chosen. As well as demonstrating the practical use of BrPSO, BrPSO neural networks are also able to provide a new more accurate solution for the problem presented.

3.5.1 Introduction

There are a large number of new creative small ship designs especially involving multi-hulled constructions. One such such design is the the Trimaran Small Waterplane Area Centre Hull (Tri-SWACH). The Tri-SWACH hullform presents a hybrid between the trimaran and small-water-area-twin-hull (SWATH) designs. In SWATH catamaran designs a single hull consists of a narrow beam (strut) connected to a submerged hull (bulb), the advantages of SWATH compared to regular catamaran design is it superior stability and reduced wake, this has made it popular in the design of corvette class combat ships and experimental combat ships. The Tri-SWACH design uses a center hull with the same design as a SWATH hull, connected to two small side hulls, outriggers, as depicted in outline in figures 3.6 and 3.7. Initial research shows that this vessel has outstanding motion performance allowing for operations in very harsh weather conditions [171].

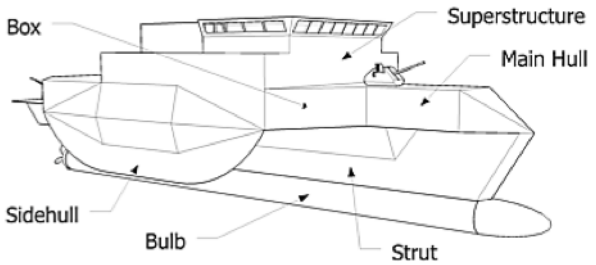


Figure 3.6: Tri-SWACH cross-section (figure from [3])

An important part of evaluating a ships performance is understanding the resis-

tance it makes when moving through the water. The Froude number, Fr, is a dimensionless coefficient used to determine flow on a external field acting on a body. In naval architecture the Froude number is a dimensionless form of velocity and is an important value used for characterising a ship's behaviour which is used to calculate the wave making resistance (the resistance created by the ship pushing through the surface of water) and the total resistance of ship [172]. The Froude number can then be used in Froude scaling which allows the measurements from small tow-tank prototype models to be scaled up whilst retaining the same hydrodynamic behaviour. Using tow-tank models it is possible to determine the dimensionless total resistance coefficient of the hullform to be used in scaling, in general it is assumed that the majority of the total resistance is caused by the wave making resistance, the hydrostatic resistance assumed to be negligible, therefore the Froude number should be able to provide a relation to the total resistance.

However due to the novelty of the Tri-SWACH design it presents challenges of its own for properly evaluating its characteristics. Tow-tanks experiments are expensive and timely to perform and consequently for a novel design there is a lack of experimental data. In addition, there is no analytical solution or reliable numerical method able to approximate water resistance for this hullform. This work uses the limited tow-tank data from experiments conducted at the Stevens and Webb Institutes and at the United States Naval Academy (USNA) as part of the ACCeSS program, which investigate the resistance characteristics of the Tri-SWACH design in calm water.

Under the circumstances of limited experimental data artificial neural networks (ANNs) and other bio-inspired metaheuristic techniques such as particle swarm optimization offer the possibility of being able to extract the underlying physics, to build a resistance model that can predict the dimensionless resistance coefficients that could be used to refine and scale up the Tri-SWACH hull design.

3.5.2 Neural Network Architectures and Training

Neural networks have previously been applied to Tri-SWACH and similar ship designs but with a limited degree of utility [173] [174]; [175].

In the experiments of [175] it was discovered better results could be obtained from larger nets than had been used in the earlier work of [174]; although as shown by Couser [173] too many neurons can result in overfitting and consequently poor predictive output. Couser [173] uses a neural network with a single hidden layer of 15 neurons. Given the interference behaviour of Tri-SWACH is more complex than for catamarans an additional 5 neurons are used. Thus a neural network with a single hidden layer of 20 neurons is used (3-20-1 net). Furthermore the use of a two hidden layer network is also investigated, even though Couser [173] found a second layer had little effect and often resulted in worse predictive output, the additional complexity of the dynamics of Tri-SWACH warrants a deeper architecture to try and fully capture the non-linearities. In addition using BrPSO as a powerful training algorithm means that it should be able to easily optimise the additional weights added by the extra layer. To keep the number of hidden neurons constant two hidden layers of 10 neurons are used (3-10-10-1 net). The hyperbolic tangent (tanh) function is used as the neuron activation function, functionally equivalent to the logistic sigmoid used in [174] [175], and all swarms used in the experiments below consisted of 100 particles.

The neural networks are trained using particle swarm optimisation, this work is also used to compare the training abilities of PSO-lBest and the newly developed BrPSO algorithm for what appears to be a challenging optimisation problem.

Different measures were used for training and for test performance assessment. It was discovered that training was most effective when the fitness function is the root mean squared error (RMSE) between the estimated and the target function values. However for the out-of-sample test data, the accuracy was measured in terms of the mean absolute error (MAE), this choice being made in order to better compare these results against previous work [175] in which the MAE was the quoted measure.

3.5.3 Data

In the ACCeSS towing tank experiments nine side hull configurations were considered, comprised of three possible lateral (inboard, mid, outboard) and three longitudinal (fwd, mid, aft) positions of the side hulls relative to the central hull. However not all consortium members took measurements at all positions. Among the ACCeSS data, that from the Webb Institute [176] is the only complete set of experimental data from a single source; it was decided to follow [175] in using these data alone for resistance predictions, since experimental setups necessarily differ in detail, and [175] gives evidence of a discrepancy between resistance measurements for the same hull configurations obtained by different institutes. The side hull locations considered are shown in outline in figure 3.7, and relevant measurements given in Table 3.3, in which %LS is a dimensionless measure of longitudinal side hull position (the longitudinal location from the stern divided by the length of the center hull), and %TS is a corresponding measure for the transverse side hull position, given as the transverse location from the center hull's center line divided by ten times the beam of the center hull (the factor of 10 being a scale factor used [174] [175] to bring %TS into line with the magnitude of %LS).

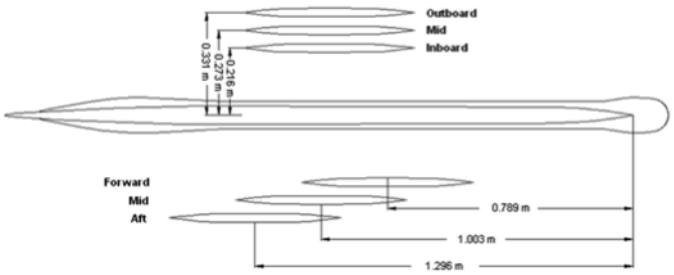


Figure 3.7: Tri-SWACH model side hull locations for towing tank tests (figure from [177])

Position	A fwd-outer	B fwd-mid	C fwd-inbd	D mid-outer	E mid-mid	F mid-inbd	G aft-outer	H aft-mid	I aft-inbd
%LS	37	37	37	47	47	47	57	57	57
%TS	10.1	12.8	15.5	10.1	12.8	15.5	10.1	12.8	15.5

Table 3.3: The nine side hull positions considered, together with dimensionless descriptors of the geometry (used here as ANN inputs). The 34 varying-speed measurements associated with column E will be the test data.

For each of these nine positions, resistance measurements were made at a range of speeds for 34 Froude numbers between 0.1 and 0.5, which given the decision to restrict the study to the Webb Institute data alone, results in a total of 306 available data points for training and testing. The task in this current work will be to use a multilayer ANN, trained with particle swarm optimization, to approximate the resistance functions R_T (Figure 3.8) and C_R (3.9) for side hull position E (the mid-mid position, selected also in [175] as the test data set). In contrast to [174] [175] predictions will here be based only on the minimal three inputs %LS, %TS, and Froude number (Fr) .

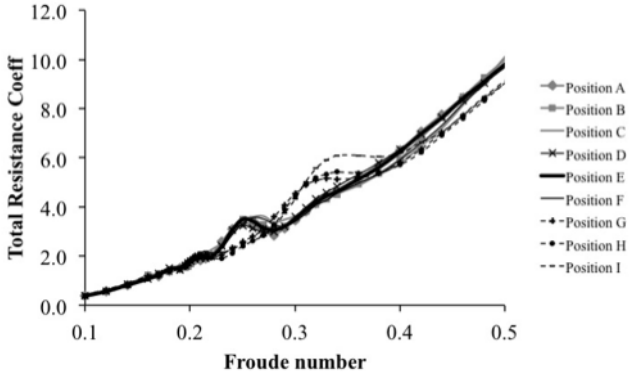


Figure 3.8: Total Resistance (R_T) as a function of Froude number (Fr) ; the curve in bold is the test data (mid-mid; position E), the shape of which was well predicted in [175] using Bayesian Regularization, but only with two additional (Reynolds number) network inputs.

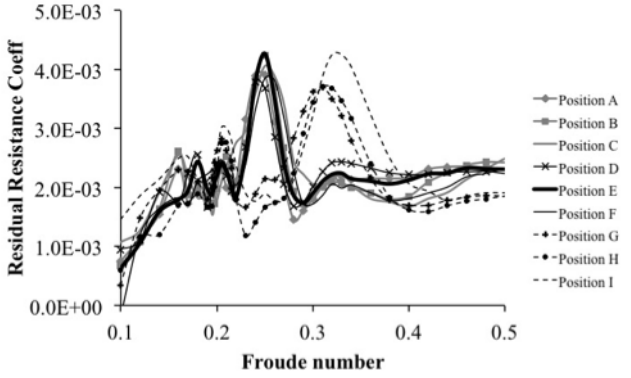


Figure 3.9: Residual Resistance Coefficient (C_R) as a function of Froude number (Fr) ; the curve in bold is the test data (mid-mid; position E), whose most significant features (the peak and side-lobes) could not be effectively predicted in [175] using any network architecture

Input scaling

To further aid training the network it often helps if all the inputs are of the same magnitude [178]; this helps to reduce the variance in the magnitude of the network weights. The definition of the %TS parameter included a division by 10 to make it of similar magnitude to %LS. It was decided here to divide both these inputs by an additional factor of 10 to make them of a similar magnitude to the Froude number input. Output scaling: none was done for the R_T prediction problem, but given the very small magnitude of C_R it was decided to multiply the targets by 100 during training to better separate the values.

Data Partitioning

When training an ANN it is usually advised that the data is split into three subsets, training, validation and test, an advised ratio is usually around 3:1:1. Given the size of the proposed network architectures it would be expected that a validation set would be needed to avoid overfitting to the data. However the training error was then unacceptably high: it appeared the reduced training set was insufficient to capture the complexity of the underlying function, and hence it was decided not to validate. In this case the decision was made to omit the validation set due to the scarcity of data and to maximise the size of the training set. It is also observed that this data should have very little noise which reduces the risk of overfitting to bad features. The Webb Institute data set contains 306 examples overall, with 272 available for training/validation after the mid-mid configuration data points had been extracted as a test set. As will be seen, the lack of a validation set did not seem to harm test data prediction; the most likely explanation is that standard procedure in towing tank experiments is to filter the raw signal data and average the measured resistance over the test time, thus reducing experimental noise to a minimum.

3.5.4 Prediction of Total Resistance (R_T)

It can be seen in Figure 3.8 the variation of R_T with Froude number Fr for all side hull configurations is in most places a smooth function; this can be well-approximated by a quadratic function, $R_T(Fr) = 25.269Fr^2 + 7.525Fr - 0.7129$.

It was thus decided to fit this function to the 272 training data examples and instead predict the residuals, notable only in the region of the prismatic humps.

Comparing PSO Training Methods

The first series of experiments compares the performance of BrPSO with standard PSO on the training data set, considering two alternate architectures with the same number of hidden units, but deployed differently in terms of layers. Because of the time-demanding nature of the experiments and the need to carry out a substantial number of runs it was decided to restrict the number of PSO fitness evaluations to 10,000, and compare performance at that point. The results of these experiments are shown in Table 3.4 and it is clear that BrPSO outperforms PSO-lBest for both architectures. These initial results also contradict the conclusions of Couser [173] and show that a two layer network can indeed provide substantially better predictive outputs.

		RMSE		MAE	
Training Method	Architecture	Mean	Min	Mean	Min
Standard PSO	3-20-1	6.4670.713	4.878	0.2450.016	0.205
Standard PSO	3-10-10-1	3.0660.386	2.144	0.1350.015	0.100
BrPSO	3-20-1	3.6131.024	2.148	0.1520.035	0.094
BrPSO	3-10-10-1	2.1100.446	1.078	0.0940.016	0.051

Table 3.4: Comparison of performance of standard and Breeding PSO (BrPSO) on the R_T training data set, in terms of Root Mean Squared Error (RMSE) and Mean Average Error (MAE) achieved after 10,000 iterations (50 independent runs)

Test data comparison of BrPSO prediction with actual R_T

The final set of training runs are carried out using BrPSO and the 3-10-10-1 architecture, and stopping when the PSO algorithm has converged and no global best update has been made for 1000 iterations. The global best weights are then used to make a prediction for the R_T curve for the mid-mid test data. Each run is then used to make independent predictions. The mean MAE over the test data was 0.118 ± 0.026, with 190867 ± 97662 iterations being taken (min 96061, max 419350), compared to a quoted mean MAE of 0.132 in [175]. The predicted R_T curve from an example run is shown in Figure 3.10 and it can be seen that the characteristic prismic hump is correctly captured. While these MAEs are not dissimilar

it should again be noted that the BrPSO results are obtained using only three model inputs, %TR, %LS, and Fr, while the work of [174] [175] requires two additional Reynolds number inputs in order to make any reasonable prediction for the resistance functions.

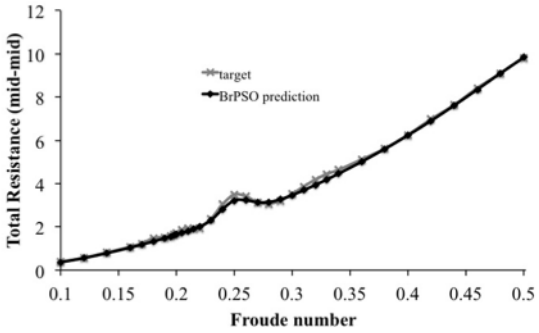


Figure 3.10: Test data comparison of BrPSO prediction with actual RT

3.5.5 Prediction of Residual Resistance Coefficient (C_R)

As in the case of RT, 10 runs were performed, with the same convergence criterion, giving in this case an average MAE of $2.424\times10^{-4} \pm 0.493 \times 10^{-4}$, with 27656 ± 11209 iterations being taken (min 15708, max 55332). When compared to the results of [175] it is clear from Figure 3.11 that the BrPSO model performs far better for this function than Bayesian Regularization, which was in turn more effective than the Levenberg-Marquard training used in the Tri-SWACH work of [174]. An example prediction is shown in figure 3.11, together with a representative example from [175]. It can be seen that the BrPSO-generated curve manages to capture the correct general shape of C_r but does not quite hit the resistance peak around $Fr = 0.25$, or show the full complexity of the side-lobes. These results are similar to those of Couser [173] for predicting C_R for catamarans, in both cases the predictions fail to capture the small double dip interference patterns that occur at low Froude numbers for both designs. This is due to the rest of the function being relatively smooth, and as such the NN produces a relatively smooth output for this obscure region, one way which may be used to further enhance prediction in this region could be to use additional localised networks to be trained in this region

which are aggregated with the more general network. Other approaches could use different weightings for the training samples based on approximated gradients, this would allow areas with a high degree of change to have more focus on in training.

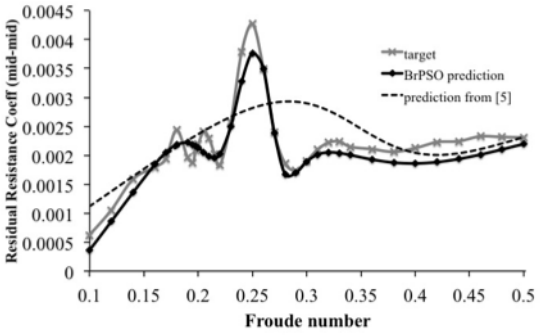


Figure 3.11: Test data comparison of BrPSO prediction with actual CR

From these results it can be seen that BrPSO provides a powerful tool for training neural networks to approximate what are deemed to be challenging function approximation problems in this example application.

Although excellent resistance prediction was achieved by the BrPSO-trained ANN, the results obtained are limited by the experimental data to one specific Tri-SWACH hullform, with only two parameters (%TS and %LS) varied in this configuration. Test predictions were also only made for the E configuration, and it would be interesting to see how this method performs when the test data is set to another configuration which has very different R_T and C_R behaviours comparatively. Furthermore, more elaborate neural network training and model aggregation techniques could be applied in an attempt to try and better capture the highly non-linear regions where wave interactions occur.

3.6 Conclusions

This work has developed a hybrid particle swarm algorithm with a discrete crossover operator. A particle is potentially replaced by a new child particle formed by breeding a randomly selected member of the population with the current global best using discrete crossover. The algorithm has been analysed analytically and empirically and then the search capability has been compared against other relevant PSO algorithms from the literature.

Analytical and empirical analysis in Section 3.3 shows the advantages of incorporating crossover operations into PSO and how it leads to a high probability of an improved particle position being formed. A lower bound for the probability of successful crossover is given, and it is seen that the overall behaviour of crossover success converges towards this bound as the dimensionality of the problem increases. It was also shown that using a tournament procedure to select parents further helps boosts crossover success probability. With respect to overall behaviour of the swarm it was seen that gBest crossover results in a clustering behaviour around the global best. This can accelerate convergence and is controlled by the crossover rate parameter. Currently the analytical analysis is limited and only provides very loose bounds for the behaviours, further work involving more complex hypergeometry and combinatorics analysis may be able to provide tighter bounds and a more detailed explanation of behaviour in all regions of the search space. It would also be worthwhile extending the analysis with respect to the assumed underlying random distributions, rather than assuming a uniform distribution it would be worthwhile to see how behaviour under a normal or levy distribution would change, as these may provide a better representation of particle behaviour.

To further enhance the ability of crossover a random mutation is added. A linear mutation is applied as a perturbation to the child particle's position. This can further increase the search range of crossover, although careful selection of parameters, $M_{\mathrm{F}} < 1$, is required to avoid search bias around the axes. Section 3.3.6 provides an empirical analysis on how the mutation parameters effect the search behaviour of the BrPSO($M_{\mathrm{P}}, M_{\mathrm{F}}$) algorithm. It was found that some degree of mutation signifi-

cantly enhances the search abilities of the BrPSO algorithm for all the test functions considered, but using static parameter settings resulted in a tradeoff between exploration and exploitation behaviour, to overcome this a dynamic adaptive mechanism was implemented.

In Section 3.4.2 a self-adapting mechanism for the mutation parameters was introduced. This was achieved by extending the PSO search space to encompass these parameters, as well as using a different form of the velocity update, Equation 3.40, to handle the dynamic nature of the parameter search. Comparing the new BrPSO-SAM algorithm against other relevant PSO algorithms on the CEC'05 benchmark problems it was seen that BrPSO-SAM is a competitive PSO algorithm. The main advantages of BrPSO-SAM are it's enhanced exploration capabilities and fast convergence; this makes it particularly well suited for functions with unbounded domains and uni-modal or low degree multi-modal landscapes.

It is worth mentioning though that when compared to minima found by state-of-the-art DE SHADE and it's variants for $f_{05}1 - f_{05}14$ all of the PSO algorithms and crossover hybrids considered here are still lagging in comparison. This suggest that there is still plenty of scope for additional work in hybridising PSO and DE for increasing the search ability of PSO. Rather than focusing on using only the crossover mechanism it maybe worth focusing on the elements that make SHADE and JADE successful compared to other DE algorithms which is the adaptive parameterisation and the use of archiving parameter values.

Other routes for further improving BrPSO-SAM could involve applying crossover operators in more advanced PSO base algorithms, similar to the analysis presented by Epitropakis [136] for PSO-DE hybrids. In current implementations BrPSO-SAM only uses the basic PSO-gBest which is known to have sub-optimal behaviour in some cases. BrPSO-SAM may also be improved by investigating further the effects the crossover rate and breeding probability have on the search capability for the CEC'05 benchmark functions, and perhaps extending self-adaptation mechanism to also incorporate these parameters.

In conclusion this work has provided a successful introduction and exploration

of a new powerful PSO-crossover hybrid and has shown that further elaboration and development of the ideas presented here can be a fruitful avenue for PSO research in the future.

Chapter 4

Calibrating the Heston Model using Evolutionary Algorithms

This chapter looks at the use of advanced evolutionary algorithms for the task of calibrating the stochastic volatility Heston model. The algorithms are tested by calibrating the Heston model for sets of artificial option price surfaces generated using predetermined Heston model parameters, this methodology allows for a direct evaluation and comparison of the algorithms performances. The algorithms tested consist of advanced particle swarm optimisation (PSO) variants, including the Breeding particle swarm optimisation introduced in Chapter 3, advanced differential evolution (DE) algorithms, and PSO/DE local search hybrids. In addition, this work introduces a PSO-DE hybrid algorithm that is able to utilise the observed strengths of both families of algorithms for the given task.

4.1 Introduction

When pricing financial derivatives it is important that the underlying models used for asset price behaviour, see Section 2.3, are able to replicate observed market prices as best possible. To acheive this, suitable model parameters need to be found. *Model calibration* for financial derivatives pricing is the procedure of fitting a parameterisable model of the underlying asset price behaviour, usually a set of stochastic differential equations (SDEs), to a set of observed market prices.

Model calibration is as important as deriving the theoretical model itself, if it

can not properly replicate market behaviour then it cannot be reliably used. The accuracy of the parameterisation of the model with respect to observed market prices has a direct impact on the profit and risk profile of a portfolio [179]. Two approaches to calibration can be used: *historical* or *implied*. Historical parameters are fitted to a time series of past data, whilst implied parameters are fitted to an array of option contracts at a fixed time, usually the current time. The implied parameters give an estimate of the market current conditions and from this the current market price, whilst historical parameters are more useful for predicting future market conditions and prices. This work is interested in implied parameters and from hereon calibration refers to obtaining the implied parameter set.

In the case of calibrating the Black-Scholes (BS) model only one parameter is required to be calibrated: the implied volatility, $\hat{\sigma}$, from the observed market prices, and in this case a closed form solution exists by using an inverse form of the analytical Black-Scholes solution. The issue with the BS model is it assumes that the implied volatility is a constant which may not be an accurate assumption [180]. In fact what is often observed is a volatility smile [118], where the implied volatility changes with resect to the moneyness and time-to-maturity of an option. To try and capture this behaviour stochastic volatility (SV) models are used.

More complex models which incorporate stochastic volatility are given in the form of

$$dS_t = \mu_t S_t dt + \sqrt{v} S_t dW_t^{(1)} \tag{4.1}$$

$$dv_t = \alpha(S_t, v_t, t)dt + \nu\beta(S_t, v_t, t)\sqrt{v_t} dW_t^{(2)}, \tag{4.2}$$

where $dW_t^{(1)}$ and $dW_t^{(2)}$t are two, possibly correlated, Brownian motions. In these cases the inverse problem for parameter estimation as simple as the BS case. These models have more than just one parameter that needs to be estimated, additionally these parameters can interact and influence each other, consequently this means a unique solution for the implied parameters not may exist. The Heston model [4] is the most popular SV model and uses a correlated mean reverting process to model

the volatility of the underlying asset:

$$dv_t = \kappa(\bar{v} - v_t)dt + \sigma\sqrt{v_t}dW_t^{(2)} \tag{4.3}$$

$$dW_t^{(1)}dW_t^{(2)} = \rho dt. \tag{4.4}$$

The price of a European call option, C, for asset price, S, strike price, K, and time-to-maturity, τ, calculated by the Heston model is

$$C(S,K,\tau) = SP_1 - e^{-r\tau}KP_2 \tag{4.5}$$

where P_1 and P_2 are the in-the-money probabilities $P_j = \Pr(\ln(S) > \ln(K)$ but obtained under different measures. The Heston model's popularity stems from the fact that this model has a known analytical form of the characteristic function [4] (cf) of the probability distributions P_j. To evaluate the Heston option price it only requires numerical integration of the complex integrals of P_j.

The Heston model requires five parameters to be calibrated: κ, the mean-reversion rate; $\bar{v}$, the long-term variance; σ, volatility of volatility; ρ, the correlation between the two Brownian motions $W_t^{(1)}$ and $W_t^{(2)}$, and finally v_0, the initial value of the volatility.

4.2 Heston Model Calibration

When calibrating the Heston model a simple analytical form of the inverse problems does not exist, therefore numerical minimisation/optimisation techniques are required. These methods are used to minimise a loss function between a set of observed market prices and parameterised model prices, where the algorithms are used to search over the 5D model parameter space. Let $\mathbf{C} = \{C_i(\tau,K)\}$ be the set of observed market prices for a set of options at a fixed time with varying time to maturity $\tau \in \{\tau_1 ... \tau_T\}$ and strikes $K \in \{K_1 ... K_k\}$, then the objective is to find the set $[v_0, \theta, \rho, \kappa, \sigma]$ s.t.

$$\text{argmin } \|\hat{\mathbf{C}}(v_0,\theta,\rho,\kappa,\sigma) - \mathbf{C}\| \tag{4.6}$$

Two assumptions are made, the first is that the market prices represent a unique solution for the parameterisation of the Heston model, and second, in the use case of some algorithms, is the search space is globally convex. There is debate as to the nature of the Heston calibration surface, whether it is globally convex or not. In some instances studies report the existence of local minima [179] [181] [182], but whether this an artefact of inadequacies in the optimisation method used or is characteristic of the search space is another question. Cui *at al* [127] use a new form of the analytical derivative to analyse the calibration space and determine that the structure is a narrow long valley with a flat basin for the optimal parameter set and that there is no evidence of local optimum. In respect to the artificial benchmark functions previously discussed this can be view as similar to the shifted Rosenbrock function, $f_{05}2$. Other factors to be considered when calibrating the Heston model are: calibration is sensitive to the loss function; pricing method; and the initial guess for starting values used in some minimisation algorithms [119].

When using gradient-decent based numerical optimisation approaches the methods can be slow and inefficient because the analytical derivatives are unknown and have to be computed numerically. Recently, Cui *et al* [127] has overcome the inefficiency of calculating the derivatives by using a new formulation of the Heston characteristic function, this allows them to obtain an analytical form of the Heston parameter derivatives which only requires numerical integration to be evaluated. This method represents the forefront of Heston model calibration in terms of speed and efficiency. Though, the method of Cui *et al* [127] is limited to the Heston model. In the case of more elaborate SV models [6] this method cannot be applied; for theses cases numerical optimisation techniques provide a more generalised and flexible approach for model calibration.

The other issue when using unconstrained minimisation algorithms, such as Nelder-Mead or Levenberg-Marquadt, is that they are dependant on an initial guess of the solution [183]; this requires past experience and judgement of the practitioner which questions the robustness of these methods of calibration [184]. A more analytical approach is the Smart Parameter method introduced by Gauthier [183] where

an initial guess for ρ and σ are made by solving a simultaneous equation of two similar put options, although there is no guarantee on the robustness of this approach. In the EA-hybrid approach by Gilli *et al* [185] the PSO and DE population are used to provide initial starting guesses for Nelder-Mead minimisation which eliminates the need for any user input and proves to be more efficient and accurate than either of the approaches used individually.

As an alternative to unconstrained minimisation techniques meta-heuristic optimisation algorithms, such as EAs provide an appealing alternative. Firstly they do not require any additional information about the function i.e. derivative calculations, or rely on an initial guess of the solution, and as has been shown these algorithms can efficiently explore over intricate search space landscapes. Even though when compared to unconstrained minimisation algorithms EAs are more costly in terms of computational runtime the parameter estimations are more accurate and consistent [119]. Moreover, these methods provide an easy to implement and more flexible approach for model calibration.

4.2.1 Heuristic Calibration Methods

Evolutionary algorithms such as particle swarm optimisation and differential evolution have previously been used as methods to calibrate the Heston model [186] [183] [119] [185] [185] [187]. In one of the first approaches for applying DE to Heston model calibration Vollrath *et al* [186] apply DE/rand/1/bin and use parameter settings N = 15D, CR = 0.5, and F = 0.8. When compared against Levenberg-Marquardt (LM) and Downhill Simplex (DS), despite the DE taking 1093s compared to 76s for LM, the error norm was 1000x smaller compared to DS whilst LM completely failed. Further experiments found it took 200 generations, which at the time took 20 minutes on their computing architecture, for DE to converge to a stable optima. Gauthier *et al* [183] find that comparing their smart parameter estimation combined with LM is over 100x faster than DE to acheive the same level of accuracy. These results suggests that DE is capable of achieving at least the same level of accuracy as gradient-decent methods but has the disadvantage of being computationally slower.

Gilli et al [185] further investigate the use of evolutionary optimisation algorithms both DE and PSO for calibrating the Heston and Heson-Bates options pricing models on artificial test data. They use DE/rand/1/bin, and PSO-gBest, and they also investigated the use of hybrid algorithms by incorporating local Nelder-Mead (NM) searches into the EAs. In these hybrids, after every 3 iterations members of the population would be selected to be used as the initial guess for the NM search. The motivation behind this approach is that although EAs have good global search abilities, convergence can be slow and assuming there is either non or very few local minima using NM can speed up convergence towards the global optimum. They find that for all the number of fitness evaluations (1250, 5000 and 20000) the solutions of all the algorithms give a pricing error of $< 1\%$ for the calibrated Heston model, and after an extended number fitness evaluations they all converge to the exact parameter set. The hybrids outperformed the regular EAs, with PSO being the worse, calibration using only DE was only slightly worse than using the hybrid DE-NM algorithm. For the more elaborate Heston-Bates model [6], calibration results were not as good as for the basic Heston model, this is due to a higher degree of interaction between the Heston-Bates parameters resulting in a more heavily multi-modal search space.

Haring [187] *et al* diverge from the use of either DE or PSO and focus on using the Cuckoo Search (CS) algorithm, this is most similar to BBPSO using a Levy distribution, for calibrating American style options under the Heston model. They find convergence is achieved after around 400 iterations, which corresponds to approximately 8000 fitness evaluations, although there is no context for these results with respect to comparison against other known algorithms.

It can be seen that even with the use of hybrid algorithms the fundamental DE and PSO algorithms used in the calibration literature are extremely rudimentary with respect to the EA literature. Consequently EAs are not being best represented in this application, and although previous studies show potential of EAs it may be that more advanced variants of these algorithms can result in better performance, i.e. higher accuracy with fewer fitness evaluations. The popularity and use of state-

of-the-art EAs remains largely confined to those in the field of EA research. This work aims at comparing the use of more advanced EA algorithms that are available from literature, as well as the newly introduced BrPSO algorithm (see Chapter 3), for calibrating the Heston model.

4.3 Evolutionary Algorithms Investigated

There are numerous variations of PSO and DE algorithms within the literature as has previously been discussed in Section 2.1. Here it has been choose to investigate the use of some of the more popular variations which have been used in other comparative studies [32] [73]. The full list of EAs used here is given in Table 4.3.

The PSO algorithm used for Heston calibration by Gilli *et al* [185] was the basic PSO-gBest, for which they use an inertia coefficient with the velocity update rather than a constriction factor. Constriction factors have been shown to improve performance [19] making PSO-gBest with constriction factor (PSO-gBest-cf) one of the algorithms tested. Further to adding a constriction factor it is known that using local topologies can help avoid local minima (if any exist in this case) and also test the slightly less rudimentary PSO-lbest-cf with ring topology. Furthermore, the popular PSO variants used in the comparative study [32] are also tested.

In addition to these popular PSO variants the use of the newly introduced Breeding particle swarm optimisation algorithm (BrPSO) is used. Given that this problem is essentially unbounded, as only an initial domain can be guessed and the solution may exist outside of the initial domain. BrPSO and BrPSO-SAM provide hopeful prospects as they have shown exceptional searching capability for unbounded domains and fast global convergence for unimodal problems (Section 3.4.1).

For the DE algorithms, first of all the DE/rand/1/bin which is used by both Vollrath [186] and Gilli [185] is tested. Given that it is known that the Heston parameters can influence each other, this corresponds to a rotated problem, as such the use of exponential crossover is tested using DE/rand/1/exp. Finally, from the literature it is clear that SHADE and L-SHADE [73] are currently the best DE

Algorithm	Description
	Particle Swarm Optimisation (PSO)
PSO-gBest [8]	Global best PSO variant used for calibration by Gilli *et al* [185].
PSO-gBest-cf [19]	Global best PSO with constriction factor.
PSO-lBest-cf [19]	Local best PSO using ring topology and constriction factor.
CLPSO [32]	Comprehensive Learning PSO, using an exemplar based velocity update equation.
CPSO [43]	Cooperative PSO, uses sub-swarms for each search dimension.
wFIPS [31]	Fully informed PSO uses a velocity based on weighted contribution of the whole local neighbourhood.
UPSO [30]	Unified PSO uses a linear combination of global and local best learning.
FDR [45]	Fitness-distance-ratio PSO, uses a velocity term based on maximising the fitness-distance ratio,
BrPSO(0,0)	Hybrid PSO with binomial crossover without mutation.
BrPSO-SAM	Hybrid PSO using crossover with self-adaptive linear mutation.
	Differential Evolution (DE)
DE/rand/1/bin [53]	Basic DE with random parent mutation and binomial crossover, used for calibration by Vollrath *et al* [186] and Gilli *et al* [185]
DE/rand/1/exp [53]	Basic DE with random parent mutation and exponential crossover.
jDE [76]	DE/rand/1/bin, using an adaptive control parameter search.
JADE [78]	Historical p-best mutation using an external archive.
SHADE	Adaptive control parameters using a historical external archive.
L-SHADE [73]	SHADE with linear decreasing population.
	Local Search Hybrid
PSO-gBest-NM [185]	Global best PSO with Nelder-Mead local search, used for calibration by Gilli *et al* [185].
DE/rand/1/bin-NM [185]	Basic DE with Nelder-Mead local search, used for calibration by Gilli *et al* [185].
L-SHADE-NM	L-SHADE with Nelder-Mead local search, introduced in this work.
PSO-L-SHADE-NM	Initial PSO-gBest-NM with switching to L-SHADE-NM, introduced in this work..

Table 4.1: Full list of the evolutionary algorithms used in this work for calibrating the Heston model. A more detailed description of these algorithms can be found in Section 2.1.

algorithms [188], these are used along with two other popular adaptive DE variants, jDE [76] and JADE [78].

Following Gilli *et al* [185] the hybrid versions of the PSO-gbest and DE/rand/1/bin algorithms with a Nelder-Mead (NM) local search are also used for comparison. These hybrids use a local search are every 10 PSO/DE iterations and use the top three fittest population members as the starting points for the local search, these population members are then updated accordingly with respect to the local search results. In addition to the hybrids introduced by Gilli *et al*, two new hybrids are proposed in Section 4.6: L-SHADE-NM and PSO-L-SHADE-NM.

4.4 Methodology

The calibration problem is setup using an artificial data set of option prices as the target calibration price surface, **C**. The target option prices of the calibration surface are generated using the Heston model price equation with a known set of predetermined model parameters. The same parameter sets and experimental setup as Gilli *et al* [185] is used. Using an artificial data set makes it possible for the calibrations to find exact parameter settings for the Heston model and allows for direct interpretation of the quality of the calibration results. Table 4.2 gives the parameter sets used for the ten Heston calibration experiments.

Set	1	2	3	4	5	6	7	8	9	10
$\sqrt{v_0}$	0.3	0.3	0.3	0.4	0.2	0.5	0.6	0.7	0.8	0.3
$\sqrt{\theta}$	0.3	0.3	0.2	0.2	0.4	0.5	0.3	0.3	0.3	0.2
ρ	-0.3	-0.7	-0.9	-0.5	-0.5	0	-0.5	-0.5	-0.5	0.0
κ	2.0	0.2	3	0.2	0.2	0.5	3.0	2.0	1.0	3.0
σ	1.5	1.0	0.5	0.8	0.8	3.0	1.0	1.0	1.0	0.5

Table 4.2: The ten parameter sets used to generate the artificial calibration surface data for the Heston model, these are the same as used in [185].

The calibration surface, **C**, is comprised of 147 call options, $C(\tau,K)$, taken over a grid of strike prices, K, and maturity times, τ, these are again the same as used in [185], $\tau \in \{\frac{1}{12}, \frac{3}{12}, \frac{6}{12}, \frac{9}{12}, 1, 2, 3\}$, $K \in \{80, 82...120\}$, with a spot price of $S_0 = 100$ and interest rate $r = 0.02$.

4.4.1 Loss Function

The objective of the optimisation algorithms is to minimise the error between the calculated call prices using the parameterised Heston model with parameters found from the search space and the equivalent set of known prices given by the calibration surface, $\mathbf{C}$. Following the methodology of Gilli *et al* [185] the fitness function of an EA population member, $\mathbf{x}_i \in \mathbb{R}^5$, is measured as the sum of the relative errors

$$\text{fit}(\mathbf{x}_i) = \sum_t \sum_k \frac{|C_{t,K}(\mathbf{x}_i) - C_{t,K}(\mathbf{P}_p)|}{|C_{t,K}(\mathbf{P}_p)|}, \tag{4.7}$$

where $\mathbf{P}_p$, $p \in \{1,2\ldots10\}$, is one of the predetermined parameter sets used to generate the calibration surface for each experiment given in Table 4.2. It should be noted that in the denominator the absolute value is taken, this is due to the possibility of negative prices occurring from the numerically generated calibration surface. Without taking the absolute value, following the original experimental design by Gilli *et al* [185], it was found here that this could lead to problems of creating negative fitness values creating unstable calibrations. In practical applications negative prices do not occur and hence taking the absolute of the denominator is not usually required.

The error norm, ε, of the calibrated parameter set, $\mathbf{x}_{\text{opt}}$, is given by the Euclidian distance with respect to the known parameter set used to generate the calibration surface and the optimal set found by the algorithm a,

$$\varepsilon(\mathbf{x}_{\text{opt}})_p^a = \|\mathbf{x}_{\text{opt}} - \mathbf{P}_p\|_2. \tag{4.8}$$

4.4.2 Additional Considerations

With respect to pricing options under the Heston model there are many factors and variations to consider, to provide full clarity of the experimental setup specifics of the pricing methodology used here are noted.

Heston Characteristic Function

There are many forms of the Heston characteristic function [4] [121] [122] [123] [124] [125], the pricing in this work uses the formulation of Albrecher's Little Trap [123]. This is shown to give more numuerically stable results for the integration than the original characteristic function posed by Heston [4]. Albrecher *et al* [123] use a mutliplative factor to separate the characteristic function into a more numerically stable form. The characteristic function is calculated as:

$$
\begin{aligned}
f &= e^{A+B+C} \\
d &= \sqrt{(\rho\sigma \mathrm{i}\omega - \kappa)^2 + \sigma^2(\mathrm{i}\omega + \omega^2)} \\
g &= \frac{(\kappa - \rho\sigma \mathrm{i}\omega - d)}{(\kappa - \rho\sigma im\omega + d)} \\
A &= \mathrm{i}\omega(\log(S) + (r-q)\tau) \\
B &= v_t \frac{\kappa}{\sigma^2}((\kappa - \rho\sigma \mathrm{i}\omega - d)\tau - 2\log(\frac{1 - ge^{-dtau}}{1 - g2}) \\
C &= \frac{v_0}{\sigma^2}\frac{(\kappa - \rho\sigma \mathrm{i}\omega - d)(1 - e^{-d\tau})}{(1 - ge^{-d\tau})}
\end{aligned}
\tag{4.9}
$$

where ω is the domain of integration.

Numerical Integration

The numerical integration scheme can effect both the accuracy the runtime performance of the optimisation algorithm. With respect to pricing method Rouah [119] tests the fast-fourier-transform (FFT) against Gauss-Legendre integration for parameter estimation, FFT shows a marginally better errors. However, when tested with EAs for calibration Gilli *et al* [185] found that there is little difference between the integrations schemes used, though, in Section 4.8 it will be seen how observed local minima can be an artefact of the integration scheme used.

In this work a 16-point Gauss-Legendre (GL-16) scheme is used. The integration of the characteristic function is performed over the domain of $\omega \in [0, b]$, where b is sufficiently large enough that the integration stabilises. To reduce the number of calculations required during numerical integration of the probabilities P_1 and P_2

the Strike Vector method [189] can be used.

Strike Vector Computation

One of the major drawbacks of heuristic methods is the number of times the fitness function needs to be evaluated. To speedup computation the Strike Vector method [189] is used. This is a simple computational strategy that reduces the number of times the probability density function (PDF) needs to be computed. The PDF is not dependant on the strike price K, hence for all options with the same maturity the PDF only needs to be computed once. When using numerical integration schemes such as Gauss-Legendre the PDF is computed once for each point of integration for each maturity time. Algorithm 4.1 shows the Strike-Vector method applied with numerical integration for pricing a grid of options over a set of maturities $\tau \in \{\tau_1 \dots \tau_T\}$ and strikes $k \in \{k_1 \dots k_K\}$.

Algorithm 4.1 Strike Vector Method for Heston Call Prices

Calculate vector of integration points and weights ω,$\mathbf{w}$
for $\tau \in \{\tau_1 \dots \tau_T\}$ **do**
  Calculate vectors of characteristic function integrands, $\text{cf}_1(\omega, \mathbf{w})$ and $\text{cf}_2(\omega, \mathbf{w})$
  for $k \in \{k_1 \dots k_K\}$ **do**
    Calculate integrals $P_1(\text{cf}_1, \omega, \mathbf{w}, S, k, \tau, r)$ and $P_2(\text{cf}_2, \omega, \mathbf{w}, S, k, \tau, r)$
    $C(\tau, k) = SP_1 - e^{-r\tau}KP2_2$;
  end for
end for

4.5 Results and Discussion

Firstly the global best parameter estimations will be discussed, this looks at the errors of the parameter estimations achieved after an excessive 20,000 fitness evaluations. This gives an indication of the overall potential of how well an algorithm can perform given an excess of time and it's potential for much harder problems, but in practice computation regarding the number of fitness evaluations is often limited and only a certain level of accuracy for the parameter estimations need to be achieved. The second part of the discussion looks at the performance of the algorithms from this more practical standpoint, the number of fitness evaluations are limited to 1000 and 5000, and also how many fitness evaluations it takes to achieve

a practical level of accuracy for the parameter estimations. Before proceeding with the discussions the metrics used to to evaluate the algorithms performances are introduced.

4.5.1 Error Measures

To compare the performance, which here refers to the accuracy of the parameter estimations, of the EA algorithms the error-norms of N independent runs for each parameter set are aggregated. Two methods are used to aggregate the results of the N runs, the first is the simple mean and the second is the 75% quantile, Q_{75}. It is proposed in [190] that using quantiles provides a more robust measurement of a stochastic search algorithms performance because it gives a probability bound of the expected algorithms performance, and unlike the mean it is less biased by large outlier values. For algorithm a for parameter set p and with runs $n \in \{1,2\ldots N\}$ the overall performance is measure as:

$$E^a_{p,\text{mean}} = \frac{1}{N}\sum_{n=1}^{N} \varepsilon^a_{p,n} \tag{4.10}$$

$$E^a_{p,Q_{75}} = \varepsilon \text{ s.t. } \Pr(\varepsilon^a_{p,n} < \varepsilon) = 0.75, \tag{4.11}$$

where ε is the parameter estimation error norm given in Equation 4.8.

Three separate metrics are used to aggregate the performance of an algorithm across all experimental parameter sets. The first is the simple mean, but given the large range of magnitudes of the error measures across all parameter sets this value to expected to be highly biased, although it does give an indication of overall reliability if the standard deviation is low enough. The second metric is the *log-mean*, which is calculated by taking the mean of the $\log_{10}$ error measures, this gives a more balanced aggregation across a wide range of magnitudes. Finally, the *standardised-log-mean* (SLM) is used; firstly for each experimental parameter set the $\log 10$ error measures for each algorithm are normalised such that the distribution of error measures for each parameter set have a mean $\mu_{\log 10,p} = 0$ and standard deviation $\sigma^2_{\log_{10},p} = 1$; this gives a relative metric of how well each algorithm performs with respect to the mean performance of all algorithms on a given parameter set. The

final metric is then the sum of the SLM over all parameter sets for the given algorithm. The three metrics for the overall performance, M^a, of algorithm a over all P number of experimental parameter sets are

$$M^a_{\text{mean}} = \frac{1}{P}\sum_{p=1}^{P} E^a_p \tag{4.12}$$

$$M^a_{\text{log-mean}} = \frac{1}{P}\sum_{p=1}^{P} \log_{10}(E^a_p) \tag{4.13}$$

$$M^a_{\text{slm}} = \sum_{p=1}^{P} \frac{\log_{10} E^a_p - \mu_{\log_{10},p}}{\sigma^2_{\log_{10},p}}. \tag{4.14}$$

where the error measure E^a_p can either be the mean or 75% quantile given in Equations 4.10 and 4.11 respectively.

4.5.2 Global Best Estimations

Here the overall best performance of the algorithms are evaluated. Table 4.3 gives the 75% quantile, Q_{75}, and the mean error-norms for each of the algorithms over 30 independent runs for each experimental parameter set after 20,000 fitness evaluations, from this the maximum capability of the algorithms can be assessed. It should also be noted that the discussions exclude the results of parameter set 2 for which all algorithms find large local optimum, this is discussed in more detail in Section 4.8

PSO Performance

First of all it can be seen that the DE algorithms provide far more accurate solutions than the majority of PSO algorithms. The more advanced PSO algorithms perform relatively poorly, whilst the simpler PSO algorithms give more accurate and consistent results across all parameter sets. Of the advanced PSO algorithms it is surprising to see that CLPSO, even though performing extremely well on the DeJong and CEC benchmark functions [32] [90], fails to provide any significant results in this application, showing respectively large error norms of order of 10^{-1}-10^{-2} for all parameter sets. CPSO and CLPSO are both the worst of all the algorithms tested here, both show high log-means of close to 0 and SLMs close to 1, meaning that on

75% Quantile Euclidian distance ($E^a_{p,Q_{75}}$)										
	1	2	3	4	5	6	7	8	9	10
PSO-gB	4.72e-03	3.04e+00	6.10e-03	2.03e-04	2.58e-03	1.10e-01	6.48e-02	5.64e-02	1.15e-01	2.70e-01
PSO-gB-cf	5.45e-06	3.58e+00	1.19e-05	3.21e-09	4.50e-07	3.40e-02	1.46e-02	3.60e-03	7.49e-02	2.21e-01
PSO-lB-cf	1.69e-03	4.11e+00	1.39e-03	2.35e-06	1.53e-04	1.63e-01	7.87e-02	4.32e-02	1.26e-01	1.77e-01
BrPSO	3.58e-09	4.47e+00	4.00e-08	3.84e-12	1.91e-10	3.41e-03	8.49e-04	4.64e-05	4.20e-03	5.27e-02
BrPSOSAM	1.69e-05	2.67e+00	9.45e-06	1.32e-08	1.25e-06	3.26e-02	1.18e-02	5.14e-03	7.11e-02	2.35e-01
CLPSO	4.38e-01	4.15e+00	4.37e-01	1.95e-01	9.35e-02	7.32e-01	3.78e-01	3.58e-01	2.98e-01	5.43e-01
CPSO	1.72e+00	3.51e+00	2.39e+00	1.85e+00	2.61e-01	2.15e+00	9.46e-01	1.89e+00	1.57e+00	1.19e+00
UPSO	9.53e-03	4.44e+00	1.63e-02	2.27e-06	4.62e-04	2.55e-01	1.56e-01	8.12e-02	1.87e-01	3.31e-01
wFIPS	7.17e-02	4.89e+00	4.77e-02	1.07e-02	9.62e-03	4.69e-01	2.67e-01	9.44e-02	1.32e-01	1.64e-01
FDR	2.44e-03	3.31e+00	1.28e-03	1.94e-05	1.16e-04	1.13e-01	5.86e-02	4.49e-02	1.30e-01	2.28e-01
DE/r/1/b	6.48e-05	2.33e+01	7.92e-05	1.04e-05	5.82e-06	7.02e+01	5.01e-05	8.72e-06	1.64e-05	3.88e-05
DE/r/1/e	6.34e-05	1.39e+01	7.38e-05	1.72e-05	9.45e-06	7.00e+01	4.90e-05	2.25e-05	2.42e-05	3.84e-05
JADE	7.70e-12	3.90e+00	4.53e-12	9.02e-12	1.52e-12	2.77e-09	1.03e-11	9.84e-13	6.98e-13	2.83e-12
jDE	2.70e-08	1.66e+00	3.60e-08	1.75e-08	1.27e-08	1.11e-02	5.40e-06	2.73e-08	3.84e-08	1.11e-07
SHADE	2.31e-12	3.59e+00	5.30e-12	2.03e-12	5.93e-13	9.37e-11	7.78e-12	1.57e-12	2.59e-12	1.59e-12
L-SHADE	5.99e-13	2.77e+00	1.04e-12	4.00e-13	3.06e-14	1.22e-12	4.59e-15	2.67e-14	2.50e-14	2.71e-14
PSO-gB-NM	3.16e-05	4.26e+00	1.33e-05	3.89e-05	2.19e-05	3.06e-05	4.61e-05	2.92e-05	3.79e-05	3.23e-05
DE/r/1/b-NM	2.99e-05	7.42e+00	2.47e-05	6.86e-05	2.34e-05	2.48e-05	8.18e-05	4.13e-05	3.23e-05	4.72e-05
Mean Euclidian distance ($E^a_{p,\text{mean}}$)										
	1	2	3	4	5	6	7	8	9	10
PSO-gB	4.02e-03	2.61e+00	4.05e-03	1.80e-04	1.63e-03	8.52e-02	5.07e-02	3.54e-02	8.37e-02	2.07e-01
PSO-gB-cf	7.56e-05	2.77e+00	2.06e+00	2.41e-09	5.22e-07	2.65e-02	9.34e-03	3.55e-03	4.86e-02	1.40e-01
PSO-lB-cf	1.27e-03	2.64e+00	1.04e-03	2.40e-06	1.28e-04	1.33e-01	5.56e-02	3.00e-02	9.39e-02	1.04e-01
BrPSO	5.32e-07	2.62e+00	3.17e+00	2.08e-12	1.75e-10	1.80e-03	5.06e-04	4.18e-05	2.73e-03	3.38e-02
BrPSOSAM	2.20e-05	1.91e+00	1.43e-05	1.28e-08	1.27e-06	1.28e+00	7.42e-03	5.53e-03	5.77e-02	1.38e-01
CLPSO	2.73e-01	3.35e+00	3.06e-01	1.90e-01	6.51e-02	5.15e-01	3.02e-01	2.09e-01	2.21e-01	3.44e-01
CPSO	1.25e+00	2.43e+00	1.77e+00	8.03e-01	3.94e-01	1.81e+00	9.15e-01	1.45e+00	1.08e+00	7.86e-01
UPSO	6.30e-03	3.43e+00	1.47e-02	5.26e-06	4.96e-04	2.17e-01	1.06e-01	4.87e-02	1.40e-01	1.95e-01
wFIPS	5.46e-02	3.58e+00	3.80e-02	6.56e-03	6.10e-03	3.95e-01	2.19e-01	7.94e-02	9.68e-02	1.27e-01
FDR	1.81e-03	2.29e+00	8.80e-04	1.52e-05	1.40e-04	9.53e-02	4.39e-02	2.68e-02	9.18e-02	1.52e-01
DE/r/1/b	4.45e-04	1.82e+01	1.63e+01	2.23e-05	5.21e-06	5.59e+01	3.28e-05	8.40e-06	1.36e-05	2.80e-05
DE/r/1/e	1.27e-03	9.50e+00	3.11e-01	1.17e-04	8.72e-06	4.67e+01	3.79e-05	1.90e-05	1.69e-05	2.83e-05
JADE	9.47e-12	2.37e+00	4.16e-12	1.16e-11	3.98e-12	1.78e-04	1.25e-11	2.51e-12	7.63e-13	2.37e-11
jDE	2.27e-08	1.04e+00	3.05e-08	2.00e-08	8.19e-09	5.70e-03	5.39e-06	3.01e-08	1.18e-07	2.80e-07
SHADE	2.33e-12	1.69e+00	4.01e-12	1.34e-12	5.45e-13	2.36e-10	5.77e-12	9.39e-13	1.75e-12	1.78e-12
L-SHADE	4.77e-13	1.42e+00	8.35e-13	2.60e-13	1.72e-14	8.24e-13	2.31e-14	2.40e-14	1.61e-14	1.94e-14
PSO-gB-NM	1.72e-05	2.42e+00	9.31e-06	2.77e-05	1.58e-05	2.22e-05	4.07e-05	2.09e-05	2.55e-05	2.51e-05
DE/r/1/b-NM	2.24e-05	6.51e+00	1.98e-05	5.24e-05	1.62e-05	1.87e-05	6.38e-05	3.32e-05	2.48e-05	3.05e-05

Table 4.3: The 75% quantile and mean (respective standard deviations can be found in Table C.1) of the Euclidian distances between the parameter sets found by the EAs and the known optimal parameter set (1-10).

	Mean			Q75			Rank	
	Mean	Log Mean	SLM	Mean	Log Mean	SLM	Mean	Std
PSO-gB	5.25e-02	-1.65	0.54	7.00e-02	-1.52	0.56	13.17	2.32
PSO-gB-cf	2.54e-01	-2.65	0.28	3.87e-02	-3.20	0.14	10.00	1.79
PSO-lB-cf	4.66e-02	-2.06	0.43	6.56e-02	-1.94	0.49	11.17	1.72
BrPSO	3.57e-01	-4.12	-0.11	6.80e-03	-4.99	-0.28	7.17	3.92
BrPSOSAM	1.65e-01	-2.94	0.17	3.96e-02	-3.06	0.13	9.33	1.75
CLPSO	2.69e-01	-0.51	0.86	3.86e-01	-0.36	0.90	16.17	1.33
CPSO	1.14e+00	0.05	0.96	1.55e+00	0.17	1.01	17.33	1.03
UPSO	8.09e-02	-1.66	0.57	1.15e-01	-1.58	0.60	13.50	1.87
wFIPS	1.14e-01	-1.06	0.73	1.41e-01	-0.94	0.78	14.83	2.04
FDR	4.58e-02	-1.99	0.43	6.42e-02	-1.87	0.48	10.83	2.32
DE/r/1/b	8.02e+00	-2.80	0.53	7.80e+00	-3.38	0.42	13.00	4.20
DE/r/1/e	5.22e+00	-2.82	0.44	7.78e+00	-3.31	0.36	12.33	3.78
JADE	1.97e-05	-9.34	-1.38	3.12e-10	-10.00	-1.51	3.00	0.00
jDE	6.34e-04	-5.94	-0.65	1.24e-03	-5.96	-0.64	4.67	1.03
SHADE	2.83e-11	-10.33	-1.69	1.30e-11	-10.26	-1.59	2.00	0.00
L-SHADE	2.77e-13	-11.74	-2.07	3.75e-13	-11.67	-1.97	1.00	0.00
PSO-gB-NM	2.27e-05	-4.17	-0.10	3.13e-05	-4.01	0.00	5.17	0.98
DE/r/1/b-NM	3.13e-05	-4.01	0.07	4.16e-05	-3.89	0.11	6.33	1.03

Table 4.4: Total error measures, mean, log mean and standardised log mean (SLM) and average rank, over all the parameter sets, for both the mean and 75% quantile errors in Table 4.3 (respective standard deviations can be found in Table C.2).

average they are one standard deviation worse than the average performance of all the algorithms tested. Slightly better are UPSO, FDR and wFIPS, they all perform very similarly but UPSO and FDR show the better performance for parameter sets 4 and 5 with error-norms of order 10^{-6}-10^{-5}, and FDR shows marginally better overall performance out of the three with the lowest SLMs.

The best overall performing PSO algorithms are the two BrPSO algorithms, with BrPSO (not mutation) being better than BrPSOSAM. Even though the mean-error-norm metrics are high, due to one large outlier in each case skewing the results, occurring in sets 3 and 6 respectively, the negative SLM for BrPSO shows that overall it performs better than average. BrPSO performs better than all PSO algorithms on every parameter set, and for sets 1,3,4 and 5 it performs similar to the state-of-the-art DE algorithms with error-norms of order $10^{-8} - 10^{-12}$, but it is seen to find sets 9 and 10 the hardest. BrPSOSAM, shows the same pattern of performance for each of the parameters set but is always a couple of magnitudes worse that the BrPSO error-norms. The core BrPSO behaviour is the same for each algorithm, this shows that in this application the self-adaptive behaviour in BrPSOSAM slows down the convergence of the BrPSO algorithm. Comparing this observation with previous results, when using BrPSO on benchmark optimisation function, hints that the search landscape has a low-degree of modality and is close to being unimodal, which favours the faster converging BrPSO with no mutation.

Surprisingly the basic PSO algorithms do extremely well compared to their more advanced counterparts, PSO-gBest-cf for the Q_{75} values it has a low log-mean of -3.20 and SLM=0.14, with the closest other PSO algorithms, excluding BrPSO, are PSO-lBest-cf and FDR with log-means of around -1.8 and SLMs of 0.48. The performance of PSO-gBest-cf is similar to BrPSOSAM but not as good as BrPSO; this is not too surprising because BrPSO used PSO-gBest-cf as the base algorithm but then enhances the exploration and exploitation by using crossover. Both PSO-gBest and PSO-lBest-cf have similar performance with PSO-lBest-cf being slightly better with lower log-mean. Compared to PSO-gBest-cf both of these algorithms have slower convergence, PSO-gBest because it does not use a constriction factor,

and PSO-lBest-cf because of the weaker information sharing topology, this further supports previous claims that in this application faster converging algorithms perform better.

Overall the PSO algorithms show relatively disappointing performance, excluding BrPSO and PSO-gBest-cf. This could partially be because the majority of PSO algorithms have been developed for multi-modal functions, such as CLPSO, whilst in this application there is growing evidence to support that the Heston calibration search landscape has low modality and that simplicity prevails. In addition this also shows there is a need for a more versatile and adaptive PSO algorithm to provide a robust optimiser for all applications; BrPSOSAM provides this performing well on the multi-modal test functions but also being able to adapt for applications such as here.

DE Performance

Comparing the basic variations of the two EA families, PSO-gBest-cf and DE/rand/1/*, looking at the mean error metrics both DE algorithms are two magnitudes larger than the PSO means which is caused by DE/rand/1/* poor performance for parameter set 6, furthermore the Q_{75} results indicate that this is not due to an outlier. On the other hand the log-means are comparable between DE/rand/1/* and PSO-gBest-cf, so although DE/rand/1/* has a large error for set 6, DE/rand/1/* achieves similar or lower error-norms than PSO on the other sets. Compared to PSO-gBest-cf excluding set 6 DE has performs more consistently of with error-norms of around order 10^{-5} whilst PSO shows better performance for sets 1,4 and 5 but is significantly worse for sets 7,8,9 and 10.with error-norms of around $10^{-}2$. Furthermore, it can be seen from the convergence plots in Figure 4.1 that the basic DE shows slower initial convergence than the PSOs, but it is only for a high number of fitness evaluations, (> 10000), that its convergence accelerates past PSO. Overall, based on the SLM and given that it was able to successfully optimise all the sets compared to DE/rand/1/* PSO-gbest-cf is the more reliable optimiser.

Comparing the state-of-the-art DEs used here, jDE, JADE, SHADE and L-SHADE, it is obvious that they all outperform all of the PSO algorithms. L-SHADE

is by far the best algorithm used and shows incredibly low and consistent final mean error norms of order 10^{-13}-10^{-14} for all parameter sets. It is also worth noting that SHADE and L-SHADE are the only DE algorithms that show high levels of performance on set 6, whilst jDE and JADE both show a lapse in performance with mean error-norms of order 10^{-4}-10^{-5}. Set 6 shows a high level of variance for both jDE and JADE, as well causing issues for the two basic DE algorithms, all of this suggests that this set shows some sort of characteristic that is disruptive for DE but perhaps less so for PSO, one possible situation could be a deceptive-function [86] type shape where the local and global minimum are separated by a flat basin with long steep sides.

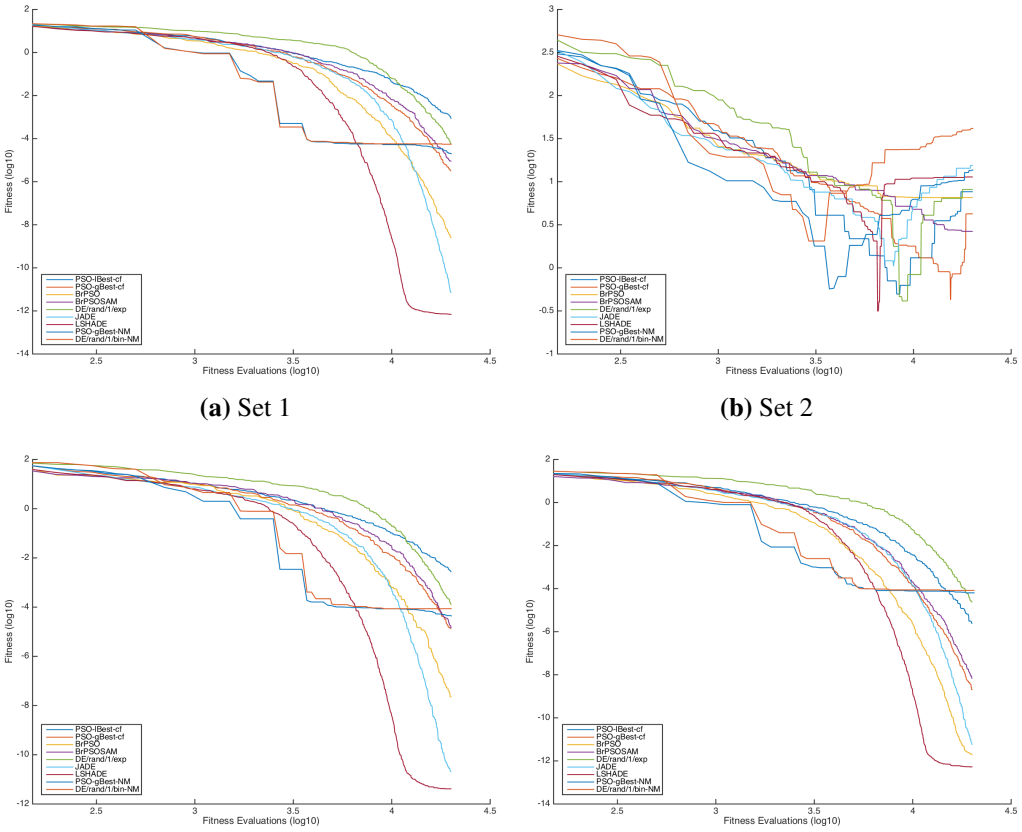
Basic Hybrids

It should first be noted that this discussion excludes the use of new L-SHADE hybrids which are introduced later on. It can be seen from the convergence plots, Figures 4.1, that the hybrid algorithms initial converge considerably faster than all the other EA algorithms considered. Compared to their un-hybridised counter-parts, the hybrid algorithm improves on the results of the PSO-gBest algorithm showing that this additional convergence added by the Nelder-Mead search is beneficial, but for DE/rand/1/bin the final error norms are around the same, although there is improvement for set 6. Overall hybridisation improves both PSO and DE algorithms. The hybrids works well because the explorative behaviour of the EAs provide a good starting points for Nelder-Mead local search to then refine, given the similarity between the results of the two hybrids it shows that the local-search becomes the dominant search behaviour and that the EAs both work as well as each other to get the local-search to within the region of the global basin of attraction. However, there is an exploration-exploitation tradeoff, in this case the use of the local-search algorithm limits the exploration ability of the EAs and as can be seen in Figure 4.1 this results in the algorithms prematurely converging and the population stagnating too quickly. This means that the hybrid algorithms can not take advantage of further exploration and refinement of solutions seen in L-SHADE and BrPSO algorithms for larger numbers of fitness evaluations, and as a results the final parameter

estimations are not as good.

Summary

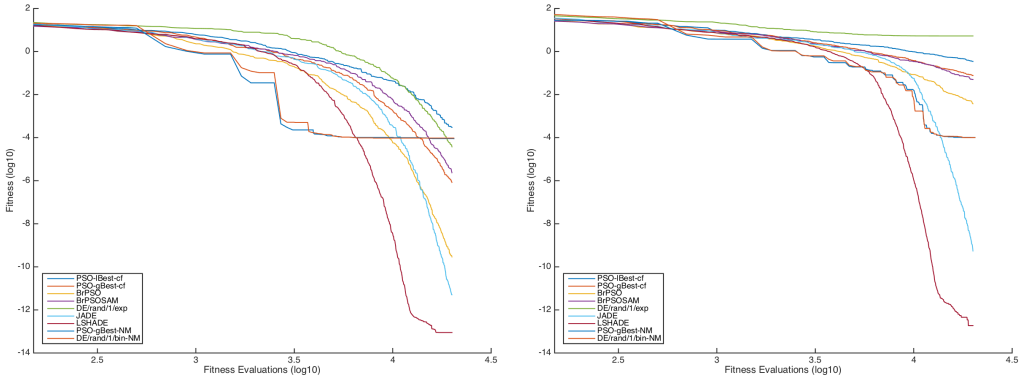
Overall it is clear that for highly accurate and reliable results L-SHADE is the best algorithm to use and the advanced DEs are significantly better for this task, but BrPSO is still highly competitive and is the best PSO method presented. The disadvantage of the DE algorithms are that they are slow to initially converge and if fitness evaluations are limited the hybrid approaches may produce quicker and still reasonably accurate parameter estimations. Based on the performance of the algorithms considered it can be inferred that the search landscape is dominantly unimodal [127]. Parameter sets 2 and 6, both proved difficult for all and most algorithms respectively, which implies that in some situations the search space may not be simply convex and could be multimodal with some prominent local minima.



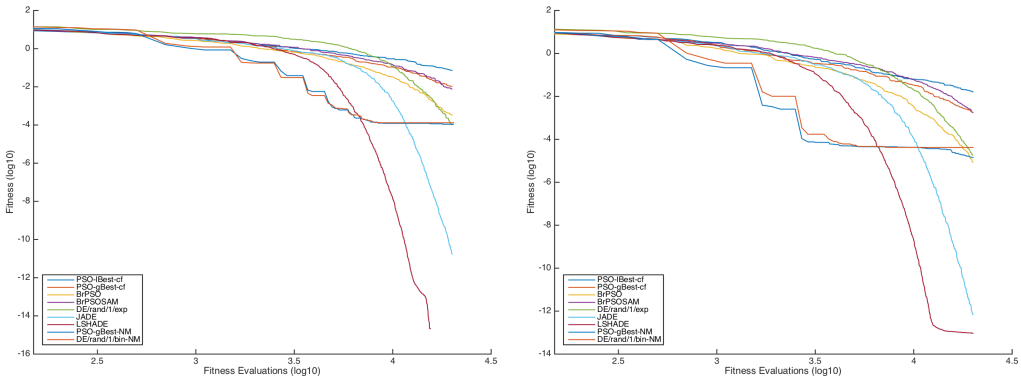
(a) Set 1 **(b)** Set 2

(c) Set 3 **(d)** Set 4

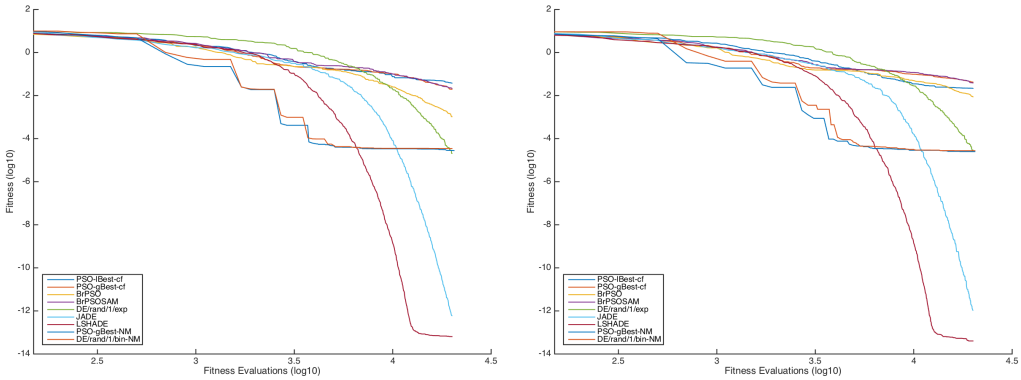
Figure 4.1: Convergence plots of the median calibration fitness for the each of the Heston parameter set experiments.



(e) Set 5 **(f)** Set 6



(g) Set 7 **(h)** Set 8



(i) Set 9 **(j)** Set 10

Figure 4.1: cont. Convergence plots of the median calibration fitness for the each of the Heston parameter set experiments.

4.5.3 Practical Parameter Estimations

The previous discussion focuses on the overall capability of the search algorithms, however in practical applications the amount of computation allowed can be limited or only a certain level of precision for the parameter estimations needs to be achieved.

Limited Fitness Evaluations

Table 4.5 show the Q75 error-norms when using less fitness evaluations. For lower fitness evaluations of 1000 from the log-mean and SLM values it can be seen that PSO-gBest and FDR actually perform the best out of all the PSO and DE algorithms, showing that PSO has better initial exploratory behaviour. The best algorithm after 1000 fitness evaluations is the PSO-gBest-NM hybrid, which is able to take advantage of the exploratory behaviour of PSO-gBest but is able to more efficiently refine the solutions via the local search.

For 1000 fitness evaluations all the DE algorithms, except the SHADE algorithms, perform extremely poorly with log-mean$\approx$ 0.5, in comparison to the PSO algorithms with log-mean $\approx$ 0.3. After 5000 fitness evaluations there is very little change in the log-means for the jDE and JADE, DE/rand/1/* algorithms, whilst most of the PSO algorithms move to negative log-means indicating that they have located the global basin and are beginning to converge.

After 5000 fitness evolutions the characteristics of the behaviour described in the previous section begin to show, with the exception of the poor DE performance, the dominance of the L-SHADE algorithms begins to become clearer with a significantly smaller log-mean of -2.1 compared to the best PSO at the time (BrSPO) with a log mean of -0.80. However, the basic hybrids are still the best for a low number of fitness evaluations and the values of L-SHADE and BrPSO do not compare to the to two basic hybrids with log-means of -3, but as was previously seen this results in premature convergence and the error-norms do not improve much comparing between 5000 and 20000 fitness evaluations.

Overall it can be seen that for smaller number of fitness evaluations PSO finds

	FE = 1000			FE = 5000		
	Mean	Log Mean	SLM	Mean	Log Mean	SLM
PSO-gB	1.504	0.186	-0.962	0.556	-0.224	0.187
PSO-gB-cf	2.302	0.361	0.207	0.370	-0.501	0.009
PSO-lB-cf	1.988	0.294	-0.235	0.436	-0.354	0.105
BrPSO	2.117	0.313	0.074	0.223	-0.813	-0.167
BrPSOSAM	2.347	0.353	0.189	0.493	-0.309	0.060
CLPSO	1.938	0.309	-0.168	1.259	0.088	0.485
CPSO	2.468	0.386	0.303	2.191	0.322	0.631
UPSO	2.249	0.366	0.298	0.468	-0.308	0.209
wFIPS	2.049	0.327	-0.041	0.870	-0.035	0.383
FDR	1.555	0.194	-0.807	0.732	-0.124	0.293
DE/r/1/b	3.464	0.531	1.207	12.540	0.300	0.946
DE/r/1/e	3.301	0.523	1.184	11.572	0.253	0.827
JADE	3.413	0.538	1.147	3.100	0.496	0.800
jDE	3.554	0.554	1.248	3.178	0.504	0.804
SHADE	1.496	0.196	-0.757	0.056	-1.279	-0.559
L-SHADE	1.524	0.193	-0.743	0.019	-2.105	-1.230
PSO-gB-NM	1.111	0.052	-1.491	0.013	-3.327	-2.017
DE/r/1/b-NM	1.607	0.173	-0.653	0.013	-3.197	-1.769

Table 4.5: Error measures (mean, log mean, and standardised log mean; standard deviations given in Table C.3) for the 75% quantile of the Euclidian distances between the parameter sets found by the EAs and the known optimal parameter set (1-10), over 30 independent runs for each algorithm. The number of fitness evaluations are limited to 1000 and 5000.

better results than DE and shows that PSO has better initial exploration. From the results using 20,000 fitness evaluations and from the convergence plots DE begins to find better solutions with better exploitation later on, after around 7500-10,000 fitness evaluations. But it is clear that the hybrids, especially the PSO-gBest-NM hybrid, are the best methods for low fitness evaluations, where the local search is able to take advantage of the exploratory power of PSO; however this does come at a cost of premature convergence.

Parameter Estimation Tolerance

In practice achieving extreme accuracy and exact values of the parameters is not of interest and is not usually possible due to market noise, in these controlled setups it is of course possible, as demonstrated by the results of L-SHADE. From practical perspective it suffices to find the parameter estimations within a certain error tolerance. Table 4.6 gives the 75% quantiles for the number of fitness valuations used to achieve parameter estimation with an error tolerance of $\|\mathbf{x}^a_{\text{opt}} - \mathbf{P}_p\|_\infty < 0.0005$,

	Fitness Evaluations									
	1	2	3	4	5	6	7	8	9	10
PSO-gB-cf	12950	n/a	12800	8200	10350	n/a	n/a	n/a	n/a	n/a
BrPSO	9238	n/a	9100	6650	7563	15300	16175	14900	15800	n/a
BrPSOSAM	14050	n/a	14150	9500	12000	n/a	n/a	n/a	n/a	n/a
DE/r/1/b	15750	n/a	15600	14038	13300	n/a	15663	13813	14525	14625
DE/r/1/e	14863	n/a	16063	14950	13525	n/a	16050	14550	14500	15425
JADE	14313	n/a	14550	13100	14975	18550	15488	14575	14650	16550
jDE	15500	n/a	15950	14075	16213	n/a	17475	14900	15050	15313
SHADE	6888	15725	6850	7300	6550	9800	7675	6700	6838	6800
L-SHADE	5500	10800	5250	5500	5138	7425	6300	5313	5150	5263
PSO-gB-NM	3522	n/a	3705	3758	2704	11712	5107	2703	3674	4656
DE/r/1/b-NM	3674	n/a	4682	4701	2902	11737	5681	2903	3702	4696

Table 4.6: 75% Quantile for the minimum number of fitness evaluations finding parameter estimations, x_i, of parameter, p_i, with absolute error $|x_i - p_i| < 0.0005$ for all five parameters, n/a indicates this level of accuracy was not achieved.

i.e. each parameter is accurate up to at least three decimal places. Table 4.6 given for a limited set of algorithms, using the best algorithms from previous discussions, where n/a indicates that the algorithm was unable to errors within the tolerance level.

All of algorithms are able to find acceptable parameter estimations after around 10-15,000 fitness evaluations, although for sets 6-10 PSO-gBest-cf and BrPSOSAM are not able to acheive this level of accuracy at all. DE/rand/1*, jDE and JADE algorithms all require around 15,000 fitness evaluations showing that these DE algorithms have slow initial exploration locating the global basin to exploit. BrPSO is better than this DE algorithms requiring only between 7-9000 fitness evolutions for sets 1,3,4 and 5. SHADE and L-SHADE show better and more consistent performance requiring around 7000 and 5000 fitness evaluations respectively. L-SHADE is also the only algorithm that is able to find acceptable parameters in the Q75 for set 2. As previously seen, the hybrid algorithms converge the fastest, and are able to find acceptable sets after around 3500-5000 fitness evaluations, PSO-gBest-NM requires the lowest number of only 2700 fitness evaluations for sets 5 and 8.

Overall from a practical standpoint the basic hybrids, in particular the PSO-gBest-NM hybrids require the least amount of fitness evaluations to acheive the same level of acceptable accuracy, but L-SHADE also shows very good and consistent performance requiring only 5000 fitness evaluations for most of the parameter

sets. The three main conclusions from the results are that the basic PSO algorithms should good initial exploration but converge slowly, the hybrid methods also show good initial convergence but quickly stagnate, whilst the DE algorithms are slow during initial exploration but eventually show good exploitation and convergence; as such an ideal EA for this application would combine all of these positive aspects, to acheive this L-SHADE hybrids are explored.

4.6 L-SHADE Hybrids

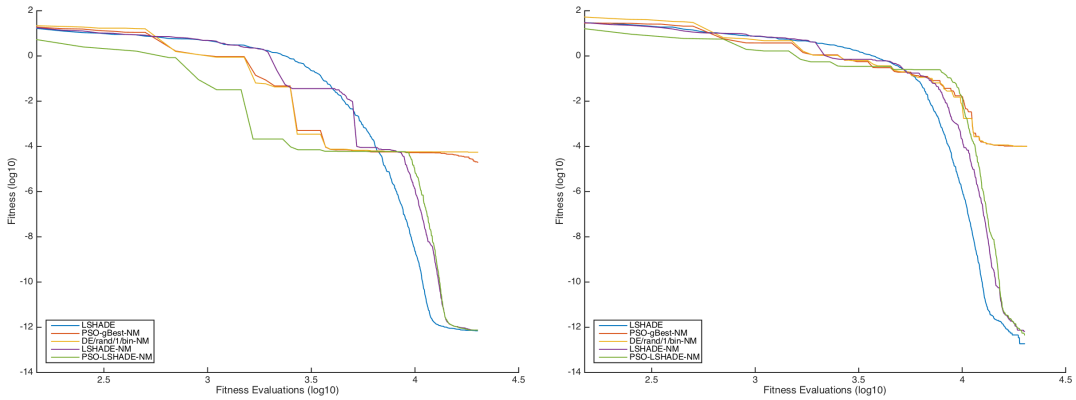
Given the success of the hybrid methods for early evaluation and the individual success of L-SHADE for overall optimisation, the same hybridisation method using the Nelder-Mead local search has been introduced into the L-SHADE algorithm. The combination of the two algorithms aims to provide the speed of initial convergence seen for the hybrid methods, but eliminating the problems of stagnation by using L-SHADE which sustains population diversity using the historical index mechanism.

4.6.1 PSO-L-SHADE

It was observed that the L-SHADE-NM hybrid showed some improvements in initial convergence with respect to the L-SHADE algorithm but was still slower than the original PSO and DE hybrids due to the larger diversity of the L-SHADE population. A second hybrid using PSO and NM is proposed, this algorithm combines all three approaches demonstrated here: PSO; DE and Nelder-Mead, and uses their observed advantages in a co-operative hybrid approach [132].

The hybrid algorithm runs with PSO-gBest-NM for the first 10 iterations to provide the initial exploration and allowing for one NM local search at the end of the PSO search. The PSO population is then used in L-SHADE-NM to complete the optimisation. This hybrid takes advantage of PSO-gBests' fast ability to locate the global basin, and then uses L-SHADE-NM to refine the solution and preserve population diversity. For L-SHADE-NM the frequency of the NM is exponentially reduced following the sequence $\mathrm{nm}_1 = 10$, $\mathrm{nm}_k = 2\mathrm{nm}_{k-1}$.

As an example, Figure 4.2 show the convergence of the two L-SHADE hybrid



(a) Set 1 **(b)** Set 6

Figure 4.2: Fitness-Distance plots for the sets of Heston parameters used in the artificial calibration.

	1	2	3	4	5	6	7	8	9	10
L-SHADE	4400	11163	4250	4550	4238	6850	5200	4338	4200	4300
PSO-gB-NM	2704	n/a	3702	3707	2703	10971	4703	2101	2703	3705
DE/r/1/b-NM	2704	n/a	3739	3702	2703	10719	4693	2704	3702	3715
L-SHADE-NM	5000	n/a	4713	5182	3900	7600	5240	4400	4600	4700
PSO-LSHD-NM	1654	n/a	4368	2480	2419	9738	2708	1654	1653	2504

Table 4.7: 75% Quantile for the minimum number of fitness evaluations finding parameter estimations, x_i, of parameter, p_i, with absolute error $|x_i - p_i| < 0.0005$, n/a indicates this level of accuracy was not achieved.

methods for parameter sets 1 and 6. Compared to regular L-SHADE as expected, the two hybrids show faster initial convergence but then begins to stagnate at the same fitness as the original PSO and DE hybrids. Although improvements are seen for just the L-SHADE-NM hybrid there initial rate of convergence still is not as fast as the original hybrids and can be seen in Table 4.7, it takes longer to find acceptable parameter estimations where $|x_i - p_i| < 0.0005$ for all parameters. When utilising the initial PSO-gBest convergence in PSO-LSHADE-NM this results in a significant reduction in the amount of fitness evaluations required; the majority of cases only requiring 2500 or less fitness evaluations. On the other hand there still exists the exploration/exploitation tradeoff and even though initial convergence has improved it can be seen in Figure 4.2 that it takes the hybrids longer than regular L-SHADE to finally converge.

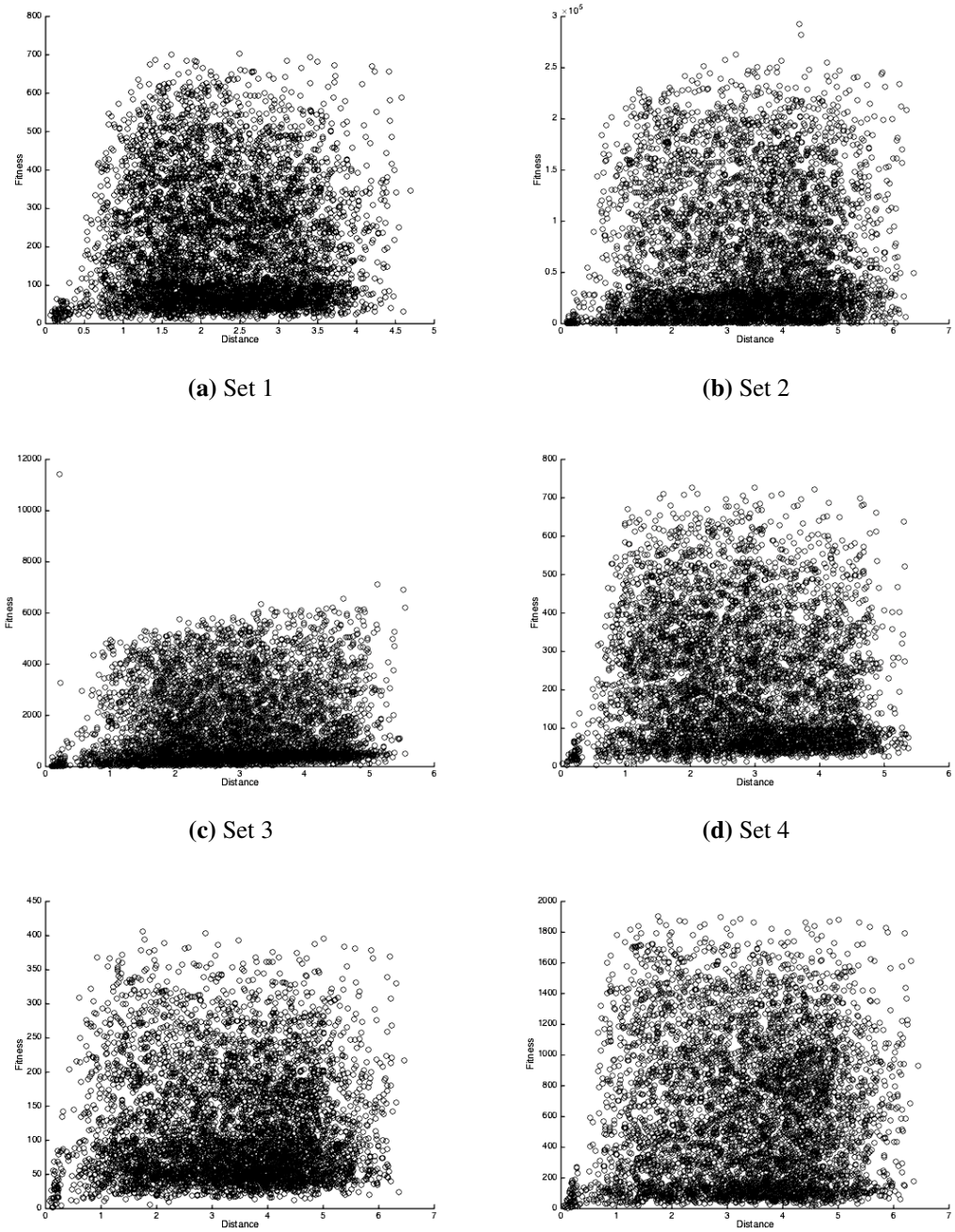
4.7 Fitness Distance Analysis

Fitness-distance analysis can be used to try and gain further insight into the characteristics of the search space landscape. Fitness-distance analysis is a simple technique and looks at the correlation between the Euclidean distance of a solution from the known global optimum and its respective fitness value. This can be used to determine global characteristics such as convexity, possible existence of local minima, and as a rough guide to determine how hard the optimisation problem can be.

The Fitness-Distance plots for each Set 1-10 is shown in Figure 4.3.Each plot has been generated from a sample of 5000 uniformly i.i.d parameter sets from the search domain, and a further set of 50 points located close to the global optimum selected from a normal distribution around the known optimal parameter set, with a standard deviation of 0.1. The fitness is calculated as the relative calibration error, Equation 4.7.

From the Fitness-Distance plots, Figure 4.3, it is not possible to accurately determine the shape of the landscape due to the large loss of information by compressing all the dimensions into a single distance value, but certain characteristic of the landscape can be inferred. Firstly it can be noticed that the search space is not obviously convex, a strictly convex landscape would show a distinct linear relationship between the distance and fitness [168]. There is some evidence of global convexity by looking at the lower bounds of the fitness as the distance increases which does show a subtle linear relation, this is particularly prominent for sets 9 & 10. It can also be seen that the landscape does become more convex close to the optimum, this suggests that there is a very narrow and steep global basin.

There is also a concentration a fitness values present along a narrow range of fitness values, this suggests that a long valley-like structure exists and based on the surrounding but sparsely populated region of higher fitness also suggest that the global search falls into a deep narrow valley but with a relatively shallow gradient along inside. But from these plots it is not possible to determine if these valleys lead to the global optimum or are separated by barriers of surrounding high fitness solutions, thus making these areas local optimum. There are some cases for Set 7



(a) Set 1

(b) Set 2

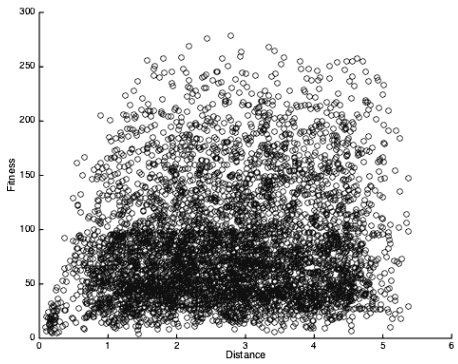
(c) Set 3

(d) Set 4

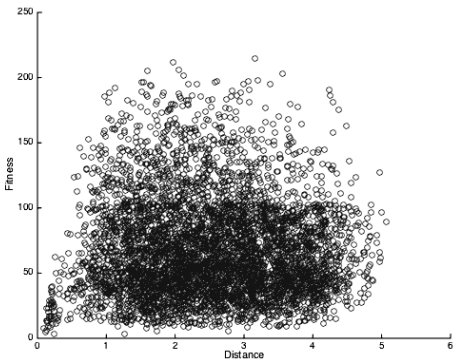
(e) Set 5

(f) Set 6

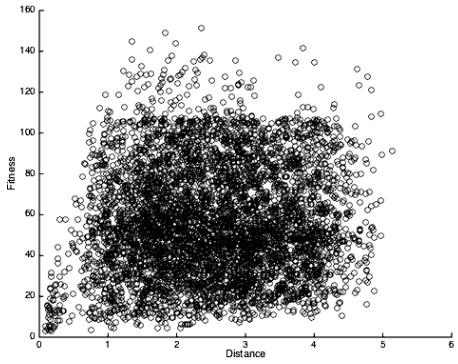
Figure 4.3: Fitness-Distance plots for the sets of Heston parameters used in the artificial calibration. The distance is the distance of the parameter set from the known optimal parameter set.



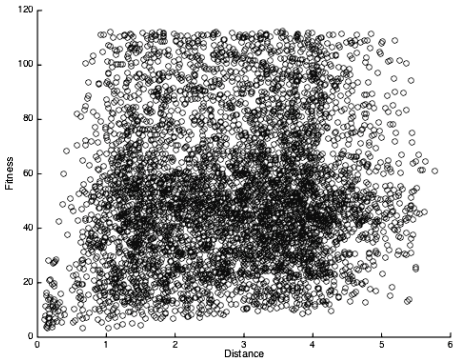
(g) Set 7



(h) Set 8



(i) Set 9



(j) Set 10

Figure 4.3: cont. Fitness-Distance plots for the sets of Heston parameters used in the artificial calibration.

that suggest deep local optima given that there are lone low value fitness values at respectively far distances from the global optimum.

What is also interesting to note is that given the different calibration surfaces for each of the sets, is the how large the relative pricing error can range between different sets, although the same general shape prevails for all the sets it seems the gradient of the valleys can can greatly vary.

This analysis further supports the need to use heuristic algorithms with global exploration abilities in order to efficiently locate the small basin of attraction around the unique solution, whilst other minimisation algorithms (NM, LM etc.) may fall into either the wrong valley, or if started too far away from the global optimum converge too soon in the valley far from the global basin.

4.8 Local Minima and Numerical Instability

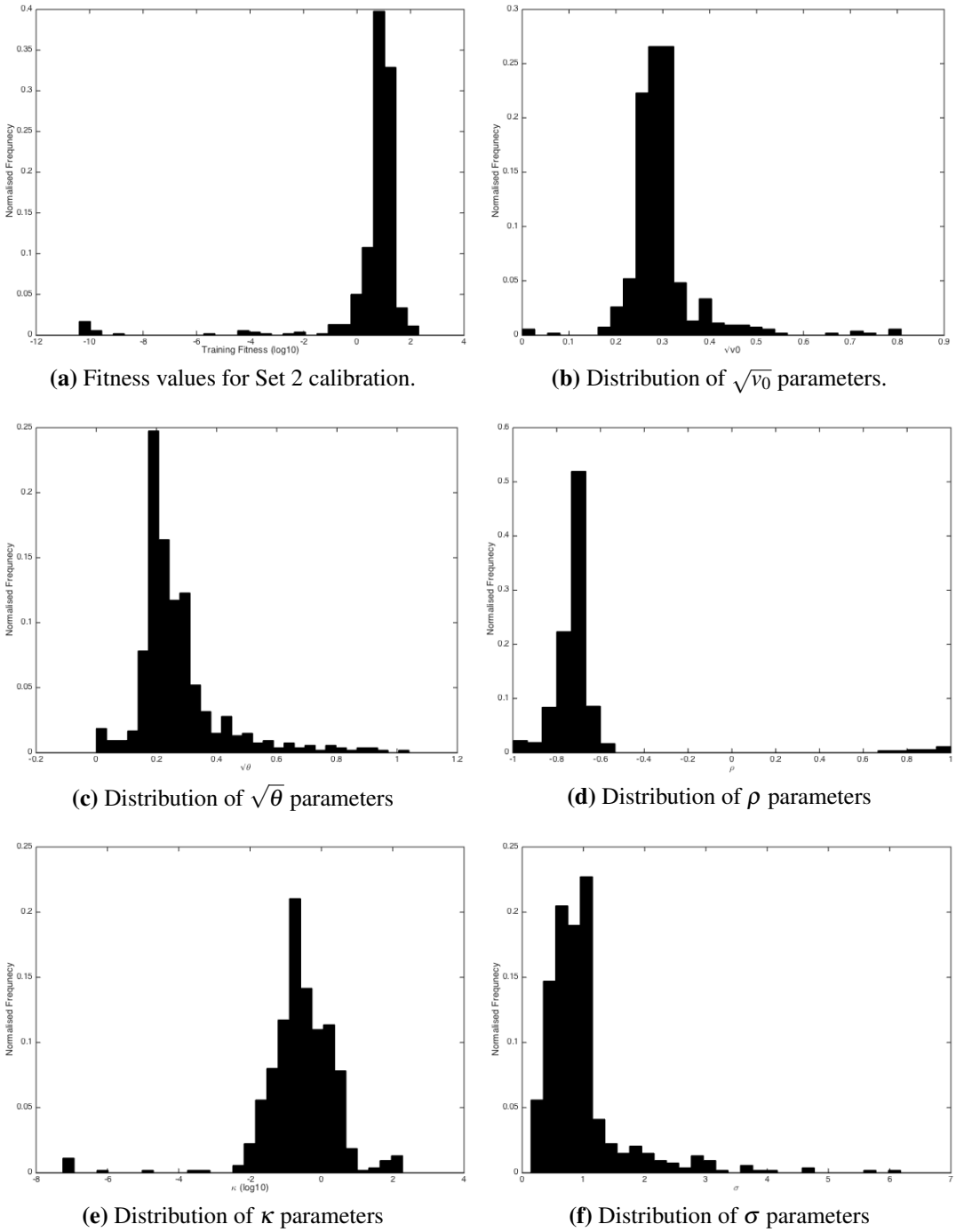
This section investigates how the Heston characteristic function and numerical integration can contribute towards creating local minima in the parameter search space, this effect is most prominent for parameter set 2. From the results in Table 4.3 the error norms for set 2 standout, the large error-norms indicate that for some reason the calibration for this parameter set proves to be extremely difficult with deep local minima. However, it is seen later on that the existence of such local minima is an artefact of the numerical integration scheme used which can result in the calibration problem becoming ill posed in some circumstances; precautionary measures are then introduced to reduce the impact of this behaviour.

Even for the L-SHADE optimisation, the error-norms are extremely large and suggests that there may be significant local minima, furthermore L-SHADE appears to be the only algorithm that was able at all to successfully estimate the parameters exactly, but with less than a 50% success rate. From the convergence plot for set 2, Figure 4.1.b, it shows that this local minimum is generally found after around 3-5000 fitness evaluations which suggests that this local minima occurs near or within the global basin.

Figures 4.4 show the distribution of each of the fitness function value and five parameters for set 2 found in all 30 of the calibration runs for all the algorithms after the maximum 20,000 fitness evaluations. The known optimal parameters for set 2 from Table 4.2 are $\sqrt{v_0} = 0.3, \sqrt{\theta} = 0.3, \rho = -0.7, \kappa = 0.2, \sigma = 1$.

Firstly, it is most noticeable that a high majority of the calibrations result in a negative fitness, this is due to negative prices occurring in the denominator of the fitness function which are due to numerical instabilities in the integration. The negative fitness functions exist as part of poor experimental design, the calibration surface is priced relative to the artificial calibration surface which can exhibit negative prices to numerical instabilities in the integration this therefore allows for negative contributions to the fitness function to build up.

This has a profound effect on the calibration parameters found which can be seen from the extremely large range of magnitudes of the parameter values, es-



(a) Fitness values for Set 2 calibration.

(b) Distribution of $\sqrt{v_0}$ parameters.

(c) Distribution of $\sqrt{\theta}$ parameters

(d) Distribution of ρ parameters

(e) Distribution of κ parameters

(f) Distribution of σ parameters

Figure 4.4: Distribution of global optimum fitness values (a), and Heston parameters (b-f), found for all the calibration experiments using parameter set 2.

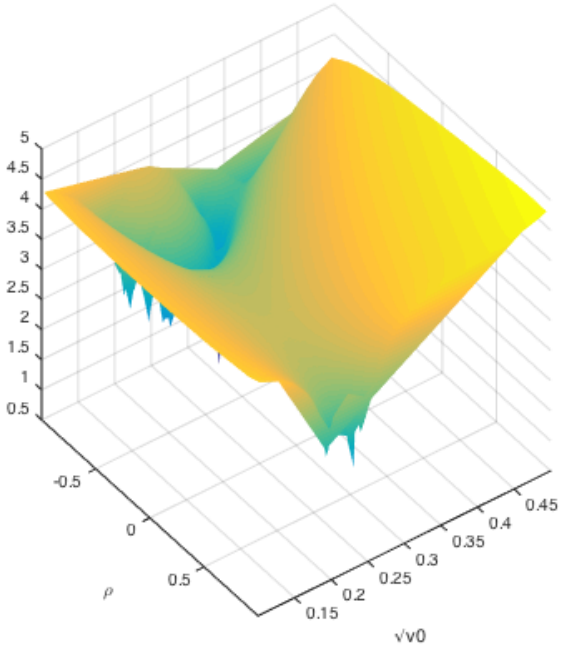
pecially for κ and σ, which exceed any reasonable estimations, similar effects of numerical integration on the stability of κ estimations has also be observed in [191], although not to the same degree as observed here.

Both v_0 and θ are distributed around the correct values, although v_0 is slightly more bias towards a lower value of ≈ 0.25. For ρ compared to the optimal parameter set there is a higher tendency towards $\rho \approx -1$ than the correct the value of 0.7, this shows that there is preferential behaviour towards this value and could be characteristic of the numerical instability. To investigate the cause of the local minima the effect of the Heston parameters on the search space landscape are empirically explored.

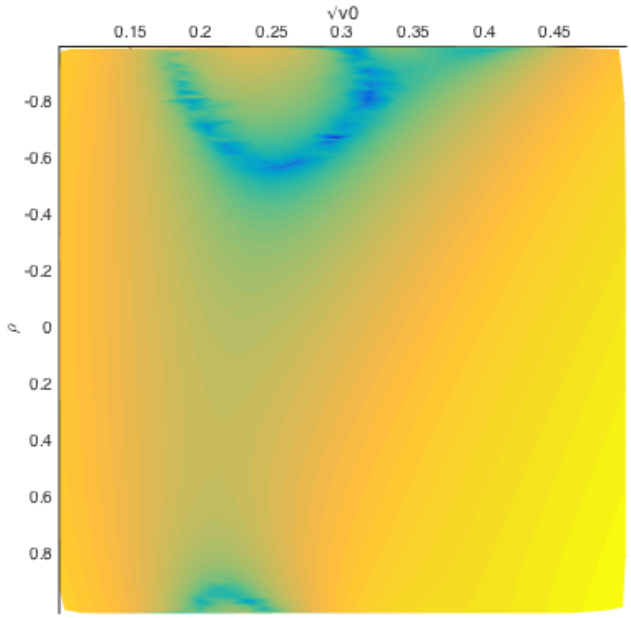
Empirical Exploration

The effect of local minima for set 2 has been empirically explored using a random search to identify search space features that can cause it to occur. The occurrence of local minima has been identified to occur with respect to the parameters $\sqrt{v_0}$ and ρ. Figures 4.8 and 4.6 shows the calibration fitness function for $\sqrt{v_0}$ and ρ whilst all other parameters in set 2 are fixed to their known optimal values. It can be seen that there is a distinct double-valley structure formed, the smaller local minima valley is surrounded by high edges, and both the global and coal optimum valleys wrap around to the basin centred at $\sqrt{v_0} \approx 0.25$ and $\rho \approx -0.55$. It is seen in 4.4.d that there are some instances of $\rho > 0.8$ occurring, for this region it is shown in Figure 4.8 to also create an area of local minima, it can be hypothesised from this that part of the cause of local minima is symmetric with respect to large absolute values of ρ. When the other parameters are paired with $\sqrt{v_0}$ (whilst the remaining parameters remained fixed at their known optimal value) it was found that the double valley structure existed along the whole range of the the second parameter; this is further evidence that the double valley only occurs due to interactions between $\sqrt{v_0}$ and ρ.

It can also be seen why the existence of this type of local minima did not effect the optimisations of sets 4 and 5, which have very similar parameter settings, the optimum $\rho = -0.5$ lies outside of the deep valley and double valley that occurs for larger negative correlation values.

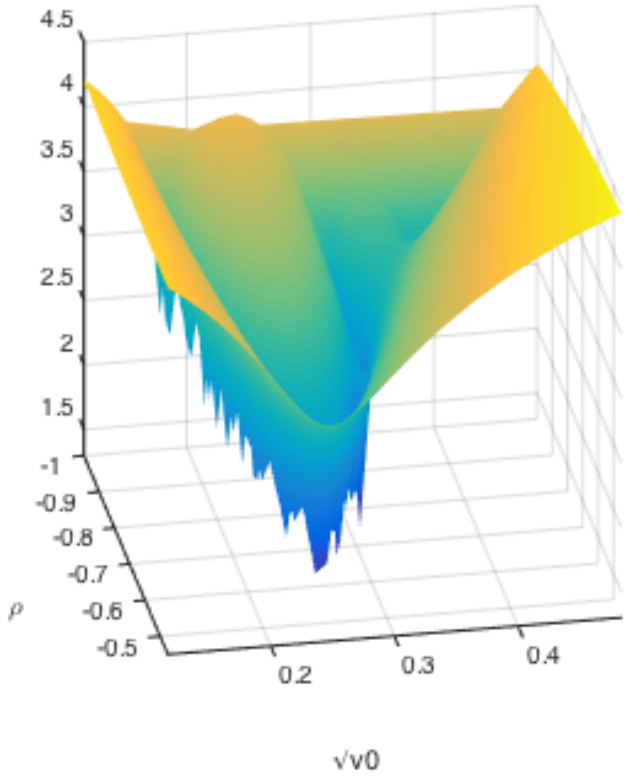


(a)

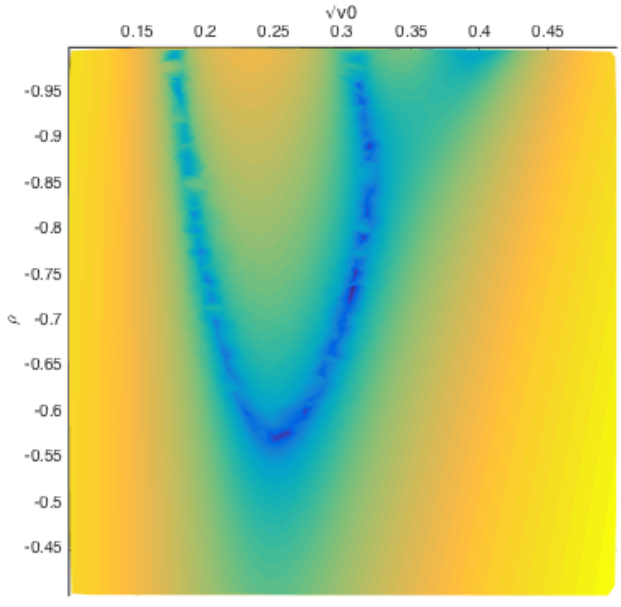


(b)

Figure 4.5: Surface and contour plot showing the interactions between $\sqrt{v_0}$ and ρ creating a double valley structure and regions of local minima.

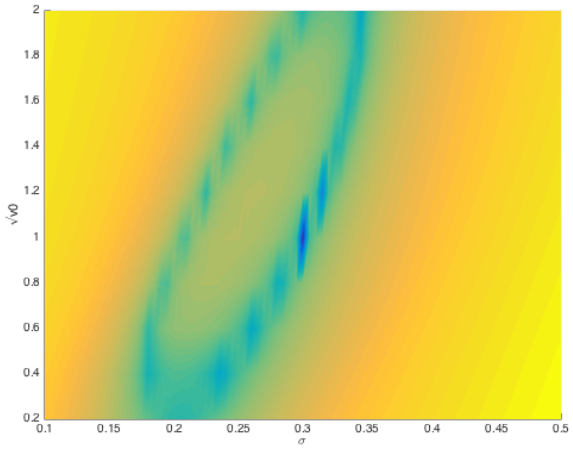


(a)

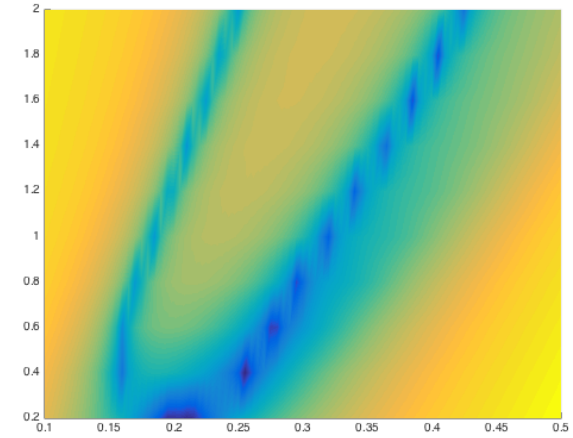


(b)

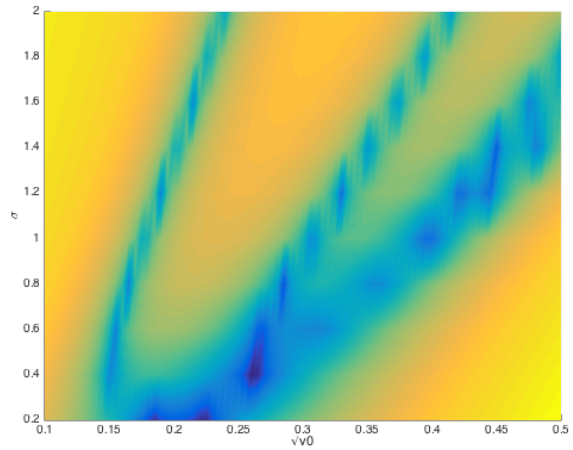
Figure 4.6: Surface and contour plot showing the interactions between $\sqrt{v_0}$ and ρ creating a double valley structure, showing in more detail for negative values of ρ.



(a) $\rho = -0.7$



(b) $\rho = -0.9$



(c) $\rho = -1$

Figure 4.7: Contour plots showing the interactions between $\sqrt{v_0}$ and σ and a fixed ρ and how the value of σ.

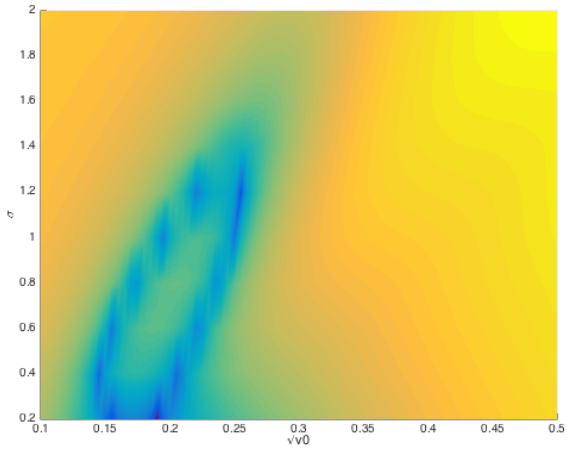


Figure 4.8: Contour plot showing the interactions between $\sqrt{v_0}$ and σ for a large positive correlation, $\rho = 1$.

The value of v_0 found by algorithms has the highest correlation with the σ parameter. Figures a-c shows how the the shape and location of the local minimum change with σ for three different correlation values, $\rho = -0.7, -0.9, -1$, in the region where local minima form.

For when $\rho = -0.7$, which is the known optimal value for set 2, the location of the global optimum can be seen (the of which was magnitude was truncated to allow for the other local minima to be more visible). It can be seen here that the the global optimum is a very small and narrow basin, which makes it hard to locate. Analogies of this search space can be seen with the deceptive function, where a small global basin is separated from the larger shallower local basin by a steep feature, even in one dimension EAs, especially differential evolution struggled to locate the global basin when trapped in the local basin. This suggests that if the EA find itself in one of the valleys of local minima it is unlikely for it to escape and find the small global basin.

As the magnitude of the correlation parameter, ρ, increases the local minima becomes a more prominent feature, and it can be seen in the extreme of $\rho = -1$ that a third valley of another local minima forms for high values of $\sqrt{v_0}$. The multiple valleys begin to show for large σ, and for all ρ it appears that only a single valley forms for low $\sigma \leq 0.2$, this observation also correlates with the violation of the

Feller condition which can lead to instabilities for large σ.

Figure 4.9 further confirms that this feature is the cause of the local minima observed in the results here, it can be seen that the correlations observed in Figures a-c between σ, ρ and $\sqrt{v_0}$ are present in the calibration result data. High values of $\sqrt{v_0}$ are found to occur along with large σ and high magnitudes of negative correlation. For the few instances of high magnitude positive correlations being found, these values of ρ correspond to values of $\sqrt{v_0} \approx 0.2$ and similarly small $\sigma \approx 0.2$, along with large values of $\kappa \approx 3$ (not shown here but can be seen in Table C.4); these found parameter sets can be observed to satisfy the Feller condition. It can be seen in Figure 4.8 that as seen for negative ρ a similar double valley local minima structure forms for $\sigma < 1.5$.

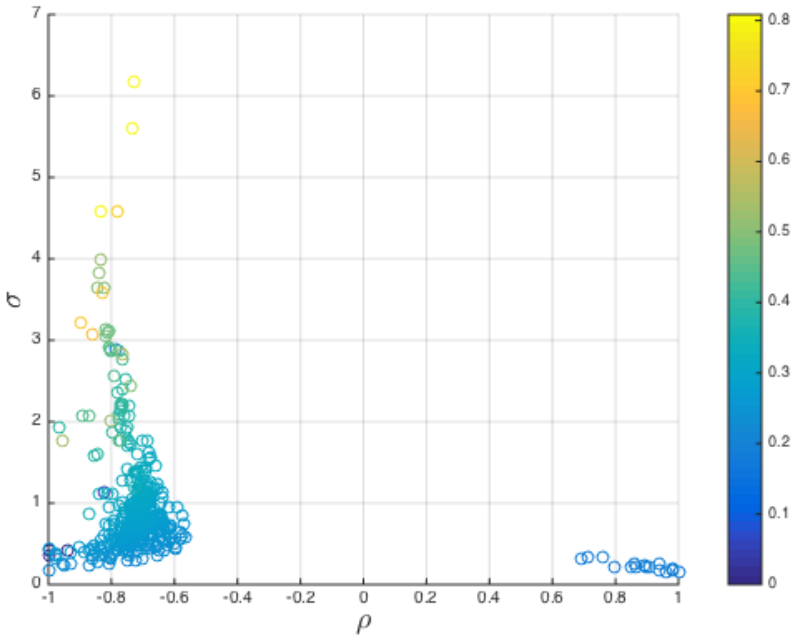


Figure 4.9: Scatter plot of the found optimal parameter sets for $\sqrt{v_0}$, σ and ρ, the value of $\sqrt{v_0}$ is given by the depth using the colour bar.

Numerical Instability

The results of Cui *at al* [127] quite firmly suggest that the Heston search landscape is unimodal and the results are based on a form of the analytical gradient. This raises the question as to whether to errors in the numerical approach used here, it

is well known that instability can arise in the numerical integration of the Heston characteristic function. It was observed that GL integration with 10 and 32 steps are used the double value feature disappears, which shows that it is an artefact of the GL-16 numerical integration used here. Given that the feature disappears for both a lower and higher number of steps this indicates that this is not due an issue with overall precision of the integration, but to do with a more general behaviour of the integral that is maximised for GL steps of around 16.

The reason the double valley structure is observed here for very specific number of GL steps for parameter 2 is a scenario of chance given the optimal parameter set, grid of options used and spacing of the integration points, which happen to all interact in a very particular way to cause the observed local minima; but this is an important observation as it highlights possible random sources of error and risk when using such option pricing models.

Figures 4.10 shows the value of the components, A, B and C of the characteristic function (CF), Equation 4.9, along the integration domain ω for the sequence of integration points used in GL-16. It can be seen that there is inherent oscillatory behaviour due to component A which is independent of the Heston model parameters, instability in the integral can therefore occur due to two behaviours:

1. How the GL sequence matches the frequency of the oscillation of component A. If the sequence of GL integration points matches the frequency of the oscillation in A then a large magnitude of instability will be exhibited in the integration. The degree of the instability can be further amplified by the integration weights. If large amounts of instability occur near the end of the integration domain then the larger integration weights will further amplify this effect.

2. How well the components B and C can dampen the oscillation of component A. If the components of B and C are well behaved then the function will monotonically decrease to 0, this therefore acts to dampen the oscillation of component A. This can also be related to the sequence of GL integration points, such that if oscillating values occur for A then they should correspond

to high damping regions of B and C and reduce this instability.

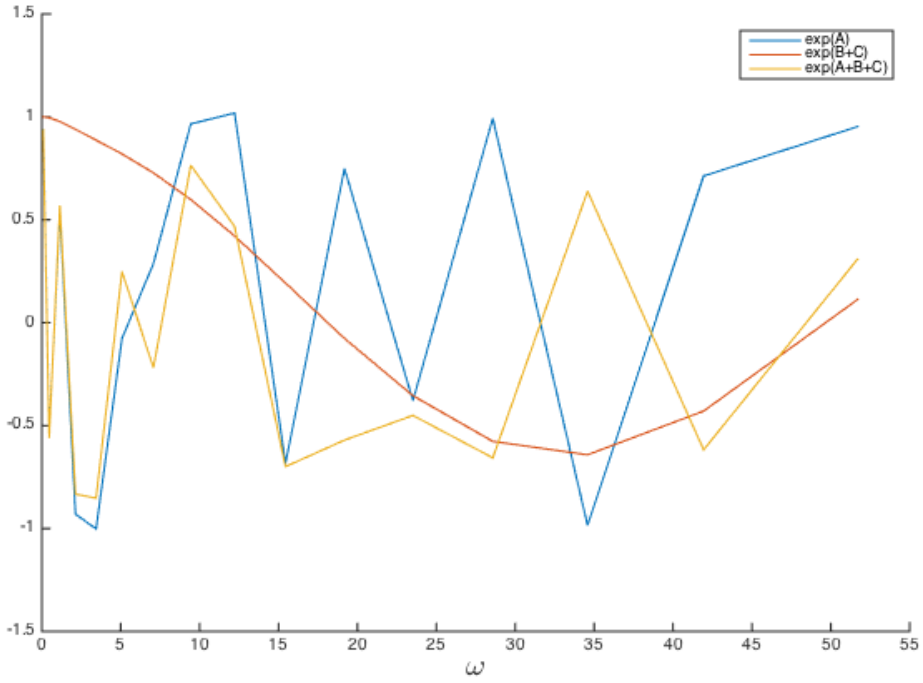
Figure 4.10.a shows the CF components for an observed unstable region where local minima occur for high magnitudes of negative correlation, this is compared to Figure 4.10.b which shows the CF components for a more stable region with a low degree of correlation. The main difference is the decay of the B and C component, for $\rho = -0.5$ this decays monotonically for $1 \rightarrow 0$, however for $\rho = -0.9$ the components form a negative well and over the range of integration does not decay. When combined with the A component it can be this that this leads to larger magnitude instabilities for $\rho = -0.9$. It is also worth noting that for GL-16 it so happens that the minimum of the negative well formed by B and C aligns very strongly with the integral region of highest oscillation for component A, which also occurs near the end of the region of integration with larger integration weights, this all accumulates to creating a large amount of instability. It is seen that the behaviour of the B and C components play a major part in creating the local minima due to the negative well.

Integral Oscillations

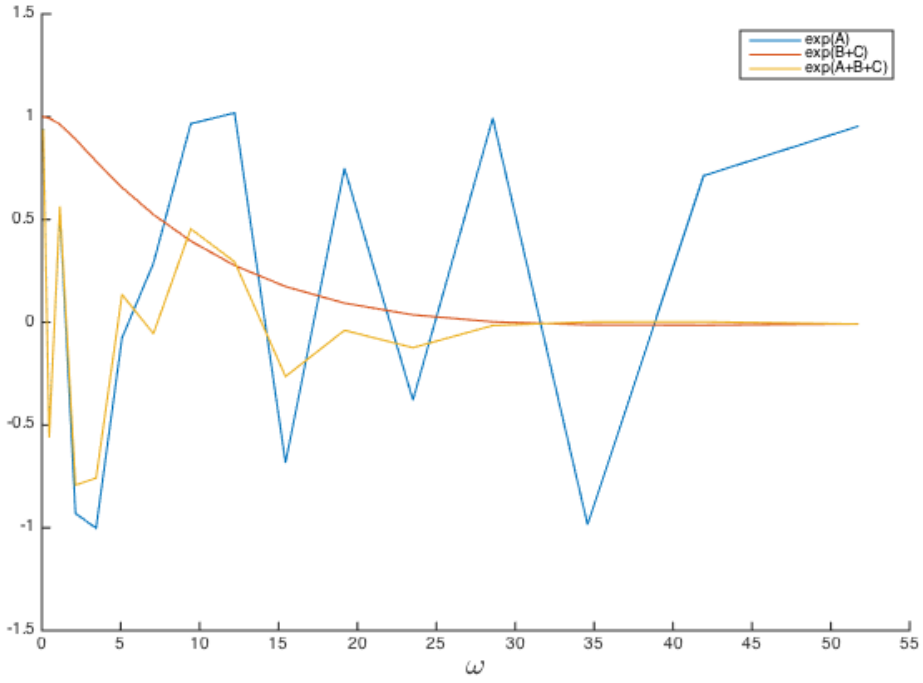
It is known that the Heston integral is not always well-behaved and can exhibit oscillatory behaviour. The main cause of this behaviour is often attributed to the terms d and g in the Heston characteristic function, and how negative real values cause branch splitting in the square-root and logarithms applied to these components. It was shown by Albrecher et al [123] that using the forms given in Equations 4.9 result in a stable integral for all parameter sets, as d and g do not cross the negative real-axis. However, upon inspection, the behaviour observed in the calibrations here for parameter set 2 arise from a simpler term occurring in both the B and C components,

$$\kappa - \rho\sigma i\omega - d. \tag{4.15}$$

The effect of the B and C components is that they create a monotonically decreasing function when well behaved, but can create a negative well which leads to numerical integration problems as previously shown. The negative well occurs when the



(a) $\rho = -0.9$



(b) $\rho = -0.5$

Figure 4.10: Surface and contour plot showing the interactions between $\sqrt{v_0}$ and ρ creating a double valley structure, showing in more detail for negative values of ρ.

imaginary argument of Equation 4.15 becomes too large, and thus when taking the exponential results in an oscillating function over the integration domain.

The degree of the effect of this oscillatory component on the integral can be related to the sequence of points used for numerical integration, and more so when the Feller condition is violated. This can be demonstrated for the conditions for which the numerical instability is observed here. First of all it is useful to expand d s.t.

$$d = \sqrt{\kappa^2 + \sigma^2\omega^2 - \rho^2\sigma^2\omega^2 + i(\sigma^2\omega - 2\rho\sigma\omega\kappa)} \tag{4.16}$$

It will now be assumed that the Feller condition is violated such that $\kappa << \sigma$, and secondly it is assumed that the degree of correlation sufficiently large, $|\rho| > 0.8$, under these conditions the terms involving κ can be dropped and the approximation can be made

$$\begin{aligned} d \approx \tilde{d} &= \sqrt{\sigma^2\omega^2 - \rho^2\sigma^2\omega^2 + i\sigma^2\omega} \\ &= \sigma\sqrt{\omega^2(1-\rho^2) + i\omega}. \end{aligned} \tag{4.17}$$

Secondly, under the assumption $\kappa << \sigma$, between components B and C, C becomes the dominant term. Considering the case where $|\rho| = 1$ and $\kappa = 0$ it can be seen that the approximation $\tilde{d}$ can be further reduced to

$$\tilde{d} = \sigma\sqrt{i\omega} \tag{4.18}$$

therefore

$$\kappa - \rho\sigma i\omega - d \approx \sigma i\omega - \sigma\sqrt{i\omega}. \tag{4.19}$$

Finally, C is the dominant term giving the approximation

$$\tilde{C} = v_0\frac{i\omega - \sqrt{i\omega}}{\sigma}. \tag{4.20}$$

Taking the exponential of this approximation and using the Euler identity it can be seen that the real component oscillates with respect to the function

$$\mathrm{Re}\left(e^{C}\right) \approx a(\sigma,\omega)\cos\left(v_0\frac{\omega-0.7\sqrt{\omega}}{\sigma}\right) \tag{4.21}$$

where $a(\sigma,\omega,v_0)$ is some function determining the amplitude. The important part of this approximation is to observe the frequency of this oscillation and that when $v_0\frac{\omega-0.7\sqrt{\omega}}{\sigma} > \frac{\pi}{2}$ the function becomes negative which introduces instability into the integral. Figure 4.11 shows that this approximation is able to suitably predict the region of the integration where negative well and instability occurs, when a small value of $\kappa = 0.2$ is used it can be seen that it shifts the location of minimum to the slightly to the left, but the approximation is still able to explain the observed behaviour. This approximation also shows why there is an observed dependance between the v_0 and σ parameters found during the calibration. The cosine function further explains why a similar double valley structure is also seen for when ρ is found to be positive.

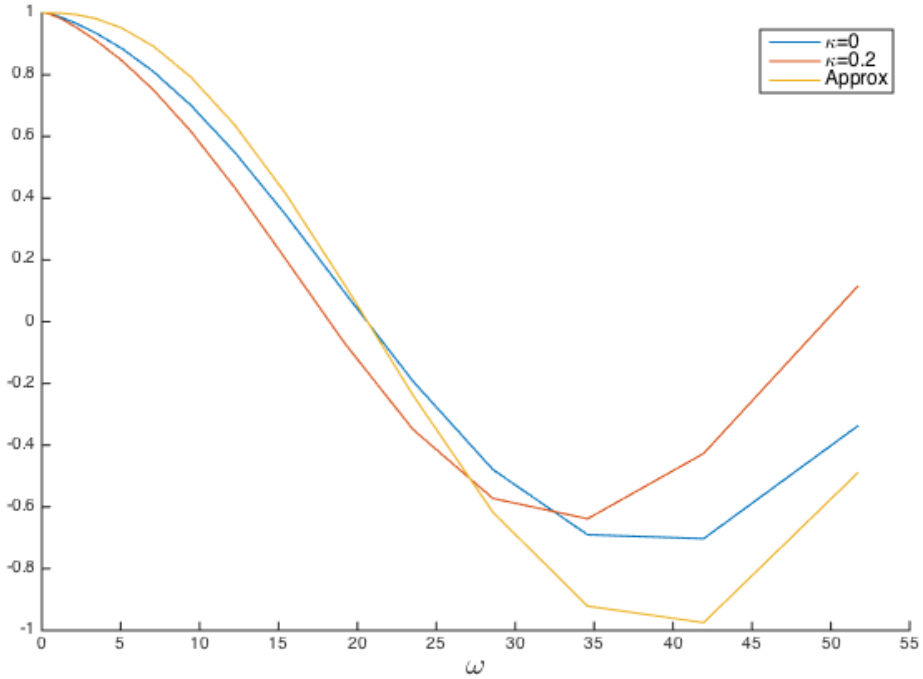


Figure 4.11: Scatter plot of the found optimal parameter sets for $\sqrt{v_0}$, σ and ρ, the value of $\sqrt{v_0}$ is given by the depth using the colour bar.

It has been observed that numerical instability can occur as a result of integration and oscillations. Given the range of parameter sets explored by the EAs the results here do not provide any sufficient conditions to stop oscillations occurring at all, but steps can be taken to try and reduce oscillations.

Reducing Oscillations

A relation can now be seen with respect to how the range of integration can have an effect on the stability of the integral. From the approximation, Equation 4.21, it can be deduced that for the components of B and C to remain positive then

$$v_0 \frac{\omega^* - 0.7\sqrt{\omega^*}}{\sigma} < \frac{\pi}{2} \tag{4.22}$$

where $\omega^* = \max(\omega)$, the maximum value of the range of integration. It can also be seen that extremely large σ can stabilise the integrand by reducing the frequency of the oscillation, this effect is seen in Figure 4.8.a. When $|\rho| < 1$ the system becomes a damped oscillator and it can be seen in the case of GL-32 that the B and C component stabilises so after the negative well, compared to GL-16 in GL-32 the region of the negative well is also lower weighted in the integration causing it to have a lesser effect.

However, it is out of the scope of this work to fully investigate the effects numerical integration can have on calibration and is left to be more throughly explored as further work.

Price Homogeneity and Damping

One simple adaptation that can be implemented to reduce oscillations in the integrand is to use normalised prices, using price homogeneity the prices can be expressed in terms of spot price as $S = 1$

$$C(S,K) = SC(1, \frac{S}{K}). \tag{4.23}$$

Considering the first component of the integrand, e^A, it can be seen that this eliminates any oscillations in the integrand due to spot price reducing the first term of A

to

$$Re[e^{\mathrm{i}\ln(S)}] > Re[e^{\mathrm{i}\ln(1)}] = 1. \tag{4.24}$$

This results in a more well-behaved integrand, this is particularly effective for short time-to-maturity options as the real part $\cos(\omega(r-q)\tau)$ will have a respectively low frequency. The factor $\mathrm{i}\omega(r-q)\tau)$ now becomes the main oscillating component, and can result in high frequencies if τ becomes too large. It is out of the scope of this work, but it would be interesting to look at methods of manipulating r, either by αr or $r+\alpha$ such that it reduces the frequency of the oscillating factor $\mathrm{i}\omega(r-q)\tau)$, this would then involve a correction term with respect to α to be applied to the integrand or option price afterwards. This idea is similar to the damping factors that have been introduced by Carr and Madden [121]. Oscillations can also occur for the B and C components of the integrand, the Carr-Madden damping tries to reduce the overall oscillations by solving $g(-\mathrm{i}\alpha)e^{d(-i\alpha)\tau)} = 1$. Again though, this does not 100% eliminates oscillations and only damps the effects, furthermore due to the dynamic nature of the Heston model parameters in heuristic searches a single optimal α may not be possible for all parameter sets explored, and extra computation would be required to compute α for each new parameter set.

4.8.1 Calibration Stability Measure

Instability in the integral can be reflected by negative prices occurring. To measure the level of instability and robustness of the calibrations the ratio of total number of negative prices occurring during all price evaluations within all the fitness function evaluations is examined.

It is first of all interesting to note that for parameter sets 2 and 6 the target calibration surface contains 1 and 2 instances of negative prices respectively, however a perfect calibration should still be able to match all the prices including the negative prices given that the price evaluation calculations (GL-16) are the same for both generating the target calibration surface and the fitness function evaluations; though this already does hint that there may be some inherent instability in these problems.

All other parameter sets have no negative prices occurring in the target surface.

To adjust for the number of negative prices inherent to the problem this is subtracted from the negative price count for each fitness function evaluation to give an adjusted scale of negative-mispricings, the absolute value of the adjusted negative price count for each price count is then taken. The adjusted negative-mispricing ratio, r, is more formally given as

$$r = \frac{1}{N_{\mathbf{C}}\text{MaxFE}} \sum_{i=1}^{\text{MaxFE}} |M_i - I_p|, \tag{4.25}$$

where M_i is the negative price count, I_p is the problem inherent negative price count, $N_{\mathbf{C}}$ is the number of price evaluations for the calibration surface (which for the experiments presented here $N_{\mathbf{C}} = 147$), and MaxFE is the number of fitness evaluations evaluated per calibration run.

Table 4.8 gives the average ratio of negative-mispricings, Equation 4.25, from the 30 independent calibrations for each parameter set using the L-SHADE algorithm. It can be seen that for the stable parameter sets the average ratio is an order of magnitude of around 10^{-4}-10^{-5}, whilst parameter sets 2 and 6 show ratios of an order of magnitude of 10^{-3}. Parameter set 2 shows the highest ratio, as may be expected, and is twice as large as the ratio for parameter 6. This shows that there is a significant amount of instability occurring in the prices during the calibration of parameter set 2, which corresponds to the observed high rate of calibration failure. However it is interesting that although parameter set 6 shows an inherently higher number of negative prices in the target surface and a larger ratio compared to other stable parameter sets it still exhibited stable behaviour with respect to accurately calibrating the parameters; although it should be noted that other algorithms did struggle significantly more compared to L-SHADE when calibrating this parameter set and sometimes did fail.

Overall the ratio of adjusted negative-mispricings gives a good indication with respect to the stability and robustness of the calibration results, and a larger ratio indicates a higher measure of instability. Methods can be applied in an attempt to

	1	2	3	4	5
Mean	4.73e-04	3.14e-03	9.02e-04	4.13e-04	1.88e-04
Std	7.90e-05	6.33e-04	1.72e-04	7.91e-05	3.93e-05
	6	7	8	9	10
Mean	1.50e-03	8.28e-05	6.36e-05	3.96e-05	3.38e-05
Std	2.15e-04	2.68e-05	2.46e-05	1.49e-05	1.50e-05

Table 4.8: The average adjusted negative-mispricing ratio, Equation 4.25, for each of the calibration parameter sets using the L-SHADE algorithm.

reduce the oscillations and instability of the integrand but there is no guarantee over the parameter search space this will hold, as such a simple safeguard built into the fitness function is recommended.

4.8.2 Simple Safeguard

A simple safeguard to implement to protect against these anomalous situations is to use a filter for negative prices in the fitness function. This has the major advantage of using heuristic optimisation methods such as EAs as they allow flexibility in defining the fitness function which does not have to be continuous or differentiable. The simple solution can be calculated using binary weights $w_{t,K}$ to indicate if the calculated price is negative (unstable) or not:

$$\text{fitness}(\mathbf{x}) = \frac{1}{\sum_{t,K} w_{t,K}} \sum_{t,K} w_{t,K} \frac{|\hat{C}_{t,K}(\mathbf{x}) - C_{t,K}(\mathbf{P})|}{C_{t,K}(\mathbf{P})} \; ; \tag{4.26}$$

$$w_{t,K} = \begin{cases} 1 & \text{if } \hat{C}_{t,K}(\mathbf{x}) \geq 0 \\ 0, & \text{otherwise} \end{cases} .$$

This has been used for the calibration of parameter set 2 using L-SHADE, and results are shown in Table 4.9 for the parameter values found averaged over 30 independent runs. It can be seen that despite the numerical instability the optimisation is now able to accurately proceed when using the safeguard. The mean values especially for κ and $\sqrt{\theta}$ have been significantly improved, where without the safeguard the mean values found are largely incorrect which is due to high variance over all of the runs, whilst when using the safeguard they are now correct up to 4 decimal places with extremely low variance. For all five of the Heston parameters the val-

	$\sqrt{v_0}$	$\sqrt{\theta}$	ρ	κ	σ
No Safeguard					
Mean	0.29856	0.43075	-0.70155	0.13462	0.98249
Std	0.00351	0.20664	0.00951	0.07807	0.05274
Safeguard					
Mean	0.30002	0.30064	-0.70007	0.19916	1.00053
Std	0.00001	0.00017	0.00002	0.00022	0.00014

Table 4.9: Comparing the mean and standard deviation of the parameter values found for the calibration of parameter set 2 using the LSHADE algorithm with and without the safeguard, Equation 4.26, used in the fitness function.

ues found using the safeguard are a lot more stable, with extremely low variance, and more accurate, showing that this simple procedure can be highly effective in improving robustness.

4.9 Conclusions

This chapter has looked at the use of advance variations of evolutionary algorithms for use in Heston model calibration. It is seen that out of the PSO algorithms considered here the BrPSO algorithm, developed in Chapter 3, finds significantly better model parameters. However, in comparison to advanced differential evolution algorithms it is clear that the LSHADE variants produced the overall most robust and accurate parameter estimations.

It was observed that the simple PSO-gbest algorithm performed surprisingly well for low numbers of fitness evaluations, showing that PSO had strong initial exploration capabilities, but lacked efficient exploitation. The converse was seen for the DE algorithms, and as such a PSO-LSHADE hybrid was proposed. This hybrid algorithm takes advantage of PSO-gbest initial exploration to then provide a population to be refined by the LSHADE algorithm, furthermore this was then hybridised with a local Nelder-Mead (NM) local search, to give the PSO-LSHADE-NM algorithm. It was seen that this hybrid algorithm was able effectively take advantage of the strengths of the three algorithms used in the hybrid, the hybrid algorithm showed good initial exploration capabilities as well as fast exploitation but managing to retain population diversity and not converging early.

An interesting observation was made for the calibration experiments for parameter set 2, and it was found that all the algorithms struggle to find the known optimal parameter set. It was found that prominent local minima, in the form of a double valley like structure, was created as a result of numerical instability resulting from the numerical integration. This effect was caused due to a combination of parameter values and the numerical integration scheme used which was Gauss-Legendre using 16 steps (GL-16). A brief analysis showed that numerical instabilities occurred in the integral due to regions of negative real values occurring in the integral terms B and C in Equation 4.9, and the choice of the numerical integration points further amplified this effect. This is an important observation as it highlights the risks of using numerical methods for calibration and how random sources of error can significantly impact results. However, the use of a simple safeguard in the fitness function, Equation 4.26, was able to significantly reduce errors in the calibration results and improve robustness.

Overall this work shows that current use of EAs in the model calibration literature undervalues the power of these algorithms by only considering the basic variations. Using more advanced variants and hybrids the calibration results in terms of efficiency and accuracy can be significantly improved, and shows potential for using these algorithms in more complex model calibration problems.

Chapter 5

Options Pricing using Neural Networks and Evolutionary Optimisation

This chapter investigates the use of neural networks for approximating option pricing functions. Neural networks are trained using Breeding-PSO to learn the option pricing functions from numerical training data. The training data consists of numerical prices (priced using Monte Carlo) sampled over the parameter space of σ, r and moneyness, the neural networks are then trained to learn the price functions with respect to these inputs. Ensemble methods are investigated and used to obtain higher degrees of accuracy for the neural network based price approximations, and results in competataive pricing accuracy when compared to Monte Carlo prices. The neural network methodology presented here is tested for European, Asian and American style options.

5.1 Introduction

The pricing of options and derivatives contracts is an important part of operations for financial institutions, one approach is to use stochastic differential equations (SDEs) to model the dynamics of the underlying asset and then derive the corresponding partial differential equation (PDE) to solve for pricing the options contract. In the simplest case for European options and when the underlying asset

is assumed to follow Brownian motion the PDE can be analytically solved by the Black-Scholes solution [116], but either when more complex models for the underlying asset , for example introducing multiple stochastic factors, or exotic options with path dependent payoffs, the analytical solution is often unobtainable and as a result numerical methods have to be employed. In this work traditional numerical methods are classed as those which are already well established approaches within the literature and industry, such as Trees, Finite Difference (FD) and Monte Carlo (MC). These methods enjoy popularity due to the vast amount of literature on the topics, and the relative ease of implementation. In particular Monte-Carlo methods are popular due to the methods flexibility and ease of implementation, Monte-Carlo being a simulation based method makes it an approachable method when complex models or exotic payoff are used. The major disadvantage of Monte-Carlo pricing is that due to the large number of independent simulation runs required to achieved reasonable accuracy makes it computationally inefficient and slow, this is illustrated via the extensive modern literature on improving such methods using high-performance computing devices such as GPUs, Multi-Core CPUs and specialised hardware. Another issue with 'traditional' numerical methods is that they do not provide a generalised solution for the model with respect to the model parameters, the solution is case specific, this means that the numerical computation has to be reran for every new parameter set, which in the environment of todays dynamic markets can lead to large amounts of computation.

The motivation of this work is to provide a fast and accurate approximation to the parameterised analytical solution for cases when otherwise numerical methods would have to be used. The aim of this work is to present a methodology that provides accuracy as good as current numerical methods employed, but is easily parameterisable and more computationally efficient.

It is proposed to use neural networks combined with 'traditional' numerical methods to produce an approximation of the analytical solution for stochastic options pricing models. In essence this methodology uses numerous numerical simulations sampled over the parameter space to train a neural network which then

operates as a function approximator over the parameter space. Similar techniques have been used in other engineering domains, such as in structural engineering to model structural integrity and stress testing [192] [193] [194]. This approach is used due to the computationally intensity of rerunning complex simulations for every reparametersition of the design.

This works provides an empirical study of using neural networks to price option contracts. To begin with the initial methodology is tested for the simple case of European call options, having the known Black-Scholes solution, and being the simplest type of payoff, the proposed methodology can be analysed and refined. The second case then increases the complexity by introducing geometric Asian call options which have a path dependent payoff, but also have an known analytical solution. By validating this methodology for geometric Asian options means that it could also then be used to price similar Arithmetic Asian options for which there currently is no known analytical solution. Finally, the case for American put options is looked at, the introduction of early exercise makes these the most challenging type of option to price and there currently is no known analytical solution, the accuracy of this methodology is compared to a test set of highly accurate numerical prices from the literature.

5.1.1 Options Pricing using Neural Networks

Using neural networks to solve SDEs and PDEs have some appealing advantages:

- Once trained a neural network can be used to approximate the generalised solution for the PDE, this is coined here as a pseudo-analytical solution. The major advantage of this solution means that the method does not have to be reran for any parametric changes and can provide quick offline results on demand.

- Feedforward computation of a neural network is relatively simple providing fast results.

- The structure of neural networks can be easily parallelised and implemented of high-performance computing devices to further enhance performance.

Using the neural network as an indirect approximators/interpolator, it is proposed to use neural networks combined with traditional numerical methods to produce an approximation for the generalised solution of stochastic options pricing models. In essence this methodology uses traditional numerical pricing results, obtained using Monte-Carlo pricing, sampled over the model parameter space, Ω, to train a neural network which then operates as a function approximator over the parameter space to approximate the generalised solution. Using a single layer neural network, $N_1()$, the price approximation for a call option, $\tilde{C}$, with model parameters, $\omega \in \Omega$, is thus given by

$$\tilde{C}(\omega, K, S_t) = N(\mathbf{W}, \omega, K, S_t) = f\left(\sum_i \sum_j w_{i,j}^l y_j^{l-1}\right) \tag{5.1}$$

where y_i^l is the output of neuron i in the layer l, when $l = L$ this represents the network output layer and when $l = 0$ this represents the network inputs, and $\mathbf{W}$ is the weight matrix found for the network via training.

The training procedure of this method may be in the short-term relatively computationally intensive compared to solving a single parameter setting of a given model, but in the long term this method can be seen to be extremely efficient as it produces a single neural network model that can be used for all parameter settings over the given parameter space. The solution can be provide in O(1) complexity requiring one simple forward pass through the neural network, which is considerably less computationally expensive than rerunning numerical computations for every new parameterisations.

The major difference in this work compared to other neural network option pricing approaches is that a generalised solution for any parameter set $\omega \in \Omega$ is attempted to be learnt, whilst other approaches tend to keep the model parameters fixed and only vary the time to maturity, τ, and moneyness ratio $\frac{S}{K}$.

One of the first attempts for options pricing with neural networks is by Hutchinson [195], and the majority of the literature focuses on using neural networks

as a non-parametric approach for pricing options, the options price model is trained from actual market data [196] [197] [198] [199]. This may not work for illiquid exotic derivatives/options where there is not enough data to fully capture the market dynamics. Models trained on market data are also black box solutions with no knowledge of the underlying market models and dynamics, in the current regulatory climate models with certain unknown behaviour may not be preferable, whereas the approach in this work uses well defined stochastic models which allows implicit control over the neural networks behaviour. Other approaches use neural networks to assist/add corrections to parametric models for example [200].

The previously mentioned approaches are non-parametric and use market data, for this work it is more of interested in capturing the behaviour of the parametric stochastic models used in theoretical pricing. A parametric model based neural network pricing approach has been implemented by Morelli *et al* [201] for pricing European options. They use the Black-Scholes model where the asset price is governed by GBM, and approximate the parametric solution using single layer MLPs and RBFs with 50 neurons. In this approach the neural network is modelling the discounted expectation i.e. the stochastic integral, of the GBM stochastic process

$$N(\tau,\sigma,r,S,K) = e^{-r\tau}E[f(S_T)] = e^{-r\tau}\int_0^{\inf} \max(S_t - K)^+ ds. \tag{5.2}$$

They used a fast path integrals approach to numerically derive the training data for European options. This work only provided a brief exploratory study and does and concludes that RBFs and MLPs could provide reasonably accurate pricing for European options, with pricing error of the order of 10^{-2} and 10^{-3} respectively. It should be noted that training the MLP took 4hrs whilst the RBF only took minutes. Results for exotic American options were not encouraging and as a result were omitted by the authors in publication [202]. It was concluded that further training methods, such as evolutionary algorithms may improve results, and the pricing of exotic contracts and complex models should be further investigated.

Das *et al* [203] introduce a hybrid parametric model which uses a similar methodology, neural networks, in this case extreme learning machines, are trained

from model generated options prices. In this case the training data is comprised of the three different traditional numerical methods to take advantage of their strengths and weaknesses. But again, the model generated is not the full generalised solution, as r and σ are not taken into consideration as model inputs. This therefore means the model would have be retrained for every new parameterisation.

One more example where neural networks have been applied directly to estimate the properties of stochastic processes in options pricing is by Kohler *et al* [204]. Kohler *et al* use neural network regression to compute the conditional expectation for the continuation values for pricing multi-dimensional American Basket options. To the best of our understanding the neural network is training on simulated sample paths and is use to learn and approximate the regression problem

$$q_t(x) = E[\max\{f_{t+1}(X_{t+1}), q_{t+1}(X_{t+1})\}|X_t = x] \quad (t = 0, 1, \ldots, T-1). \quad (5.3)$$

Kelly [205] attempted to price American put options using neural networks, although this approach was based on using real market data as the training source. Though high errors were found in the case of higher priced options in-the-money options, although this could be partially due to the low liquidity of in-the-money options in the market which was noted later on by Hospedales *et al* "[w]e discard in-the-money option quotes because trading is very inactive for those options thus their prices are not reliable" [206]. Limitations of similar approaches for American options using MLPs and SVMs for pricing American options was again later found by Pires *et al* [207].

Improvements to neural network pricing methods have involved using modular neural networks, Gradojevic *et al* [199] decomposes the model into 9 modules using moneyness and time-to-maturity as factors, when trained on data. Furthermore, in [208] Gradojevic *et al* take a different approach compared to modelling the price directly as a single function of the input parameters, but instead uses a decomposition and classification based on three factors: moneyness; implied volatility and time-to-maturity. Hospedales [206] uses a multi-model gated approach, here the output is a weighted sum of outputs from individual models where the weighting is

learnt by a second network with inputs being τ and $\frac{K}{S}$.

In other domains neural networks have been used as approximators to solve differential equations, partial differential equations and stochastic differential equation [209]. Such methods have been used in the finance literature, under the name of meshless methods (the interested reader is referred to [210]), to solve models such as the Black-Scholes equation, but this does require the problem to be well defined and again the approach is non-parametric with respect to model parameters. As such we look to provide a more flexible novel hybrid numerical method for solving stochastic models allowing for more complex pricing models, and a parametric solution.

5.2 Methodology

In this work neural networks are used to price financial options by approximating the integral of the underlying stochastic process [201], for European style options the price of a call option is estimated using a neural network $N()$ as

$$N(\omega) = e^{-rt} E[f(S_T)] = e^{-rt} \int_0^{\inf} \max(S_t - K)^+ ds \tag{5.4}$$

where $\omega \in \Omega$ is the set of the call option model parameters, one of the main differences compared to [201] is that a reduced parameter space from $5 \rightarrow 3$ inputs is used, in the case of the Black-Scholes model $\omega_i = \{M_i, \sigma_i, r_i\}$.
The proposed pricing methodology consists of four main stages:

1. **Data Generation and Sampling** : Sample the model parameter space and generate the options pricing training and validation data using a reliable numerical method for solving the SDE derived from the pricing model.

2. **Training** : Train the neural networks with the sampled parameters as inputs to learn the option pricing model from the numerical training data. Z number of independent neural network models are trained. Neural network training here is done using the BrPSO algorithm to find the neural network weights. A novel data transform, and two part fitness function are used to aid the learning

of the neural networks.

3. **Model Creation** : Ensemble the neural networks into a single model. In this work it is investigated for using either the mean or linear regression methods to obtain the set of linear weights for the outputs of the neural network models.

4. **Testing** : Input desired parameters into the network to obtain the price approximations from the neural network model. The errors of the neural network price approximations are then measured against known solutions for European and geometric Asian options, and compared against numerical Monte-Carlo price approximations.

5.2.1 Data Generation and Sampling

Monte-Carlo methods are used as the numerical method to generate the training price data, Monte-Carlo Pricing provides a flexible framework, and is particularly useful for the case of path dependent exotic options, and high-dimensional models. Although in the preceding discussion Monte-Carlo and other numerical methods are critiqued as being computational inefficient, the use of numerical methods is justified in the case of offline training. The offline training procedure is only required once, and although generating the data may be respectively costly, it is a one off cost, and once the neural network model is trained it can be used online without the requirement for such costly computation, only requiring a simple forward pass of the network. The MC pricing data has been produced using the Monte-Carlo model (using the Longstaff-Schwarz method for American options) built into the MatLab Finance toolbox [211], for each pricing run 10,000 replications and 365 periods are used; the corresponding analytical solutions/approximations are also calculated for each sampled point to provide a benchmark reference for comparison.

5.2.1.1 Parameter Space Reduction

The neural network methods used within the literature fail to fully exploit the generalisation abilities of a neural network. If the geometric Brownian motion model is

considered the options price is given as a parameterised function of the five parameters, S_t, K, r, σ and τ, and a generalised solution should be considered one that is dependent on the complete span of the parameter space, but what is commonly seen is the simplification to considering only three parameters $S_t.K$,τ, although these methods then present an approximate solution it is only limited to the fixed parameter value of σ and r. Given that the financial markets are a highly dynamic environment the values of σ and r are subject to constant change, therefore still resulting in the approximated solution to be recalculated for these new values, thus the solutions approximated by these methods are not a fully generalised solution of the problem. It will be shown that a better choice of three input parameters allows the complete range of parameterised option prices to be represented.

This work looks at the ability of neural networks to approximate a fully generalised solution with respect to the five model parameters of the Black-Scholes PDE model. To maximise performance it is possible to apply some mathematical relations of Black-Scholes options prices to reduce the neural network model space from the original five parameters to use only three and still generate a fully generalised solution.

Firstly the strike price K can be removed as a parameter by using homogeneity of an options price

Theorem 5.1. *(Merton [212], Theorem 8.9) If the return distribution is independent of the stock price then the options price $V(S,K)$ is homogenous to degree one for both S and K, this implies that for a constant $\alpha \in \mathbb{R}$*

$$V_1(\alpha S, \alpha K) = \frac{1}{\alpha} V_2(S,K). \tag{5.5}$$

The use of the moneynes ratio, $m = \frac{S}{K}$, within the literature is a special case of Theorem when $\alpha = \frac{1}{K}$.

Corollary 5.1. *Given Theorem 5.2.1.1 holds it is possible to set K as a constant,*

$K = 1$ by using $\alpha = \frac{1}{K}$ and derive all option prices for $S, K \in \mathbb{R}$ as

$$V(S,K) = KV\left(\frac{S}{K}, 1\right). \tag{5.6}$$

The use of Corollary 5.1 is the first simplification of the neural network model input space by using the moneyness ratio and reduces five parameters down to four. The final simplification is using the time scaling property of Brownian motion below.

Lemma 5.1. *Given two different diffusion times τ_1 and τ_2 and assuming no drift , $\mu_1 = \mu_2 = 0$ two Brownian motions are equivalent given that $B_1(\sigma_1, \tau_1) = B_2\left(\sigma_1 \frac{\sqrt{\tau_1}}{\sqrt{\tau_2}}, \tau_2\right)$.*

Proof. The proof is elementary following from the definition of Brownian motion, it can be seen that for two Brownian motions with $\mu = 0$, it must hold that $\sigma_1\sqrt{t_1} = \sigma_2\sqrt{t_2}$ for equivalence. ■

Following from this and assuming that the underlying price process follows geometric Brownian motion it is possible to derive below an equivalence relation based on time-scaling properties of the price process allowing for the final reduction of the neural network model input space.

Theorem 5.2. *Assuming the underlying price dynamics of S_t follow a geometric Brownian motion for an option given by $V(S,K,\tau,\sigma,r)$ the price can be calculated using a function of only three parameters M,σ and r*

$$V(S,K,\tau,\sigma,r) = \frac{1}{K_1} V\left(M, 1, 1, \sigma\sqrt{\tau}, r\tau\right) \tag{5.7}$$

where M is the moneyness of the option given as $M = \frac{S}{K}$.

Proof. The proof follows from first combining Corollary 5.1 and Lemma 5.1. Under the risk-free measure $\mathbb{Q}$ it can be seen that the interest rate scales linearly with τ. ■

This work will focus on approximating the function option price of the function $V(M,1,1,\sigma,r)$, where $K=1$ and $\tau=1$ for compactness this will be abbreviated to $\tilde{V}(M,\sigma,r)$.

The implication of Theorem 5.2 is that if prices over the whole parameter space for $\tilde{V}(M,\sigma,r)$ are known then it is possible to use the equivalence relation to obtain the price for any option $V(S,K,\tau,\sigma,r)$. Therefore it is possible to reduce the input parameter space of the neural network model from five to three and still approximate a fully generalised solution for the option price.

Theorem 5.3. *Given time-scaling relationship in Lemma 5.1 the unique mapping* $V(S,K,t_1,\sigma,r) \rightarrow V(S,K,t_2=\left(1,\sigma\sqrt{t_1}^{-1},rt_1^{-1}\right)$ *is unidirectional. Given a fixed* σ *and* r *it is not possible to find the unique time mapping* $g(t)$ *such that* $V(S,K,t_1,\sigma,r) \rightarrow V(S,K,g(t_1),1,1)$

Proof. The proof is relatively simple. Assuming Brownian motion the relationships of t_1 and t_2 are given by

$$\sigma_1\sqrt{t_1}\sigma_2^{-1} = \sqrt{t_2}$$
$$r_1 t_1 r_2^{-1} = t_2$$

under the assumption of the theorem σ_1 and r_1 and fixed and can be assumed to be 1, and σ_2 and r_2 are two separate independent variables. Thus it can be observed that two solutions for t_2 exist: $t_2 = t_1\sigma^{-2}$ given $r=\sigma^2$ which violates the independence of σ_2 and r_2; or $t_2=0$ given $\sigma_2^2 - r \neq 0$. ■

The implication of Theorem 5.3 is that it is not possible to approximate the complete span of options prices in the space $\Omega \in [S,K,\tau,\sigma,r]$ by approximating the two dimensional function $V(M,\tau,1,1)$, but it is possible by approximating the hree dimensional function $V(M,1,\sigma,r)$ which represents the most compact representation of the options pricing function. This also supports our initial claims that current neural network methods that only approximate the price as $V(S,\tau)$ cannot represent a fully generalised solution with respect to all the model parameters.

5.2.1.2 Latin Hyper-Cube Sampling

Compared to most other neural network methods in the literature which use only use two basic inputs, in this work the space is sampled over the three dimensional parameter space: interest rate, r; volatility, σ; moneyness, $M = \frac{S}{K}$.

The training data is partitioned into three sets, one for training, one for validation and one for out of sample testing. For each type of option we generate 2000 samples for training, and 1000 samples for both validation and test data sets. The range of the parameter space used is: $r \in [0.01, 0.1]$; $\sigma \in [0.1, 0.5]$; $S \in [0, 100]$; and $K \in [0, 100]$.

The crux of the proposed methodology relies upon efficiently sampling over the stochastic model parameter space. The issue faced by naive sampling methods, such as grid sampling, is that they suffer from the curse of dimensionality and do not scale well with increasing dimensions in the parameter space; as such a scalable random sampling method is required. Latin hypercube sampling (LHS) is a stratified random sampling method which gives a better distributed representation of the parameter space than just naive random sampling. In LHS each parameter is divided up into equally probable intervals, there is then equal probability that the sample will be chosen from within each interval.

5.2.1.3 Data Transforms

When using neural networks of options pricing one of the issues encountered is the large range of magnitudes of prices from deep out-of-the-money options to deep in-the-money-options. The large range of magnitudes therefore makes it very hard for neural networks to effectively output this range of values; in general neural networks work best when all the targets are a similar magnitude in value. Other work has overcome this problem by using multiple network architectures, such as gated neural nets where an initial classification process is used to pass the parameters to a secondary specialised neural network for that pricing range. This can lead to elaborate network architectures that can take longer to train and produce a less compact representation of the solution.

To resolve this issue a data transform is introduced and applied to the target

output values; the neural network is therefore learning to output the transformed value of the target values,

$$N(\omega) \to T(V(\omega)). \tag{5.8}$$

An approximation to the actual target value, $\hat{V}$, can then be retrieved by inverting the applied transform

$$\hat{V}(\omega) = T^{-1}(N(\omega)). \tag{5.9}$$

Using a transform for the training values is desirable due to the magnitude differences between in and out of the money options, otherwise the network training would become strongly biased towards the larger in-the-money options, therefore a log based transform is used as below (it was found that using strictly log only transform was not beneficial).

The transform chosen aids the network's learning by transforming the target training values to approximately similar magnitudes; this is done via a $\log_{10}$ type transform

$$T_{\text{sp10}}(x) = \log_{10}(10^x - 1)\ , x \neq 0, \tag{5.10}$$

$$T^{-1}_{\text{sp10}}(z) = \log_{10}(10^z + 1)\, z \neq 0. \tag{5.11}$$

This is dubbed a softplus-base10 (sp10) transform due to its similarity to the softplus function. The softplus-base10 transform transforms all values $x < 1$ using a $\log_{10}$ transform, mapping these x values to larger magnitude negatives values, but remains close to linearity for values $x > 1$. This function is bijective given that for all $x > 0$, and hence is applicable in the case of options pricing which does not involve negative values x.

A final set of linear transforms are applied to the asset price, S_t and strike price, K input parameters. Neural network training can be aided by ensuring that the inputs are all of roughly equal magnitude; the asset price and strike price input parameters

are therefore multiplied by 0.01 so they become similar to the magnitudes of the interest rate and volatility.

5.2.1.4 Price Resolution and Rounding

A threshold for the smallest representable price is applied, $V_{\min}$, to create a lower bound for the sp10-transformed space. A threshold value for the lower bound is required due to the fact that the price range for options in the continuous space, $V \in [0, S_{\max}]$; this means that $T_{\text{sp10}}(V)$ is unbounded, $\lim_{V \to 0} T_{\text{sp10}}(V) \to -\infty$, whilst for upper bounds it is known that the option price can never exceed the maximum asset price. Therefore the threshold makes the range of values for the neural network to approximate more compact. The threshold is simply applied within the sp10-transform as an addition of the constant $V_{\min}$, the transformed domain is now bounded by

$$\log_{10}(V_{\min}) \leq T_{sp10}(V + V_{\min}) < S_{\max}. \tag{5.12}$$

Although this may seen to limit the methodology's range for representing and accurately pricing deep-out-of-the-money options, as long as a small enough threshold is used within the practical limits of an option price quote this should not present an issue after rounding the output to a magnitude larger than the threshold. In the exchange, prices are quoted to a limited number of decimal places making the need for extremely small resolutions redundant when practically applied. Throughout this work a threshold value of 10^{-8} is applied for training, and the outputs of the neural network are then rounded to six decimal places.

5.2.2 Training

It should be reemphasised at this point that while this process is necessary time consuming the benefit of the methodology is to be seen in the much acerbated times the pricing can be achieved for test data.

5.2.2.1 Neural Network Architecture

The neural networks considered here are feed forward multi-layer perceptron (MLP) networks; two different MLP architectures are investigated. The first is a simple two layer feedforward network shown and can be defined as

$$N(I) = \sum w_{i,j,L} f\left(g(I,W)_{L-1}\right)_L \quad \forall l = 1 \ldots L \tag{5.13}$$

$$g(I,W)_l = \sum w_{i,k,l} f\left(g(I,W)_{l-1}\right)_l \tag{5.14}$$

$$g(I,W)_0 = \sum w_{i,k,0} I_k \quad l = 0 \tag{5.15}$$

where l is the layer index and $l = 0$ corresponds to the network inputs, in the case of an MLP with two hidden layers $L = 2$.

The second neural network architecture investigated is a two step multi-stage network, similar to the gated networks used in [206]. In this network architecture two smaller networks are connected, where the output/s of the first network and the original inputs are both passed in as inputs into the second network,

$$N_{\text{MSCN}} = N_2\left(N_1(I), I\right). \tag{5.16}$$

This second network, N_2, then acts as an additional corrector for errors generated by the first network; a similar network construction was used for example in the successful PSIPRED protein structure predictor [213]. The architecture implemented here uses two networks both with two hidden layers of ten neurons.

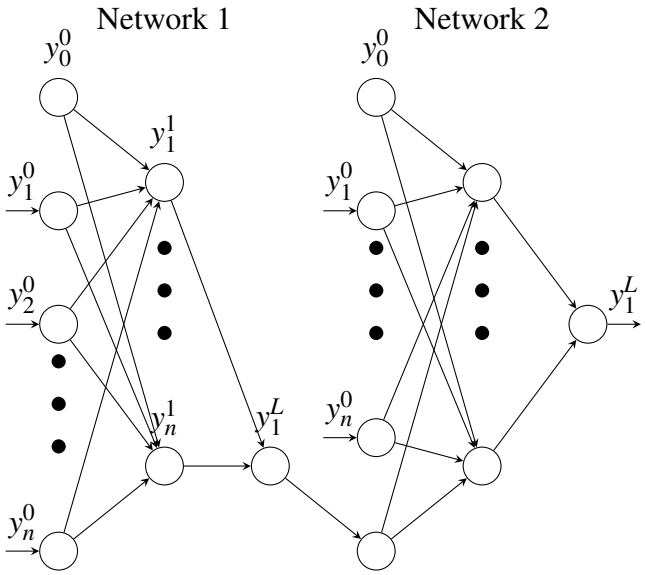


Figure 5.1: The architecture of the multi-stage network architecture. This architecture consists of two networks with the output of the first connected as an input to the second, the second network also takes in the original inputs used in network one, the second network then acts as a corrector on the output of the first.

The transfer function, $f(a)$, used in the hidden units for all network architectures is the sigmoid

$$f(a) = \frac{1}{1+e^{-a}} \tag{5.17}$$

where a is the activation value of the neuron, previously defined as the function $g(I)$.

5.2.2.2 Calculating option Price Sensitivities

To find the option price sensitivity values, known as the Greeks (Δ, ρ, κ), the derivative of the network's output with respect to the desired input, $\frac{\partial N(I_1 \ldots I_M)}{\partial I_k}$, needs to be calculated. For a single layer MLP network the derivative is simply given using the chain rule by

$$\begin{aligned} \frac{\partial N(I_1 \ldots I_M)}{\partial I_k} &= \sum w_{i,1} \frac{\partial f\left(\sum w_{i,j} I_j\right)_i}{\partial I_k} \\ &= \sum w_{i,1} w_{k,i} f'\left(\sum w_{i,j} I_j\right)_i . \end{aligned} \tag{5.18}$$

For an arbitrary MLP network with $0 \ldots L$ layers, the chain rule has to be iteratively applied

$$\frac{\partial N(I_1 \ldots I_M)}{\partial I_k} = \frac{\partial f\left(\sum w_{i,k,l-1} f(\ldots)_{i,l-1}\right)_L}{\partial I_k}. \tag{5.19}$$

Given that in this network design $f(g(I))_L = \sum w(i,k,L) f(g(I))_{L-1}$

$$\begin{aligned} \frac{\partial f\left(\sum w_{i,k,l-1} f(\ldots)_{i,l-1}\right)_L}{\partial I_k} &= \sum w_{i,k,L-1} \frac{\partial f(g(I))_{i,L-1}}{\partial I_k} \\ &= \sum w_{i,k,L-1} \frac{\partial f(g(I))_{i,L-1}}{\partial g(I)} \frac{g(I)}{\partial I_k} \end{aligned} \tag{5.20}$$

where $g(I)_l$ is the activation value of the neuron and is expanded out as $\sum w_{i,k,l-1} f(g(I)_{l-l})_{l-1}$; hence the chain rule is recursively applied as

$$\frac{g(I)_l}{\partial I_k} = \sum w_{i,k,l-1} \frac{\partial f\left(g(I)_{l-1}\right)_{i,l-1}}{\partial g(I)_{l-1}} \frac{g(I)_{l-1}}{\partial I_k} \tag{5.21}$$

$$\frac{g(I)_{l-1}}{\partial I_k} = w_{i,k,0} \quad l = 1.. \tag{5.22}$$

With respect to the multistage network also investigated, given by $N_{\text{MSN}} = N_2(N_1(I), I)$, the above recursive formula for the derivative continues from the second network into the first network $N_1(I)$

$$\frac{g(I)_{l-1}}{\partial I_k} = w_{i,k,0} + w_{i,k,1} \frac{\partial N_1(I)}{\partial I_z} \quad l = 1. \tag{5.23}$$

where $\frac{\partial N_1(I)}{\partial I_z}$ is given by Equation 5.22.

Finally given that transformations are applied to the input and output values of the neural networks these also need to be incorporated via the chain rule into the derivative of the neural network. In the case of the linear transforms $T_L(x) = Cx$

applied to the inputs this gives

$$\frac{g(I)_{l-1}}{\partial I_k} = w_{i,k,0}V \quad l = 1. \tag{5.24}$$

For the inverse soft-plus-base-10 transform applied to the output of the network the final derivative for the model of the option price is thus given by

$$\hat{V}' = N'(I)\, T'^{-1}_{\text{sp10}}(N(I)) \tag{5.25}$$

5.2.2.3 Training Method

The neural networks are trained using the evolutionary Breeding Particle Swarm Optimisation (BrPSO) algorithm [214]. BrPSO was observed to produce superior neural network training results compared to standard PSO. As explained in Section 2.2.2, when applying particle swarm optimisation for neural network training the position of each particle represents a vector in the search space where in this case the search space is the neural network weights, $\mathbf{W}$. The quality of the position for each particle is evaluated to give a fitness value fit($\mathbf{x}$); for every iteration the particles then move throughout the search space to find the optimum vector.

In this application the fitness value is calculated as a sum of the mean absolute error of the neural network approximation for the transformed option prices and the mean relative error of the inverse transform of the network output compared to the raw training values of the option price; this is given in Equation 5.30. The two-component fitness value is used because it was observed when using just the transformed option price that small errors in the compressed log transform values resulted in significantly larger errors when the inverse transform was then applied to obtain the final price approximation; this can be illustrated when considering a log transform applied to the target values y,

$$N(\omega) = \log(y) + \varepsilon \tag{5.26}$$

$$\hat{y} = e^{\log(y)+\varepsilon} = ye^{\varepsilon} \tag{5.27}$$

where ε is the training error and $\hat{y}$ is the inverted approximation of y; it can be seen that when the transform is inverted the training error grows exponentially.

When using no transform, or training with the inverse transform applied to the network output, results were poor as this fails to capture the range of output magnitudes. The combination of components in the two part fitness function allows the network to efficiently output a wide range of price magnitude via the transform but also minimise the errors that occur during the inverse transform to the final price given that this involves an exponential function. Looking back at Equation 5.26 the fitness function minimises both ε and e^{ε}. The fitness for each particle in the optimisation can thus be given by

$$\begin{aligned} \text{fit}\left(\mathbf{W_i}\right) = E_{\text{MAE}}\left(N\left(\mathbf{W_i}, \mathbf{Y^0}\right), T_{\text{sp10}}\left(\mathbf{V}\right)\right) \\ + E_{\text{MRE}}\left(T_{\text{sp10}}^{-1}\left(N\left(\mathbf{W_i}, \mathbf{Y^0}\right)\right), \mathbf{V}\right) \end{aligned} \tag{5.28}$$

where $\mathbf{W_i}$ is the matrix of neural network weights represented by particle i, $\mathbf{Y^0}$ is the vector of training input parameter sets i.e. $\mathbf{Y^0} = \{\{\mathbf{y_1^0}, \mathbf{y_2^0} \ldots \mathbf{y_n^0}\}_\mathbf{1}, \ldots \{\mathbf{y_1^0}, \mathbf{y_2^0} \ldots \mathbf{y_n^0}\}_\mathbf{J}\}$, $\mathbf{V}$ is the corresponding vector of target prices for input parameter sets, and $N(\mathbf{W}, \mathbf{Y^0})$ is the vector of neural network approximation outputs for each input parameter set given in $\mathbf{Y^0}$, and where $E_{\text{MAE}}(x,y)$ and $E_{\text{MRE}(x,y)}$ are the mean absolute and mean relative errors, respectively, of the numerical approximations x_n compared to targets y_n.

5.2.2.4 Weighted Training

Weighted training is introduced for two reasons to increase ensemble diversity (see methodology outline) and to create experts for specific regions.

Instead of bootstrapping the data, which has the issue that it risks that some models may lose the ability to generalise in areas where the data is not densely sampled, weightings are added to the training data. In this procedure networks are initialised to specialise in particular parameter regions in ways to be described in more detail below. Selected points within these regions are given the highest weighting with surrounding points given low weightings determined by distance.

Two weighted training methods are implemented, *wt-rand* and *wt-atm* dsitigusihed by the where the most strongly weighted training points are selected. In wt-rand the most strongly weighted point, the weighting center C_N, is chosen randomly for each network in the ensemble, whereas in wt-atm all the networks ensemble have the same center which determined to be the most difficult region to approximate (at-the-money-options).

The moneyness is the most influentual input with the respective to determining the output price, therefore the weighting used here will be based on the moneyness of the sample. The weighting for a training sample $\mathbf{y^i} = \{\mathbf{M_i}, \sigma_\mathbf{i}, \mathbf{r_i}\}$ is

$$q_i = \log_{10}(|M_i - C_N| + p) \tag{5.29}$$

where C_N is the chosen center for the neural network model N with respect to the moneyness, and p is a constant that controls the maximum size of weighting, this is set to 0.01, which limits the weighting to 100. The fitness function using the weighted samples is now given as

$$\begin{aligned} \text{fit}(\mathbf{W_i}) = \mathbf{q} \cdot E_{\text{MAE}}\left(N\left(\mathbf{W_i}, \mathbf{Y^0}\right), T_{\text{sp10}}(\mathbf{V})\right) \\ + \mathbf{q} \cdot E_{\text{MRE}}\left(T_{\text{sp10}}^{-1}\left(N\left(\mathbf{W_i}, \mathbf{Y^0}\right)\right), \mathbf{V}\right) \end{aligned} \tag{5.30}$$

where $\mathbf{q}$ is the sample weighting vector.

5.2.3 Model Creation

The above methodology is repeated to created a set of neural network models, $\mathbf{N}$, where $N_i()$ is used to denote the ith neural network. This set of models can then be used to produce a final pricing model. This can be achieved one of two ways, either by selecting a single best model as dictated by some selection criteria, or by using an ensemble based model by taking a combination of outputs from a subset of the original models $\mathbf{n} \subset \mathbf{N}$. Based on the ambiguity theorem [215] probabilistically ensembling a model will be statistically at least as good as selecting a single model [216].

5.2.3.1 Ensemble Methods

There are two classes of ensemble methods, training and post-training [217]. In training based methods the neural networks are trained with the intention of being in an ensemble and the ensemble output is optimised; examples of such methods are boosting [218], stacked regression [219], or negative correlation learning [220]. Training based methods have the disadvantage that the training process will be slower as a number of networks have to be trained at the same time which results in an extremely large search space considering all the weights of all the networks. Post-training methods use a set of pre-trained neural networks and then look at how to optimally combine them; which is often done using a second training set; these sets of methods have the favourable characteristic that all the neural networks can be trained independently, although the disadvantage is that it may require a larger set of networks to produce enough diversity. The work of this thesis focuses on the use of post-training ensemble methods as in addition this form of training can be efficiently parallelised.

In this work post-training linearly weighted combinations of the trained neural network models are used, this can be written as

$$\mathbf{N}(\mathbf{x})\beta = \mathbf{v} \tag{5.31}$$

where $\mathbf{N}(\omega)$ is the output matrix of the trained neural network over the sets of test set inputs $\mathbf{x}$ (where $x_i = \{\sigma_i, r_i, M_i\}$), β is the linear weight vector of the ensemble for each neural network model. There are many different ways of finding the set of linear weights [217], in this work two popular methods are used.

Mean: The simplest ensemble method is to take an equal weighting, i.e. the mean, N_{mean}, for each of the test set parameter inputs, $x_i = \{\sigma_i, r_i, M_i\}$ the aggregated model output is

$$N_{\text{mean}}(x_i) = \frac{\sum_{j=1}^{Z} N_j(x_i)}{Z}. \tag{5.32}$$

where Z is the number of trained neural network models. The mean can be a very

robust ensemble method, it is a convex combination such that the ensemble outputs are bounded by the individual outputs of the neural networks. When using the mean is that it assumes the errors for each network, ε_i within the ensemble at every output is symmetrically distributed such that they effectively cancel each other out, or probabilistically the distributions is such that $\mathbb{E}[\varepsilon_i] = 0$. For neural network options pricing using the mean ensemble has been seen to improved option price predictions [221].

Linear Regression: Rather than using an equal weighting which may not be desirable if the error distribution is skewed or there is a high degree of collinearity, linear regression using a second set of training data, with targets $\mathbf{y}_2$ and inputs $\mathbf{x}_2$, can be used to find the optimal set of weights. Linear regression is used to find a weight vector β such that it minimises $\mathbf{N}(\mathbf{x_2})\beta - \mathbf{y_2}$. The linear regression can be achieved in two ways:

- Using unconstrained regression where the weights are found by solving $\beta = \mathbf{y_2}\mathbf{N}(\mathbf{x_2})^{-\mathbf{1}}$, using the Moore-Penrose pseudoinverse $\mathbf{N}(\mathbf{x_2})^{+}$. The advantage of this method is that it allows a good fit to the data to be found, but has the disadvantage that it may result in a non-convex linear combination involving negative weights or not summing to one. This can sometimes lead to out-of-sample predictions becoming unreliable.

- Using constrained non-negative least squares [222], this has the advantage that the linear combination will be convex and therefore more reliable than non-convex linear combinations, and that it can also result in a smaller final ensemble where non-significant models are given a weight of zero. The disadvantages are that the non-negative least squares algorithms are slower, and may not be able to fit as well to the data given the constraints. This is seen later on, and in the case of high collinearity the convex linear combinations are unable to acheive the same level of accuracy of non-convex linear combinations, however it is possible to increase the accuracy of the non-negative weighted ensemble by using data transforms, Section 5.3.5.4.

5.2.4 Testing

The pricing errors discussed, unless mentioned otherwise, are with regards to the out-of-sample test set inputs, $\omega_i = \{\sigma_i, r_i, M_i\}$. The errors of the neural network price approximations for each test sample, ω_i, are measured by the absolute error (AE_i) and absolute relative error (ARE_i) of the neural network price approximation, $N(\omega_i)$, with respect to the analytical Black-Scholes solution for a call option C_{BS}

$$\text{AE}_i = |N(\omega_i) - C_{\text{BS}}| \tag{5.33}$$

$$\text{ARE}_i = |\frac{N(\omega_i) - C_{\text{BS}}}{C_{\text{BS}}}|. \tag{5.34}$$

The same respective error measures (AE and ARE) are calculated for the test set Monte-Carlo price estimations with respect to the Black-Scholes solution. It should be noted as previously mentioned, see Section 5.2.1.4, that the output of the neural network approximation is rounded to six decimal places, as such all other prices discussed, the analytical Black-Scholes solution and Monte-Carlo price estimations, have all been rounded to six decimal places as a suitable level of accuracy for comparison.

The *acceptable practical pricing boundary* is defined such that the numerical price is accurate with respect to the known analytical solution up to 2 decimal places .i.e $AE_i < 0.005$. This is in regards to real world market price quotes, based on the Chicago Board of options Exchange prices [223], which are quoted up to two decimal places. This boundary is shown as a red line in the AE and ARE error plots presented, any error below this line is determined as an acceptable error for practical usage.

A measure of accuracy used to compare models is the *acceptable error rate* which is related to the acceptable practical pricing boundary. Assuming that the input samples are i.i.d, the acceptable error rate is an estimation of AER = $P(|\hat{c} - c| < 0.005)$. As well as creating a model that is accurate, it is important in this application that it generalised well over the whole input domain, therefore the final

model aims to maximise,

$$\arg\max P\left(|\hat{c}-c|<0.005\right)-\mathbb{E}[|\hat{c}-c|] \tag{5.35}$$

this will result in a tradeoff between accuracy in certain regions of the input space, measured by the absolute error, and the overall acceptable generalisation over the whole input space.

To breakdown the pricing capability of the methodology five distinct regions of options price behaviour with regards to the size of the moneyness ratio are defined.

Definition 5.1. The five price regions are defined as:

1. *Deep-Out-The-Money* : Deep-Out-The-Money (DOTM) are defined as options where the strike price is less than half of the asset price, $M < 0.5$ or $\log_{10}(M) \leq 0.30$. The options in this region lie within the degenerative region of the payoff function, and tend to have very small prices which are close to 0.

2. *Out-The-Money* : In this work Out-The-Money (OTM) options are defined as those where the moneyness lies within the region $0.5 < M \leq 0.80$ or $0.30 < \log_{10}(M) \leq 0.1$. In this region the options become more valuable and the behaviour becomes more non-linear as the payoff approaches the discontinuity.

3. *At-The-Money* : At-The-Money (ATM) options are usually defined as those where the moneyness $M = 1$, but because this work looks at a distribution of M values ATM is defined here as those options within a narrow region around $M = 1$, this encompasses ATM options and close-to ATM options, $0.80 < M < 1.25$ or $-0.1 < \log_{10}(M) < 0.1$. This region shows the highest degree of non-linearity as the payoff around the discontinuity begins to become valuable as the probability that the contracts will become valuable at expiry increases.

4. *In-The-Money* : In-The-Money (ITM) options are where the moneyness $M > 1$, this means that the options are currently valuable, and have a higher probability of becoming valuable at expiration. Here ITM is defined within the region $1.25 \geq M < 2$ or $0.1 < \log_{10}(M) < 0.30$.

5. *Deep-In-The-Money* : Deep-In-The-Money (DITM) are defined as options where the moneyness $2 \leq M$ or $0.30 \leq \log_{10}(M)$. In this regions it is highly likely that the options will be valuable at expiration, and behaviour tends to becoming linear with respect to the asset price.

Benchmark comparisons are made against Monte-Carlo numerical methods. Monte-Carlo prices are generated as part of the test set as a means of comparing the neural network price approximation against the popular numerical method to assess this methods practical applicability. Given that the neural network approximations are trained on MC generated data it is expected that the errors of the neural network price approximations should be at least or almost as good as MC price estimations. Monte-Carlo prices inherently have some error present in the form of random noise [128], it is speculated that the neural networks may even be able to produce more accurate prices than MC by being able to smooth out the noise present in the training data.

The methodology is tested on three different types of options, European, Asian and American, each adding a progressive degree of complexity to the problem.

5.3 European option Pricing

European call options are first explored, these options have the simplest form of the payoff function at maturity given as $(S_T - K)^+$, and when the underlying asset price behaviour follows geometric Brownian motion the analytical Black-Scholes solution can be used to give an exact price. The availability of an analytical solution allows the errors and accuracy of the neural network price approximations to be precisely evaluated and compared, from these results initial limitations and ways to improve the methodology can be determined before being applying it to more complex cases.

For the case of European call options the methodology is first investigated using two different neural network architectures, multi-layer-perceptron with 2-hidden layers (MLP-2L) and 2-stage multi-stage network (MSN), each architecture is also implemented with two different numbers of total neurons used in the hidden layers, 20 and 40 neurons, evenly distributed over the hidden layers for each architecture, this results in a total of four different architectures that have been initially explored. For each one of these four architectures a total of 300 independently trained neural networks models are learnt. The neural networks are trained using the evolutionary algorithm BrPSO [214] (for more details see Chapter 3). The neural network training, validation and test data is generated using MC price estimations (10,000 replications and 365 steps) sampled from the input space Ω.

The selection of the best model will be a time consuming process, with many factors to be considered, which will be dealt with sequentially in Sections 5.3.2 - 5.3.4. Before this however, there will be a short discussion of behaviours within different price regions as they elucidate the general problems faced by the neural networks in this option pricing task.

5.3.1 Comparing Price Region Error Behaviour

To highlight the general behaviour of the neural network approximations observed in each price region Figure 5.2 shows the distributions of the magnitudes of the AREs, given as $\log_{10}(\text{ARE}_i)$, for each one of the four network architectures investigated within each of the price behaviour regions, DOTM, OTM, ATM, ITM and DITM; the log is taken because of the wide range of magnitudes of the AREs. In general the overall error behaviours for all network architectures considered here are very similar for each within each of the price regions; however the errors behave very differently when regions are compared.

The most distinct behaviour, when regions are compared, is the error distribution for DOTM options, Figure 5.2.a, which strikingly bimodal, combining very good performance (errors of order 10^-8) with very bad (errors of the order 1). However it should be noted that most cases in which the error is 0 are ones in which the option was priced close to 0 i.e. it was worthless. Overall therefore the practical

value of the models in this region is debatable because only low valued options can be priced accurately.

The peak at ARE$\approx$ 100% for DOTM options is caused by the rest of the low valued options in the price range $10^{-6} < C_{\text{BS}} < 0.01$, in these instances the neural networks approximate these prices as $\hat{C} = 0$ and undervalue the options. This suggests that the gradient of the degenerative slope of the neural network approximation to the solution is too steep and prices decay too rapidly.

Moving onto the OTM region a much less noticeable bimodal behaviour is observed; this region has the highest frequency for ARE$>$ 100% , and is because this region has the highest frequency of prices in the range$10^{-6} < C_{\text{BS}} < 0.01$ and therefore suffers the same problem as described for DOTM options. There is also a second behaviour for which there are some extremely high AREs. In these cases the neural network approximations are considerably overestimating the prices, this occurs for low priced options with a small σ and this issue can be further illustrated later on when looking at the accuracy of the Vega. This is due to the discontinuity of the option payoff function, in this case a small σ is equivalent to a smaller τ and the option price function more closely resembles the final payoff condition.

As the price behaviour become more linear in the ITM and DITM regions the the distributions tend towards being more normally distributed; it should again be noted that this is the distribution of the magnitude of ARE. This is because the as the price behaviour becomes more linear, and for ITM options the range of magnitude of prices becomes a lot smaller, as well as the SP10 transform has less on effect the learning of these prices is easier.

Although the AREs are seen to decrease as the region moves from DOTM$\rightarrow$DITM, this does not mean that the price approximations in the regions become more accurate with respect to the acceptable pricing boundary. Because option prices also increase as the region moves from DOTM$\rightarrow$DITM in order to maintain acceptable performance the ARE should correspondingly decrease. While from Figure 5.2 shows evidence of this happening i.e. the mean ARE is decreasing, the effect is not sufficient to provide adequate overall price regions.

Table 5.1 show the the acceptable error rate, which proportion of prices that lie within the acceptable price range.The results of Table 5.1 highlight the issue of consistency of the neural network model approximations for ITM and DITM options by showing that only a small percentage of the prices generated by all the models are acceptable. This therefore leads to the question of model selection, how can the models that generate the acceptable prices be recognised and selected for the test cases, this will be addressed later on when considering model diversity and aggregation methods.

	DOTM	OTM	ATM	ITM	DITM
MLP-2L(20)	0.728	0.135	0.012	0.013	0.017
MLP-2L(40)	0.727	0.133	0.012	0.012	0.017
MSN(20)	0.729	0.139	0.010	0.009	0.012
MSN(40)	0.729	0.141	0.010	0.010	0.013

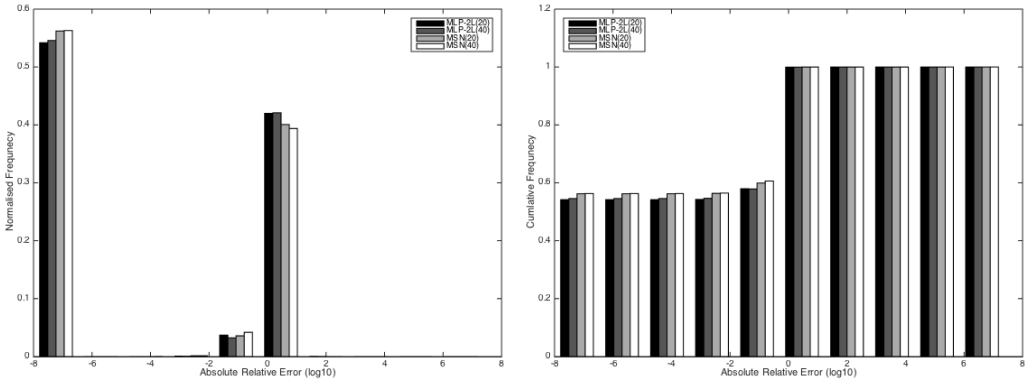
Table 5.1: Proportions of price approximations within the acceptable error boundary for each of the 5 regions.

5.3.2 Exploring Network Architectures

5.3.2.1 Comparing Network Size

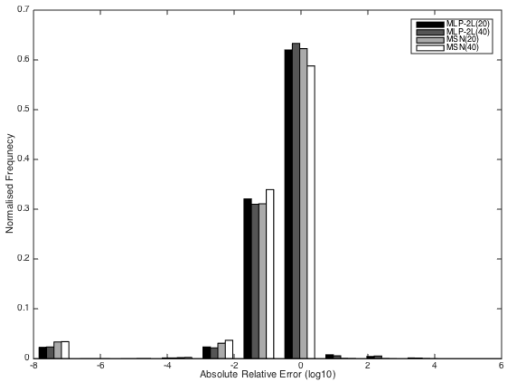
The effect of the number of neurons used in the hidden layers is briefly investigated for both of the neural network architectures, MLP-2L and MSN, used here to establish if a sufficient number of neurons has been used for the neural network approximations. Both 20 and 40 hidden neurons have been explored; for MLP-2L this corresponds to networks of the form 3-10-10-1 and 3-20-20-1, and for MSN 3-10-1-4-10-1 and 3-20-1-4-20-1, respectively.

From the RHS of Figure 5.2 the observed effect of the number of neurons on cumulative frequency is more significant for the MSN architectures than for the MLP architectures; for all of the regions it can be seen that the 40 neuron MSN architecture has a higher percentage of smaller AREs compared to 20 neurons for each of the price behaviour regions: DOTM and OTM have a higher percentage of ARE$<$ 10% and for ATM and ITM ARE$<$ 1% and DITM ARE$<$ 0.1%. In addition from Table 5.1 it can be seen that the 40 neuron MSN has a marginally

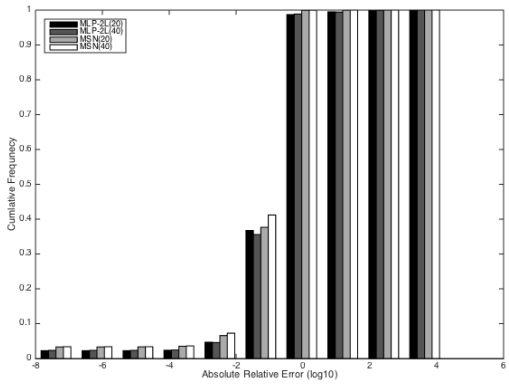


(a) Deep-Out-The-Money

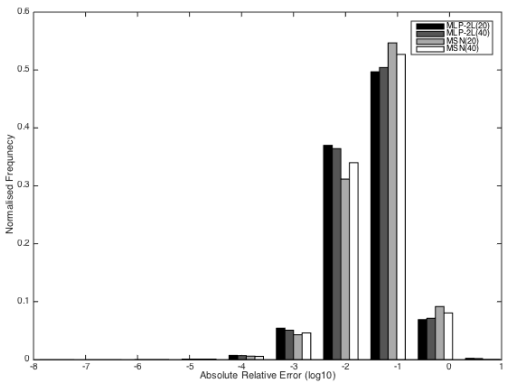
(b) Deep-Out-The-Money (cumulative dist')



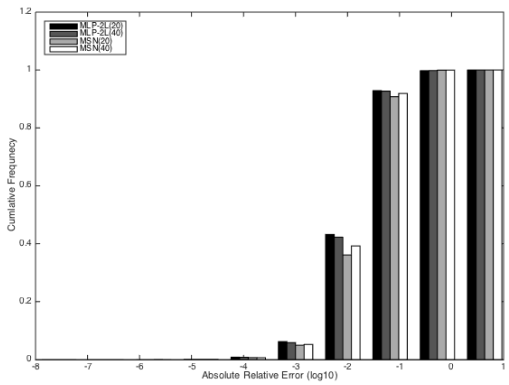
(c) Out-The-Money



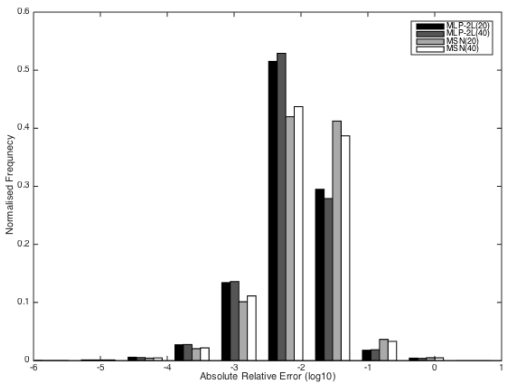
(d) Out-The-Money (cumulative dist')



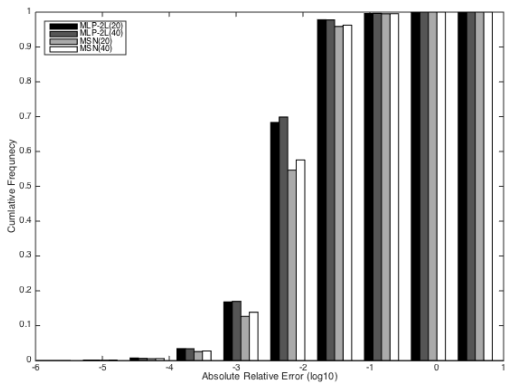
(e) At-The-Money



(f) At-The-Money (cumulative dist')

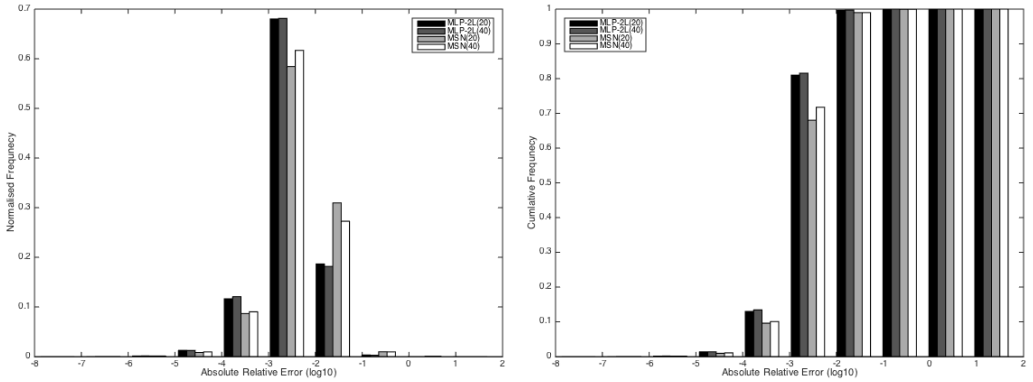


(g) In-The-Money



(h) In-The-Money (cumulative dist')

Figure 5.2: Distribution (LHS) and cumulative distribution (RHS) of absolute relative price errors of the 4 neural network architectures for the 5 defined regions of options prices.



(i) Deep-In-The-Money **(j)** Deep-In-The-Money (cumulative dist')

Figure 5.2: cont. Distribution (LHS) and cumulative distribution (RHS) of absolute relative price errors of the 4 neural network architectures for the 5 defined regions of options prices.

better acceptable practical error rate for OTM, ITM and DITM options.

Although the overall distribution of errors for the smaller 20 neuron MLP-2L architecture seems marginally more accurate, the maximum error compared to 40 neurons is a lot larger. With this in mind the 40 neuron architecture could be considered more reliable. From hereon the discussion will focus on the larger 40 neuron networks due to their slightly better reliability determined by the maximum AREs observed.

The analysis of the effect of network size only looks at two different numbers of hidden neurons. This work is sufficient to confirm that more neurons can produce a more accurate/reliable model and that a sufficient number of neurons are used to closely represent the upper bound of approximation ability. For future work it would be insightful to investigate a wider range of neuron numbers to see if there is any further improvement in the approximation ability of the networks or if there is an asymptotic limit to the approximation accuracy with regards to the number of neurons. It would also be interesting to the find the best network settings with respect to complexity and accuracy; for this purpose it would be sufficient to use an evolving network architecture to find the optimal design with respect to network complexity and approximation accuracy.

5.3.2.2 Comparing Architecture

With respect to each of the price behaviour regions, at first glance the neural network architectures, MLP-2L and MSN, show very similar patterns of error behaviour. However looking at the behaviours of the two architectures in more detail brings to light some important differences and shows that the MLP-2L here has a better overall function approximation ability. (As decided above, the discussion continues with a focus only on the larger 40 neuron networks.)

The differences to accuracy between MLP-2L and MSN for DOTM and OTM are respectively smaller than for ATM, ITM and DITM options. Hence while MSN performs better for DOTM and OTM options MLP-2L is preferable.

For DOTM and OTM options the MSN architecture shows a better level of accuracy. For DOTM options the MSN has around a 2% higher frequency of exact price solutions and respectively has around a 2% lower frequency of ARE$>$100%; this indicates that the MSN networks are better at classifying worthless options where $C_{\mathrm{BS}} = 0$. A similar behaviour can be seen for OTM options where there is a higher frequency of ARE$<$100%, this further supports that the MSN is better for approximating low-valued options albeit only slightly. The MLP-2 has a better overall mean ARE, although there are some cases of extremely high average errors caused by outlier outputs, whilst MSN has a lower rate of these outliers for DITM and OTM options.

The more significant differences in the AREs can be seen for the ATM, ITM and DITM options. For ATM options MLP-2L only has a greater frequency of ARE$\leq$1% by 3%, in addition MSN has a higher rate of ARE$\geq$100% by around 1%. For ITM options MLP-2L has a greater frequency of ARE$\leq$1% by 12% compared to MSN and for DITM MLP-2L has a greater frequency of ARE$\leq$0.1% by 10%. These differences are also noticeable when looking at the acceptable pricing error rates, although not as significant as the differences in the ARE distributions, MLP-2L(40) has better acceptable error rates compared to MSN(40) by 0.002, 0.002 and 0.004 for ATM, ITM and DITM options respectively.

Although the differences in price approximation when comparing MLP-2L and

MSN are rather small, but still favourable to MLP-2L, apart from the price output it is important to see if the networks are properly learning the correct overall behaviour of the option pricing function and not over-fitting to just the price. In fact it is possible that the price unction could be approximated quite accurately while it derivatives an insufficiently predicted. This can be validated by looking at the derivatives of the neural network outputs with respect to inputs, which correspond to the option values known as the Greeks.

5.3.2.3 Greeks

Figures 5.3, 5.4 and 5.5 show the absolute relative error for the option price sensitivities known as the Greeks, Delta (Δ), Vega (v) and Rho (ρ), for the MLP-2L(40) and MSN(40) architectures respectively; these figures show the log10(ARE) for all the separate independent networks. The AREs for the MC test set pricing (Figure a in each case) are also shown to provide a benchmark for comparison with respect to the numerical calculation.

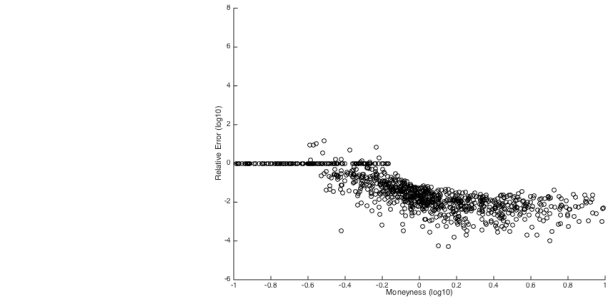
The most important Greek value is the Delta, as this is most commonly used to hedge portfolios. The MLP-2L(40) shows the more robust Delta calculation, with only a few examples exceeding a 100% relative error. The profile of these Delta values closely resembles that of the MC data, and in fact in some cases for ATM and some ITM options shows more accurate Delta values. The story is quite different for the MSN network and it can be seen that the Delta calculations are very unreliable in comparison to the 2-Layer MLP.

For the Vega both architectures show comparatively poor performance with respect to the MC values.The MC Vega has a maximum error of around order 10^2, whilst it is seen that the error for ITM options for both neural networks lie at around a magnitude between 10^5-10^8. For OTM options the MLP-2L(40) shows slightly better errors with less frequent extremely high errors. In addition some cases for options around ATM there are instances where the error is significantly better than MC, with a best error of around order 10^{-3}.

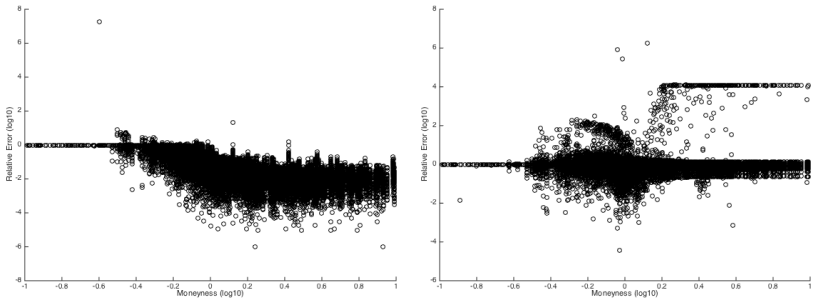
In the case of the Rho both MSN and MLP struggle to acheive the level of error given by MC. For Rho the MSN(4) architecture shows overall larger errors,

especially for OTM options compared to the MLP(40).

Even though the MSN network produces more reliable pricing outputs for DOTM and OTM options, the accuracy and nature of the solution is questioned given the errors of the derivatives. The MLP-2L network produces more robust derivative approximations, which is an indication that it has more accurately learnt the overall pricing function. As a result the MLP-2L(40) network is chosen to be used throughout the rest of this work. The following section will look at training data issues for this chosen network.

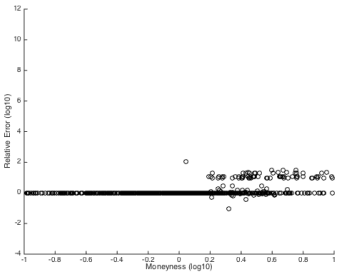


(a) MC (Res=10^{-6})

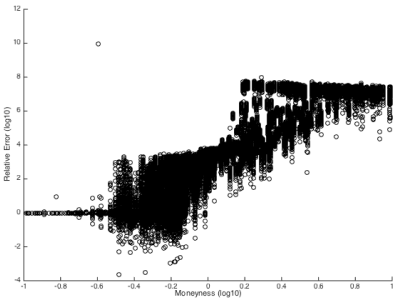


(b) 2-Layer (20N,Res=10^{-6}) **(c)** MSN (20N,Res=10^{-6})

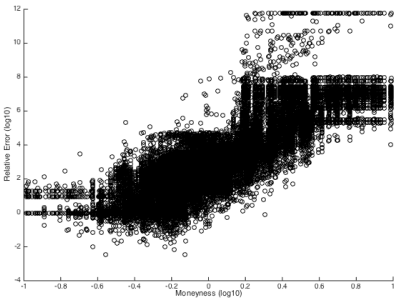
Figure 5.3: Delta



(a) MC (Res=10^{-6})

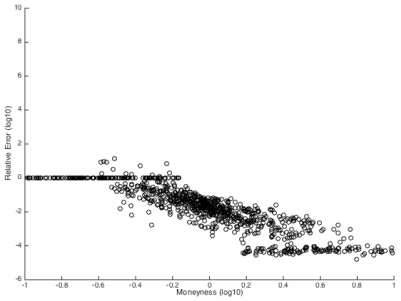


(b) 2-Layer (20N,Res=10^{-6})

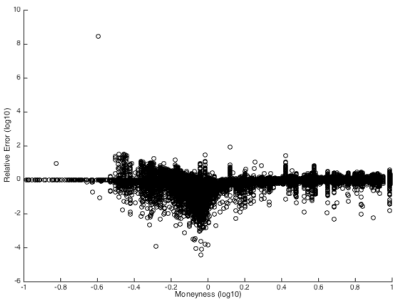


(c) MSN (20N,Res=10^{-6})

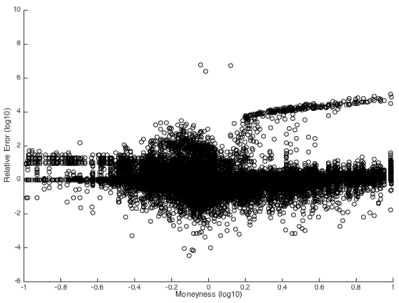
Figure 5.4: Vega



(a) MC (Res=10^{-6})



(b) 2-Layer (20N,Res=10^{-6})



(c) MSN (20N,Res=10^{-6})

Figure 5.5: Rho

5.3.3 Training Data Sensitivity

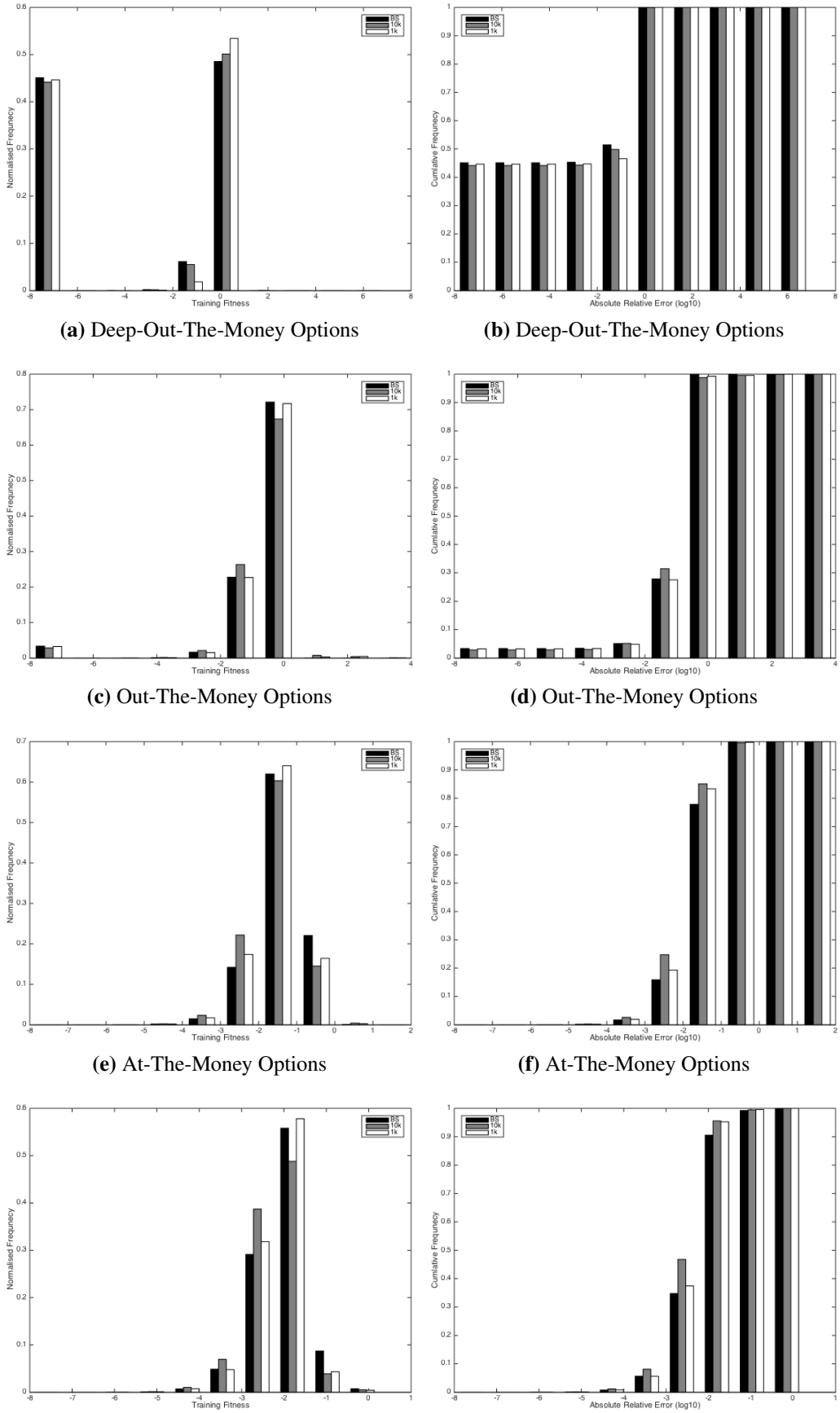
5.3.3.1 Noise

One source of potential error is in the noise from the training data, using MC simulations as the training data means there will inherently be some noise in the data, and this could effect the final function approximation. To test for effects of noise in the training data two additional sets of 300 MLP-2L(40) networks were trained, one set using the exact Black-Scholes solution and the other set using a more corse MC(1000,365) for the same training data samples previously used for the MC(10000,365) training data.

The ARE distributions are shown in Figure 5.6. The most interesting result is that the BS trained models are actually worse with a lower cumulative frequency of lower AREs than the MC(10000,365) trained models for all but DOTM options. Infact even for ITM and ATM options it can be seen from the cumulative distributions that MC(1000,365) also has a higher frequency of lower AREs than the BS training data, this in contrary to what may be expected in that increasing the accuracy of the training data leads to more accurate approximations.

There are a few possible explanations for this behaviour, essentially when training with MC data it is similar to the technique known as output smearing [224] where gaussian noise is added to the target training values. The concept behind output smearing is similar to other ensemble methods with respect to the bias-variance decomposition which aims to increase the variance without effecting the bias and thus increasing model diversity. In output smearing it is shown that the variance can be decomposed into two parts: $V_{\mathrm{O}}(f)$, the variance due to the output, and $V_{\mathrm{I}}(f)$, the variance due to the sample inputs. It was observed that the output variance was the dominant source of prediction error.

From the bias-variance decomposition it can be deduced that given the training data is exact there is low variance but as the trade-off would suggest, results in a high bias. Feedforward neural networks are a bias estimator [225], and given the presence of no-noise in the training data the bias component is the most prevalent. This implies that shape of the BS surface is difficult for a single neural networks



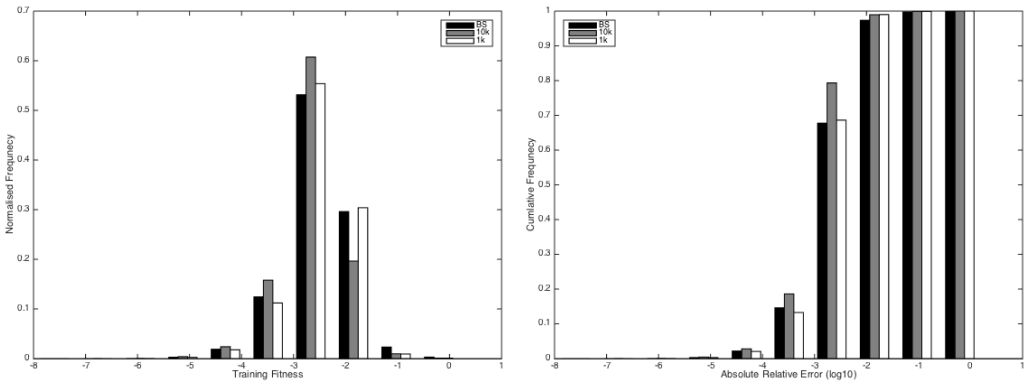
(a) Deep-Out-The-Money Options **(b)** Deep-Out-The-Money Options

(c) Out-The-Money Options **(d)** Out-The-Money Options

(e) At-The-Money Options **(f)** At-The-Money Options

(g) In-The-Money Options **(h)** In-The-Money Options

Figure 5.6: Distribution (LHS) and cumulative distribution (RHS) of absolute relative price errors of the MLP-2L networks trained using 3 different levels of accuracy for the training data, the exact Black-Scholes solution, and Monte-Carlo using 1000 and 10,000 replications.



(i) Deep-In-The-Money Options **(j)** Deep-In-The-Money Options

Figure 5.6: cont. Distribution (LHS) and cumulative distribution (RHS) of absolute relative price errors of the MLP-2L networks trained using 3 different levels of accuracy for the training data, the exact Black-Scholes solution, and Monte-Carlo using 1000 and 10,000 replications.

to estimate the entire generalised function, in particular the non-linear behaviour of the ATM and ITM options, if the bias of the estimator is too high it will be hard for it to correctly capture the curvature present in these regions. Whilst when using noisy MC data although this increases the variance the additional diversity as a result allows some neural networks to more accurately interpolate in specific regions of the data which gives rise to the better ARE distributions.

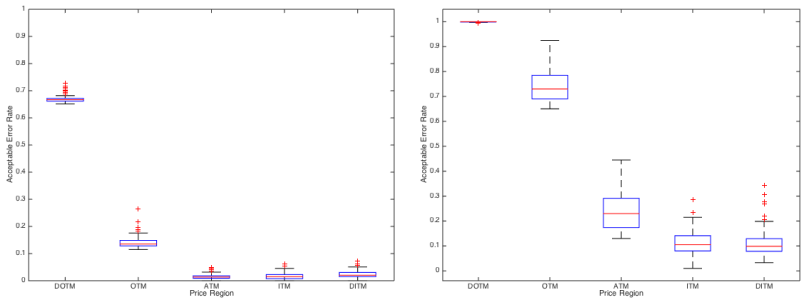
It has been observed that although one may expect the most accurate training data to result in the best function approximations due to the complexity of the function being approximated the bias of the neural network estimations become too dominant, as a compromise a low level of noise is required in the data to reduce the bias at the cost of adding some variance, another way of reducing the bias is to increase the density of the training data.

5.3.3.2 Increasing Training Data Density

To try and further improve the precision of the price approximations the size of the training data has been increased to 5000 training samples over the domain Ω. Using this new training data 100 independent MLP-2L(40) neural networks have been trained. Although using more training samples has the disadvantage that it considerably increases the training time of neural networks, the significant improvements

in the accuracy of the approximations more than compensates for this.

The improvements by increasing the size of the training data most obviously been seen when comparing the acceptable error rates for each region, Figure 5.7, with significant increases in accuracy over all regions. In particular there is a large overall improvement for all networks within the DOTM and OTM regions where the minimum acceptable error rate for the neural networks trained using 5000 samples are 0.99 and 0.6500 respectively, compared to the maximum acceptable error rate in the same regions using 2000 samples which are 0.72 and 0.26 respectively. For ATM, ITM and DITM options the acceptable error rates are all below 0.05 when using 2000 training samples, whilst for 5000 training samples the range is a lot larger for all regions and extending well above 0.1, in particular the ATM rates extend up to around 0.45.



(a) 2K training samples **(b)** 5K training samples

Figure 5.7: Boxplot of the acceptable error rates for neural networks trained with 2000 and 5000 training samples.

These results empirically show that it is more beneficial to increase the number of training samples rather than increasing the accuracy of the training data as previously discussed. This can most simply be illustrated by approximating a curve with piece-wise linear functions, using one linear function formed from two training points approximates this as a straight line no-matter the accuracy of the two training points with respect to the curve, then as the number of training points increases more of the curvature can be captured. With respect to the bias-variance tradeoff using more training samples reduces the bias of the model, but eventually this will need to be compensated by a reduction in the variance via smoothing between the

noise of the points.

In light of this marked improvement in accuracy, as shown in Figure 5.7, the final methodology presented here uses the MLP-2L(40) networks trained using 5000 samples to create the final neural network model, the final model is created either by model selection or ensemble methods.

5.3.4 Model Diversity

Ensemble models provide a significant improvement over single models but the diversity of the neural network models is important. Low diversity would mean that the variance of the models around the bias is low, which gives rise to a set of very similar and highly biased set of models. There needs to be a tradeoff between diversity and accuracy, with sufficient diversity and accuracy the neural network outputs should be compactly well distributed around the target value [226].

Training Fitness

The main source of variance of the neural network models is generated by the stochastic search algorithm (BrPSO) used for learning the parameters; assuming that there exists a unique solution of the parameters; to model the neural network the variance would exist as noise around the global optimum.

Figure 5.8 shows the distribution of fitness values for the three training methods: unwieghted; weighted-random; and weighted-atm. It can be seen that the un-weighted training follows a lognormal-like distribution, this shows that there is a higher concentration of models around the better fitness values, which based on the above assumption of convergence to a global optimum should occur. Despite this showing good convergence it indicates a limitation in the diversity of the models and there is no guarantee that a model with a low training error will have the best generalisation [227]; this has previously been shown in the lack of correlation between training fitness and the acceptable error rate of each region.

The two other training methods, weighted-random (wt-rand) and weighted-atm (wt-atm), show a higher variance in the magnitude of fitness values, and as may be expected wt-rand exhibits the overall largest degree of variance. For wt-atm the

distribution exhibits some degree of lognormality, with a mean at $\log(\text{fit}) = -2.5$, but still shows than even though the optimisation is more constrained by weighting around a specific region there is still a large enough degree of freedom to train a wide range of models. This supports the idea that the weighted training encourages diversity and expertise in different regions.

However, using the fitness value as a measure of diversity presumes this is correlated to the distribution of the outputs which may not be true, for example consider how the same fitness value can be achieved by minimising the error of different target values but to the same magnitude; this creates one fitness value but a set of highly diverse models. As such more robust measures of diversity can be used.

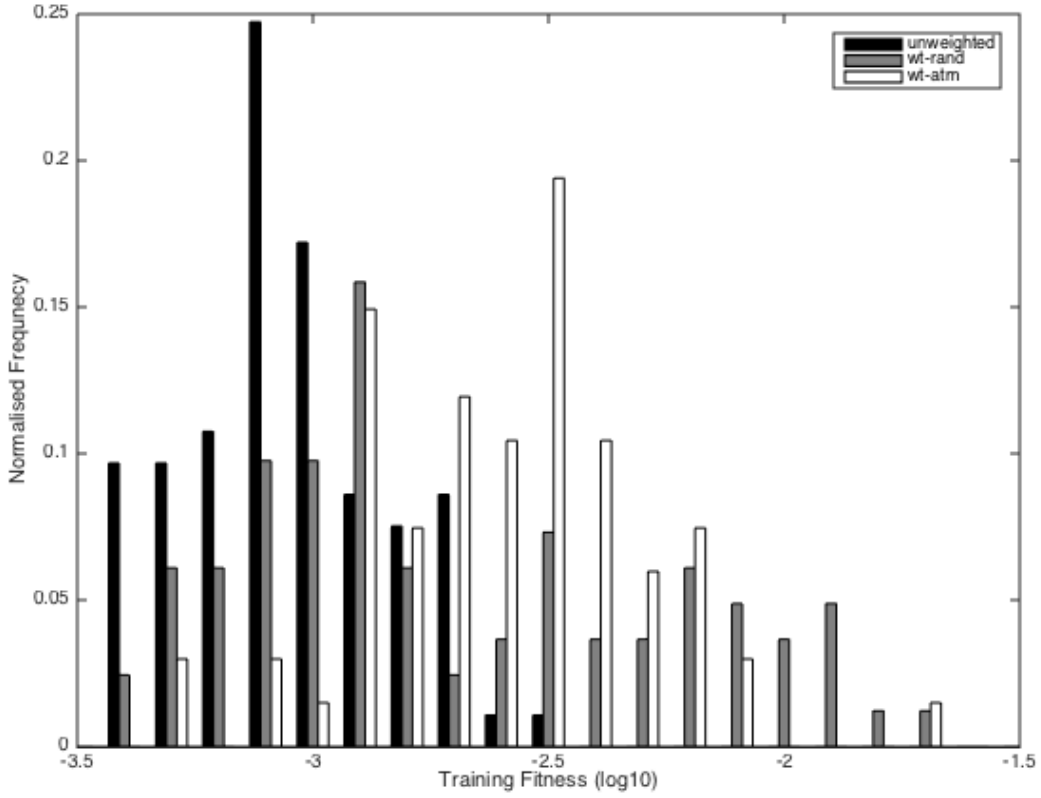


Figure 5.8: Training fitness distributions.

Diversity and Mutual Information

Two other measures based on the output values of the neural networks can be used to evaluate the diversity of the neural network models. Diversity can be measured with respect to mutual information between the ensemble members [228]; in negative correlation learning (NCL) [220] the diversity of the ensemble is measured as

$$D_{\mathbf{M}} = \sum_i \sum_j \left(\hat{y}_i^j - \hat{y}_M^j\right) \sum_{k \neq i} \left(\hat{y}_k^j - \hat{y}_M^j\right) \tag{5.36}$$

where $\hat{y}_i^j$ is the output of the ith ensemble member for the jth training sample output, and $\hat{y}_M^j$ is the ensemble output for training sample j taken using the mean. This can be interpreted in the sense that $D_{\mathbf{M}}$ measures the balance of the mean for an ensemble estimator; for an individual component a large negative value will only occur if the signs of the two components are different and signifies that the there is a balance between positive and negative errors centred around the mean output. It has been shown that this quantity can be related to the ambiguity decomposition [229].

One issue with this measure, $D_{\mathbf{M}}$, is the assumption the variance is constant; this data includes, as has been pointed out, a vast range of magnitudes, and to try and give a more balanced view of diversity over this data set the *relative-diversity* is introduced

$$D_{\text{rel}} = \sum_i \sum_j \frac{\left(\hat{y}_i^j - \hat{y}^j\right) \sum_{k \neq i} \left(\hat{y}_k^j - \hat{y}^j\right)}{\hat{y}^j + \text{res}} \tag{5.37}$$

where the resolution of the model training is added to compensate for any rounding up to zero made.

Closely related to the preceding diversity measures, $D_{\mathbf{M}}$ and D_{rel}, is the mutual information between two ensemble members. Assuming the outputs are Gaussian random variables this is measured as

$$I\left(N_i; N_j\right) = -\frac{1}{2} \log\left(1 - \rho_{ij}^2\right) \tag{5.38}$$

where $\rho_{i,j}$ is the correlation coefficient. The domain of the mutual information is $I(N_i;N_j) \in [\infty,0]$, where positive values indicate there is a significant degree of mutual information $I > 0.5$ implies $\rho^2 > 0.9$, moderate values of $I \approx 0.10$ imply $\rho^2 \approx 0.5$, whilst a value of 0 indicates no mutual information. From this the mean mutual information of the set of neural networks can be measured as

$$I(\mathbf{N}) = \frac{\sum_i \sum_{j>i} I\left(N_i;N_j\right)}{0.5Z^2 - Z} \tag{5.39}$$

where Z is the number of neural networks in the set. Methods such a NCL aim to minimise the mutual information; minimising the mutual information is the same as minimising $\mathrm{Cov}(N_i,N_j)^2$, to produce a set of uncorrelated models. One issue as noted by Clemen *et al* [230] is that the mutual information does not take into consideration negative correlation, in the case of ensembling negative correlation is a positive behaviour that can assist the ensemble by reducing the variance.

Diversity of Trained Models

Table 5.2 shows the diversity measures for the sets of neural networks (unweighted, wt-rand, and wt-atm) within each price region (DOTM, OTM, ATM, ITM and DITM) and all the regions combined (ALL). As expected the wt-rand networks show the largest degree of diversity over all regions for both diversity measures, whilst wt-atm networks show the smallest overall diversity. The wt-atm and unweighted networks are generally quite similar; for ITM and DITM wt-atm has a larger diversity, the relative-diversity helping to make the difference more explicit. The relative-diversity measures are more balanced over all regions, compared to the regular diversity measures, $D_{\mathbf{M}}$, which are dominated by the ATM, ITM and DITM values.

The MI was measured with respect to the mean-centered residues to make the outputs for each network more distinguishable. Table 5.3 gives the measured mutual information for each of the sets of neural network models for each of the price regions. For the whole domain of price regions all sets of networks show a high degree of MI, with the wt-atm being marginally lower; this could be attributed

to the significantly lower MI for DOTM and DITM options which make up a large proportion of instances. In general it can be seen that there is a moderate/high level of mutual information in the OTM, ATM and ITM regions, whilst DOTM shows a high degree of MI, and DITM shows a lower degree of MI. Given the degenerate nature of the option price function within DOTM a high degree of MI is to be expected, wt-atm shows the lowest degree of MI in this region, which for an unbiased estimator should imply a more accurate approximation when using the mean to create an ensemble. In addition, the wt-atm set has the lowest MI for OTM and DITM; for OTM options it is similar to the unweighted-training set.

An interesting observation is that ATM options show a higher degree of mutual information than ITM and DOTM. ATM options have the highest level of function convexity and sensitivity to the input parameters and therefore should be the hardest area to approximate, this is expected to give rise to a lower degree of mutual information and more diversity compared to other price regions. In the case observed for ATM options a higher degree of diversity combined with a higher degree of mutual information shows that there is a higher degree of negative correlation within ATM options, which may make ensemble methods more accurate in this region.

Overall the results shown in Table 5.2 and 5.3 indicate that there is some diversity within the sets of neural networks (unweighted, wt-rand, and wt-atm) but the relative diversity measures show that it is still very small. The small measure of diversity combined with the high and moderately high values of mutual information shows that the models are mostly positively correlated which can make finding the optimal ensemble a harder problem, but at the same time that similar functions have been learned by the networks which raises confidence in the approximation ability of the networks.

Diversity ($D_{\mathbf{M}}$)			
	DOTM	OTM	ATM
Unweighted	-1.14e-07±3.25e-07	-1.90e-05±8.93e-06	-3.66e-04±8.15e-05
wt-rand	-6.37e-07±5.62e-06	-9.35e-05±1.82e-04	-1.09e-03±4.58e-04
wt-atm	-1.02e-07±2.31e-07	-9.41e-06±6.09e-06	-3.40e-04±1.20e-04
	ITM	DITM	ALL
Unweighted	-1.10e-03±1.75e-04	-1.26e-03±2.20e-04	-5.90e-04±1.63e-04
wt-rand	-4.75e-03±1.32e-03	-7.52e-03±2.42e-03	-3.22e-03±1.66e-03
wt-atm	-2.77e-03±5.43e-04	-6.10e-03±5.99e-03	-2.45e-03±3.58e-03

Relative-Diversity (D_{rel})			
	DOTM	OTM	ATM
Unweighted	-1.61e-05±2.60e-05	-5.03e-04±1.78e-04	-2.02e-03±3.85e-04
wt-rand	-6.01e-05±3.21e-04	-1.91e-03±2.96e-03	-5.73e-03±2.15e-03
wt-atm	-2.26e-05±2.24e-05	-3.11e-04±1.23e-04	-1.50e-03±4.64e-04
	ITM	DITM	ALL
Unweighted	-1.82e-03±3.03e-04	-4.36e-04±8.59e-05	-5.93e-04±1.97e-04
wt-rand	-7.27e-03±1.87e-03	-2.30e-03±7.18e-04	-2.32e-03±1.48e-03
wt-atm	-4.34e-03±8.62e-04	-1.79e-03±1.20e-03	-1.25e-03±8.06e-04

Table 5.2: Diversity measures, the regular diversity, Equation 5.36, and relative-diversity, Equation 5.37, of the three sets of trained neural network models: unweighted; wt-rand; wt-atm.

Mutual Information (MI)			
	DOTM	OTM	ATM
Unweighted	4.31e-01±4.74e-03	2.00e-01±2.20e-03	1.72e-01±1.89e-03
wt-rand	4.50e-01±5.62e-03	3.13e-01±3.92e-03	1.32e-01±1.66e-03
wt-atm	2.83e-01±4.35e-03	1.89e-01±2.91e-03	1.97e-01±3.03e-03
	ITM	DITM	ALL
Unweighted	1.41e-01±1.55e-03	6.38e-02±7.02e-04	5.26e-02±5.78e-04
wt-rand	1.23e-01±1.54e-03	6.95e-02±8.69e-04	5.12e-02±6.40e-04
wt-atm	1.43e-01±2.20e-03	5.74e-02±8.83e-04	4.74e-02±7.30e-04

Table 5.3: Mutual information, Equation 5.39, of mean-centred outputs for the three sets of trained neural network models: unweighted; wt-rand; wt-atm.

5.3.5 Ensemble Models

For each of the sets of trained neural network models (unweighted, wt-rand, and wt-atm) ensemble methods are used to produce an overall approximation of the options pricing function. Three ensemble methods are investigated here: mean/equally weighted ensemble; dynamic centre-distance; linear regression. It is found that linear regression produces the most robust overall approximation and is adopted as the final ensemble method used here.

5.3.5.1 Mean Models

The simplest ensemble strategy is to use the mean, which is equivalent to taking equal linear weights of each neural network model in the set. The absolute error (AE) for the three sets of trained neural network models are illustrated in Figure 5.9 and the respective acceptable error rates (AER) are given in Table 5.4. Compared to the individual models, Table 5.5 gives the AERs for the best selected individual model for each price region, even this simple ensemble methods shows significant improvements on the robustness and overall accuracy of the price approximations.

For individual models the best DOTM AER is around 0.7 (70%), which is similar for most of the other models in all of the sets; for the mean ensemble this increases to an AER of 100%. For all the sets of neural networks there is a significant increases in the AER for OTM options, increasing from around 18% up to over 75%, with wt-atm showing the best AER of 100%. This vast improvement by taking the mean shows that the outputs of the neural network in each set are well distributed around the mean. However, it should be noted that the magnitude of the AER relative to the magnitude of prices in the DOTM and OTM regions are respectively large so this does not necessarily imply that the mean is very accurate and the neural networks may still be a set of biased estimators.

Increases in the AER are also seen for all the other other price regions, where for the individual best models the AER is around 5% for ATM, ITM and DITM, which is seen to increase to around 20% for the unweighted and wt-rand ensembles. The wt-rand ensemble is slightly worse than the unweighted ensemble showing a

lower AER for ITM options. It can be seen in Figure 5.9 that the unweighted and wt-rand show large peaks for OTM, ATM, and ITM prices with respect to the MC price approximations.

The wt-atm ensemble shows the best results, unsurprisingly the wt-atm model shows an extremely good AER of 89% for the ATM region. This illustrates that training a set of expert models in a selected region can results in better performance; though it would then be expected that wt-rand would perform as well as wt-atm, which is not the case here; this is due to too much diversity in the wt-rand set and there being not enough models specialising around particular regions for the benefits of the localised expert models to show. For the wt-atm training it can be seen that this also improves the performance for the OTM and ITM results either side of the ATM price region, this is due to the weighting at ATM helping to reduce the the degrees of freedom of the function approximations in the surrounding regions. For the regions further away, DOTM and DITM is can be seen that the performance of all three ensembles are very similar.

For ATM options the unweighted and wt-atm sets, both show similar degrees of diversity with respect to the measures presented in Section 5.3.4, but wt-atm shows significantly better accuracy within this region. The good performance of wt-atm along with the lower diversity and larger mutual information of the set implies that the wt-atm neural networks are better unbiased estimators within the ATM region, and least bias estimators throughout the other price regions.

Although there is an overall a significant increase in accuracy for all the mean ensemble models it can be seen in Figure 5.9 that there is a distinct cluster of very high errors within the ATM region. This is due to the effects of small time-to-maturity and the discontinuous payoff function, this area was seen to exhibit the highest variance, and the issues within this area will be discussed in more detail later on in Section 5.3.6.

The mean in simple method and when combined with the right set of models can produce a very accurate approximation. The issue when using the mean is that it follows the bias of the training data, so it is unlikely that it will perform better

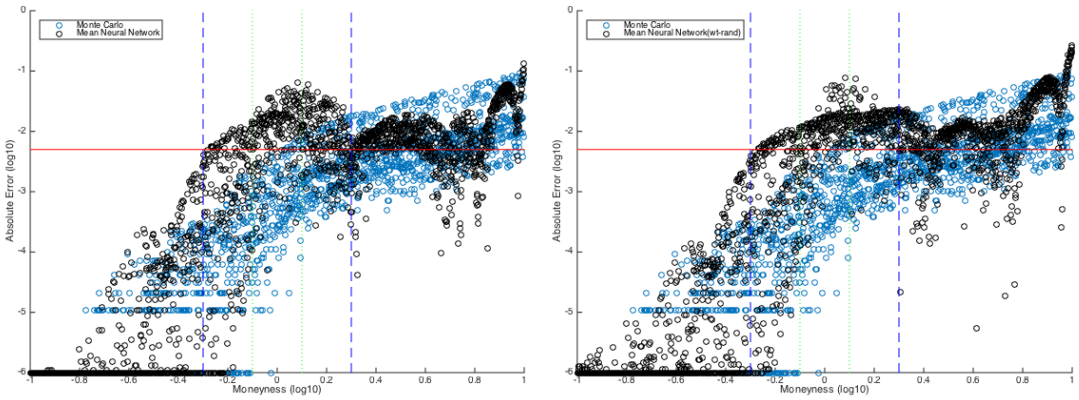
	DOTM	OTM	ATM	ITM	DITM
MC	1.00	1.00	0.96	0.77	0.46
unweighted	1.00	0.76	0.23	0.38	0.25
wt-rand	1.00	0.73	0.20	0.14	0.23
wt-atm	1.00	1.00	0.89	0.51	0.22

Table 5.4: Acceptable error rates for the mean models.

	Price Region				
Best model	DOTM	OTM	ATM	ITM	DITM
Min Fit	0.735	0.182	0.018	0.023	0.016
DOTM(43)	0.739	0.149	0.009	0.015	0.004
OTM(251)	0.735	0.182	0.018	0.023	0.016
ATM(271)	0.727	0.149	0.045	0.008	0.016
ITM(144)	0.727	0.128	0.009	0.045	0.016
DITM(78)	0.731	0.122	0.009	0.000	0.047

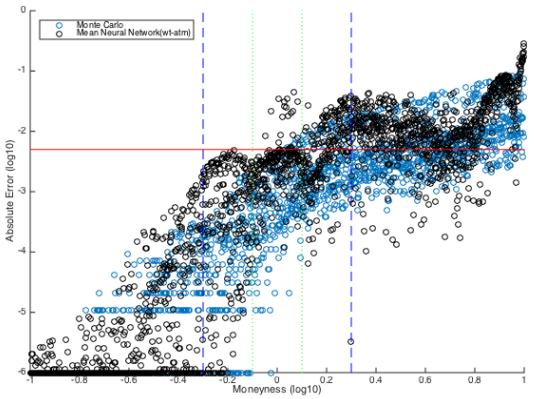
Table 5.5: The acceptable error rates for the best model for each of the price regions.

than the MC approximations, as this would imply that the neural network models begin to deviate from the training data. This issue was partly addressed with respect to training data noise, where it was suggested that more noise can allow for better interpolation of the training data due an increase in the degrees of freedom. Another manner for reducing error post-training is using regression techniques to add extra corrections to the ensemble outputs.



(a) Unweighted Training

(b) Random Center Training



(c) ATM Center Training

Figure 5.9: Scatter plots showing the absolute relative error for the 10 median-aggregated neural network European pricing models for each of the neural network architectures explored, 2-Layer (20N,Res=10^{-6}).

5.3.5.2 Centre-Distance

The center-distance weighting model, in contrast to the other two methods used, is a dynamic aggregation method; instead of a static set of weights that is generated a-priori, for center-distance weighting the linear weights are generated dynamically based on the test inputs given (M,σ,r). This method is only applied for the wt-rand models because all the networks were trained with difference centers distributed across the range of monyeness inputs.

For the center-distance weighting model the set of weights, $\beta_{\mathbf{i}}$ for an input instance i, is generated as the normalised distance between the neural network model

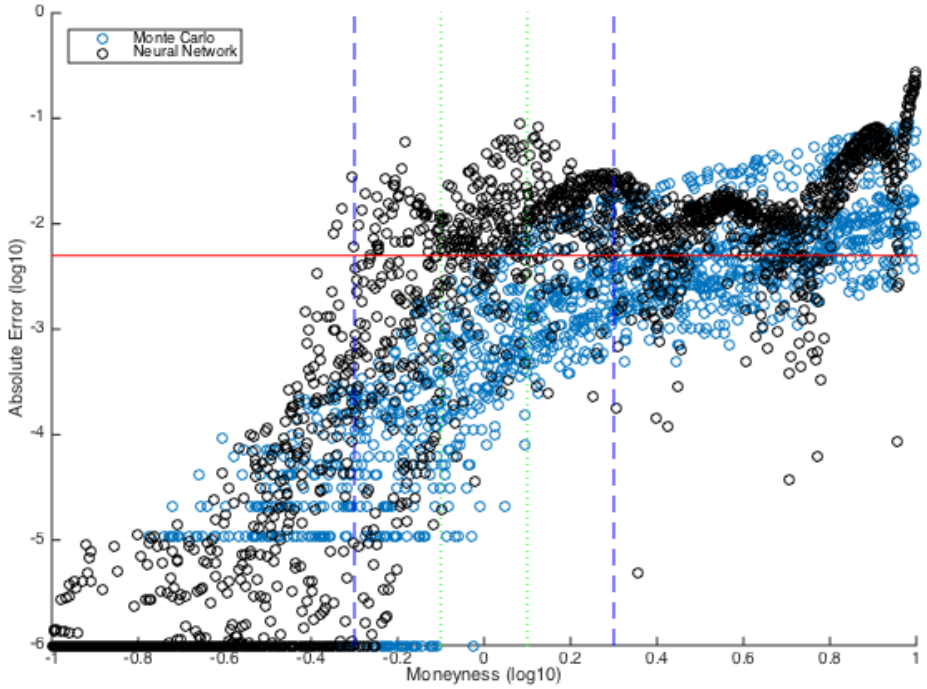


Figure 5.10: Absolute errors for the center weight ensemble using the wt-rand set of neural networks.

DOTM	OTM	ATM	ITM	DITM
0.993	0.720	0.315	0.130	0.174

Table 5.6: Acceptable error rates for the center weight ensemble using the wt-rand set of neural networks.

center C_N and the test input of the moneyness M_i. For a neural network model N

$$\beta_{i,N} = \frac{(\|C_N - M_i\| + a)^{-2}}{\sum_{j=0}^{Z} (\|C_j - M_i\| + a)^{-2}}. \tag{5.40}$$

where a is constant for determining the radius of influence, i.e. for all $\|C_N - M_i\| < a$ it reduces the variance in the magnitude of the weights, such that they are all approximately equal; this stops one model having an overwhelming contribution if it is extremely close to the given center M_i, as some diversity is still required. This results in the neural network with the closest center to the input, subject to the radius of influence, having the highest weighting as it is expected that during training it specialised most in this area and produces the best expert model for the given input instance.

However, it can be seen from Table 5.6 and Figure 5.10 that the centre-distance ensemble model exhibits disappointing results, and shows performance worse than the mean ensemble, especially for OTM and ATM options. This is either due to the neural networks not specialising well enough for the given training centers, although it was observed for the wt-atm set that the neural networks specialised well for the given region of interest (ATM), which then implies that a larger number of networks is required to provide a sufficient number of experts for the input region of interest. It may also be worth reducing the range of the centers, instead of having a continuous range over the whole input space, a discrete set of particular areas of interest are chosen for each network to then be randomly assigned as the center, this will also allow for a larger number of experts to be trained for each given region.

5.3.5.3 Regression Models

Linear regression methods are used to try and create improved linear ensembles; both convex $\beta_i > 0$, and non-convex $\beta_i \in \mathbb{R}$ weights are considered. Shortest-path linear least-squares (non-convex) and constrained linear-least squares (convex) have been used to generate the linear ensemble weights for each set of neural network models (unweighted, wt-atm and wt-rand), see Section 5.2.3.1 for more details. This variant proved to be the most effective of the three ensemble methods considered, and the discussion in this section be correspondingly more detailed.

Table 5.7 gives the acceptable error rates (AER) for both convex and non-convex weights and the absolute error plots are given in Figures 5.11 and 5.12 respectively. First of all it is noticeable that the non-convex weights result in considerably better acceptable error rates, and from this, it can be implied, overall errors.

Convex Weights

Compared to the mean/equally weighted ensembles the convex weights show increases for all but the unweighted-ITM and wt-atm ATM acceptable error rates. This shows that in general either the approximation errors are not symmetrically distributed around the mean and/or that the neural networks are biased estimators.

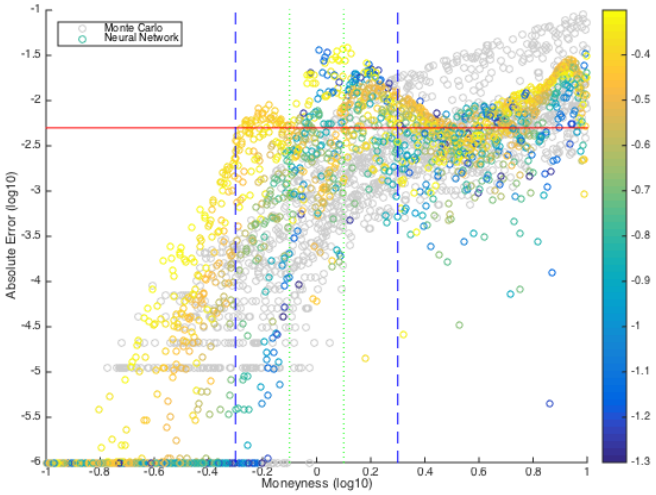
The wt-atm ensemble performs the best, compared to the other two sets of

	DOTM	OTM	ATM	ITM	DITM
Unweighted					
Convex	1.000	0.990	0.970	0.955	0.577
Non-Convex	1.000	0.830	0.635	0.295	0.446
wt-rand					
Convex	1.000	0.995	0.910	0.915	0.570
Non-Convex	0.999	0.730	0.410	0.670	0.516
wt-atm					
Convex	1.000	0.920	0.895	0.955	0.554
Non-Convex	1.000	0.990	0.780	0.700	0.411

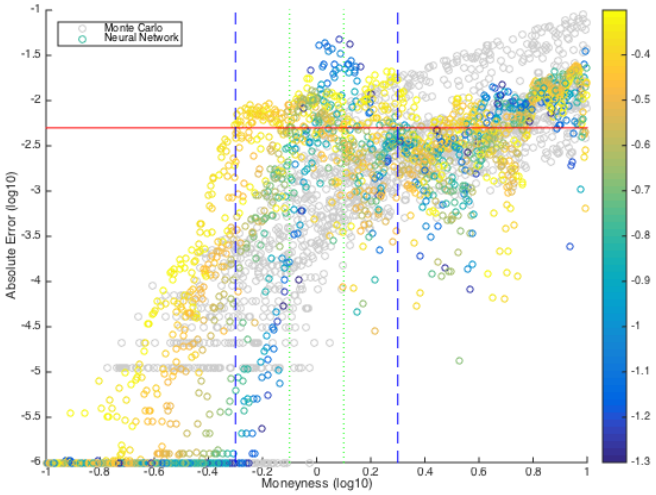
Table 5.7: Acceptable error rates for the linear regression ensembles using convex and non-convex linear combination weights for each of the three sets of trained neural networks: unweighted, weighted-random (wt-rand), and weighted-at-the-money (wt-atm).

neural networks, when using convex weights; as previously suggested this implies that the neural network estimators are less bias and the overall approximations to the target function are more accurate. It also implies that there is more diversity and less collinearity between the all the models in the wt-atm set, or at least for a selected subset, than for the two other sets of neural networks. It is also interesting to see that wt-atm ATM AER is lower using the linear regression weights compared to the equally weighted mean; this further supports the notion that wt-atm neural networks are less biased and that the error distributions are close to symmetric with a zero mean for ATM price approximations.

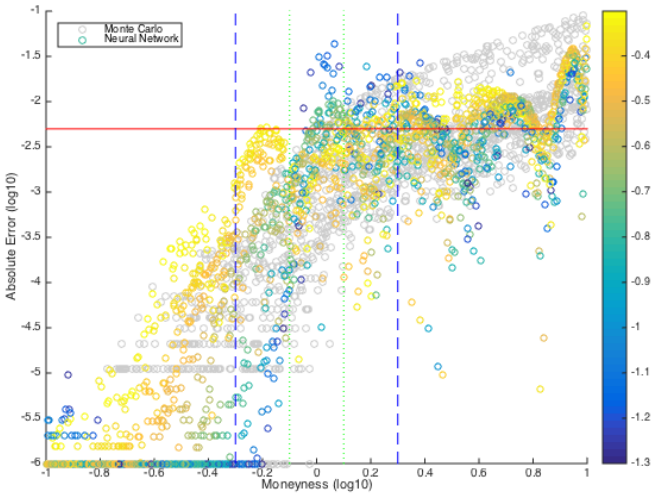
The wt-atm convex ensemble show AERs that are very similar to Monte Carlo, however from Figure 5.11 it can be seen that the convex-linear ensemble struggles with larger errors for OTM with high volatility and ATM with low volatility; this is a similar trend for all of the sets of neural networks, and effects of volatility will be discussed in more detail later on. The results for wt-atm show that the convex ensemble has good potential to match or outperform Monte Carlo price approximations, in Section 5.3.5.4 data transforms will be used to further increase the accuracy of the convex weighted models.



(a) Unweighted Training



(b) Random Center Training



(c) ATM Center Training

Figure 5.11: Scatter plots showing the absolute error for the convex linear neural network ensembles using the three different weighted trained methods. The volatility, σ, of each sample is represented as the colour depth using a $\log_1 0$ scale.

Non-Convex Weights

For the non-convex weights the acceptable error rates outperform or are very close to the MC price approximations considered here for all price regions. It is interesting to see that the presence of negative weights among the non-convex weights is able to considerably increase the accuracy of the ensemble models; a more detailed analysis further on will justify why negative weights, not usually recommended, are applicable in this situation. From the acceptable error rates shown in Table 5.7 it is hard to distinguish which ensemble model is the best; in terms of generalised performance across all prices regions with respect to the level of accuracy of the acceptable error boundary (AEB), they are all exceptional, but as mentioned this generalisation comes at the cost of lower specific accuracy if focusing on a particular price region.

To give a more detailed look at each of the ensemble models performance Table 5.13 gives the cumulative distributions of the absolute relative price errors (ARE) for each region; in addition the column of the relative error approximately corresponding to the AEB for each region is highlighted. Beginning with DOTM and OTM options it can be seen that with respect to the ARE the MC results are more accurate, especially for OTM options. For OTM options the same percentage of prices have an accuracy of $\leq 1\%$ for MC as the neural network models have for $\leq 10\%$, it is only in the case of ARE$\leq 30\%$ that the MC and neural networks are comparable in accuracy. These differences are not apparent in the acceptable error rates, but are significant, which is why it is important to now consider this finer level of detail to decide how well the ensemble models compare with MC.

The main improvements in the AER when comparing the non-convex ensembles to the convex and mean ensembles are for the ATM and ITM price regions. For ATM options the non-convex neural network ensembles show a similar degree of accuracy, with respect to the AER, as MC, although wt-atm is the worse with an AER of only around 90%. Even though the AERs are only slightly lower than MC it can be seen for the AREs that the ensembles are overall more accurate. For ARE$\leq 0.1\%$ MC has frequency of 13.5% whilst the neural networks are still over

50%, with the best being the unweighted network with 70%. For ARE$\leq$ 0.03% the unweighted network and wt-atm neural network are around the same. It is interesting to see that even at this high degree of accuracy the unweighted network is more accurate than the wt-atm network for the non-convex combinations whilst for the mean ensembles the wt-atm vastly outperformed the other two sets; this shows that the non-convex weightings are able to correct a strong bias present in the unweighted models and will be further illustrated later on when considering the principle components.

For ITM options the AER is considerable better for all the ensembles than MC. Looking at the AREs for high degrees of accuracy, ARE $\leq 0.1\%$, all the neural networks and MC are roughly the same; for ARE $\leq 0.03\%$ the neural networks become more accurate at this performance level still retaining around 10% of prices errors, whilst the MC degrades to zero. The wt-atm performs the best overall in the ITM region, especially for the higher degrees of accuracy.

Finally, for DITM options the neural networks are slightly more accurate than MC, and even for high degrees of accuracy, ARE $\leq 0.1\%$, the neural networks still have over 60% compared to 50% for MC, although this degrades very quickly when moving to ARE $\leq 0.03\%$ to only around 17% for the neural networks but MC falls to 8%; for even higher levels of accuracy, ARE $\leq 0.03\%$, MC reduces to zero whilst neural network still retain around 5% of price errors.

Overall the neural network ensembles lack the same high degrees of accuracy for DOTM and OTM as seen for MC price approximations, but this is more than made up for in the accuracy of ATM and ITM options where the neural networks are seen to vastly outperform MC. Given the neural network ensembles follow roughly the same pattern of accuracy for each price region looking at the overall AREs (ALL) the unweighted performs slightly better overall and is comparable to MC (although this is due to good and bad performance in the different regions balancing), the wt-atm and wt-rand models have around the same performance although the wt-atm is more preferable for ATM and ITM options.

The most controversial finding here is that using a non-convex linear combi-

nation with negative weights vastly outperforms a convex linear combination; the negative weights are able to extract information from the ensemble that a convex combination cannot, contrary to the literature consensus negative weights can be extremely beneficial in certain situations.

Negative Weights

The use of negative weights is a controversial topic. They are often avoided not only because they are theoretically harder to justify but often result in less reliable ensembles [219]. The main theoretical argument against negative weights is that the space of the ensemble approximation becomes unbounded. First of all consider the constrained convex weights $\sum \beta = 1$, $\beta_i \geq 0$; in this case the ensemble approximation,$\hat{y}$, is bounded by the outputs of the individual models of the ensemble $\hat{y}_i$,

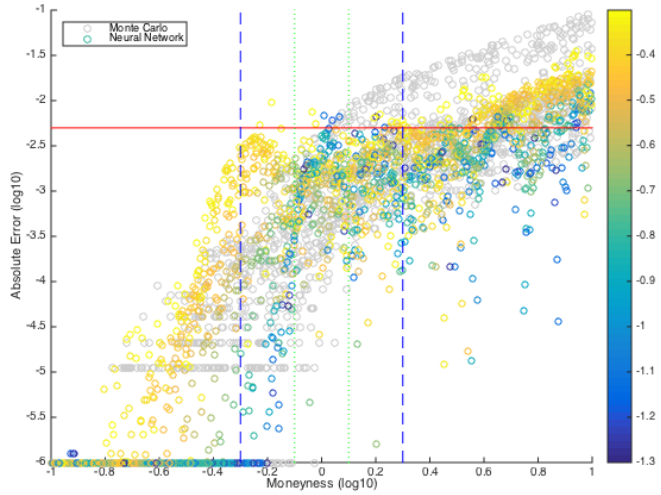
$$\min\{\hat{y_0}, \hat{y_1}, \dots\} \leq \hat{y} \leq \max\{\hat{y_0}, \hat{y_1}, \dots\}. \tag{5.41}$$

Now consider a non-convex linear weighting, even if the weights are constrained such that $\sum \beta = 1$, it means that the absolute size of the weights is now unbounded and it is possible to have $\beta_i > 1$ which can lead to the ensemble approximation now becoming unbounded. Consider a set of unbiased estimators, such that $\hat{y_i} = y + \varepsilon_i$, for a negative set of weights to remain bounded over the entire space it assumes that the conditional error distributions of $\Pr(\varepsilon_i|\varepsilon_j)$ is the same over whole space of the approximation, if this changes the cancelation of errors by the combination of negative and positive weights will not be the same and could result in explosive errors for the ensemble output, whilst in the convex case the worst case is limited by the worst of the individual models.

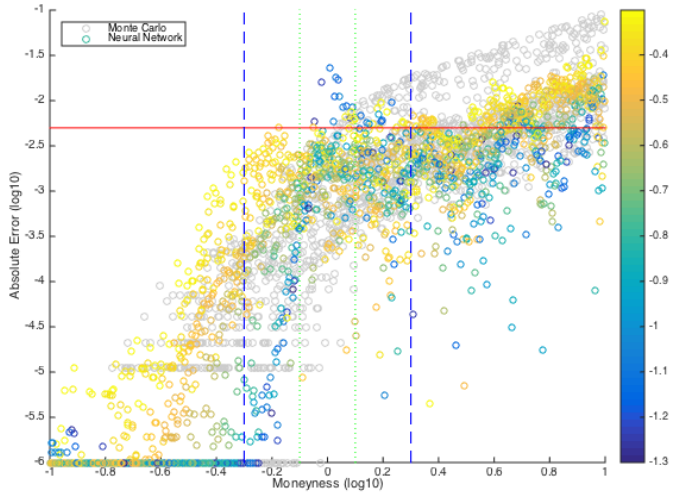
There are some advocates of negative weights [231], if the ensemble weights are related to the correlation coefficient then under the condition

$$\rho > \frac{\sigma_1}{\sigma_2} \; ; \sigma_1^2 = \min\left(\sigma_i^2, \sigma_j^2\right) \tag{5.42}$$

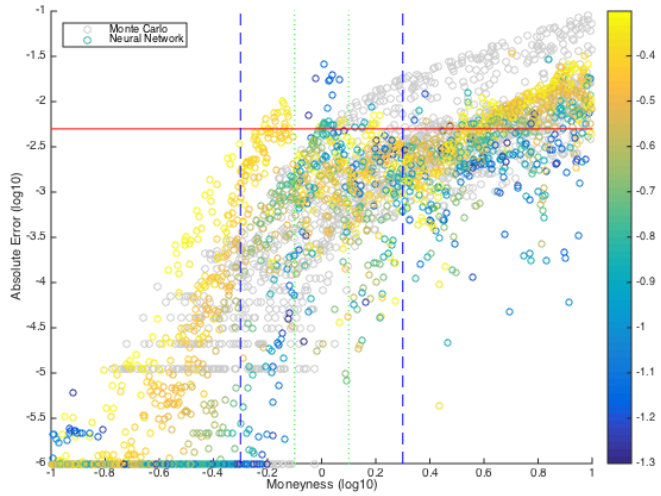
negative weights can be justified, this condition is indicative of high levels of collinearity. If assuming there is collinearity between members of the ensemble



(a) Unweighted Training



(b) Random Center Training



(c) ATM Center Training

Figure 5.12: Scatter plots showing the absolute error for the non-convex linear neural network ensembles using the three different weighted trained methods. The volatility, σ, of each sample is represented as the colour depth using a $\log_1 0$ scale.

	Absolute Relative Error										
	$\leq 0.001\%$	$\leq 0.003\%$	$\leq 0.01\%$	$\leq 0.03\%$	$\leq 0.1\%$	$\leq 0.3\%$	$\leq 1\%$	$\leq 3\%$	$\leq 10\%$	$\leq 30\%$	$\leq 100\%$
DOTM											
MC	0.774	0.774	0.774	0.774	0.774	0.777	0.796	0.870	0.930	0.990	1.000
unwt	0.683	0.684	0.684	0.684	0.693	0.696	0.707	0.741	0.827	0.934	0.997
wt-rand	0.617	0.617	0.619	0.619	0.619	0.619	0.633	0.693	0.834	0.976	1.000
wt-atm	0.597	0.597	0.597	0.600	0.600	0.609	0.626	0.677	0.806	0.977	1.000
OTM											
MC	0.195	0.195	0.195	0.200	0.235	0.335	0.635	0.920	0.980	1.000	1.000
unwt	0.080	0.080	0.080	0.080	0.090	0.115	0.190	0.365	0.600	0.915	0.970
wt-rand	0.025	0.025	0.025	0.025	0.035	0.040	0.115	0.350	0.690	0.920	0.960
wt-atm	0.060	0.060	0.060	0.060	0.065	0.080	0.135	0.240	0.580	0.915	0.975
ATM											
MC	0.005	0.005	0.020	0.045	0.130	0.510	0.815	1.000	1.000	1.000	1.000
unwt	0.005	0.010	0.050	0.210	0.705	0.980	1.000	1.000	1.000	1.000	1.000
wt-rand	0.010	0.010	0.025	0.125	0.525	0.915	0.990	1.000	1.000	1.000	1.000
wt-atm	0.015	0.045	0.070	0.210	0.560	0.925	0.985	1.000	1.000	1.000	1.000
ITM											
MC	0.000	0.000	0.000	0.000	0.305	0.745	0.940	1.000	1.000	1.000	1.000
unwt	0.005	0.010	0.035	0.110	0.370	0.890	1.000	1.000	1.000	1.000	1.000
wt-rand	0.005	0.015	0.050	0.135	0.405	0.790	0.995	1.000	1.000	1.000	1.000
wt-atm	0.000	0.020	0.085	0.165	0.425	0.900	1.000	1.000	1.000	1.000	1.000
DITM											
MC	0.000	0.000	0.000	0.081	0.541	0.827	1.000	1.000	1.000	1.000	1.000
unwt	0.007	0.016	0.044	0.179	0.653	0.997	1.000	1.000	1.000	1.000	1.000
wt-rand	0.010	0.019	0.069	0.181	0.663	0.994	0.999	1.000	1.000	1.000	1.000
wt-atm	0.006	0.020	0.047	0.161	0.657	0.996	1.000	1.000	1.000	1.000	1.000
ALL											
MC	0.291	0.291	0.293	0.324	0.528	0.721	0.868	0.947	0.974	0.997	1.000
unwt	0.250	0.254	0.268	0.323	0.527	0.724	0.785	0.834	0.896	0.967	0.996
wt-rand	0.222	0.227	0.248	0.297	0.499	0.669	0.739	0.812	0.906	0.983	0.996
wt-atm	0.217	0.225	0.243	0.293	0.499	0.688	0.741	0.799	0.886	0.983	0.998

Table 5.8: Cumulative frequencies of the absolute relative errors for the non-convex weighted ensembles.

then the conditional error distributions will be consistent over the approximation space which means that the errors can be reliably canceled throughout, for the situation of two models one model is being used to fit as a corrector term. In essence this is what the MSN architecture aimed to achieve by providing a second corrector function with respect to bias in the first neural network, although this proved to be not as successful as the simple MLP architecture.

The use of negative weights can particularly be justified in the case of a set of bias estimators, $\mathbb{E}[\hat{y}] = y + b < y$, although this should not be a problem if $\hat{y}_i = y + b_i \; ; \; b_i \in \mathbb{R}$, as higher weightings can be given to the members with $b_i > 0$. The compelling case occurs when all members of the ensemble are consistently under/overestimating, in this case $b_i < 0 \; ; \forall i$, thus the optimal solution is $\beta_i = 1 \; ; \; b_i = \max(\mathbf{b})$.

The justification for negative weights can be seen for a very particular set of conditions: a) the set of estimators have a consistent direction of bias i.e. the models are always over or underestimating; b) there exists a high degree of positive correlation and collinearity between at least two members of the ensemble to allow for reliable cancelation of errors using negative weights.

Principle Components

Using principle component regression (PCR) it is possible to see the importance of collinearity and negative weights in the options pricing ensemble. PCR is used to reduce the collinearity by forming a regression using the principle components with the highest variance, however it was found that for the PCR ensemble to outperform the MC price approximations the maximum number of components are required. PCR is related to shortest path least-squares regression by the singular-value-decompostion. The pseudoinverse, $\mathbf{B}^+$, of a matrix $\mathbf{X}$ is

$$\mathbf{B}^+ = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T = \mathbf{V}\Sigma^{-1}\mathbf{U}^T. \tag{5.43}$$

With respect to the set of linear weights, β, using shortest-path least-squares and PCR

$$\beta = \left(\mathbf{X}_d^T \mathbf{X}_d\right)^{-1} \mathbf{X}_d^T \mathbf{y} \tag{5.44}$$

$$\beta = \mathbf{V}_d^T \Sigma_d^{-1} \mathbf{U}_d \mathbf{y} = \sum_{i=1}^{d} \frac{u_i^T y}{\lambda_i} v_i \tag{5.45}$$

where d is the number of principle components used, and $\mathbf{X}_d$ is the appropriately truncated matrix. Therefore when using all the principle components PCR is the same as the shortest-path least-squares.

Upon examination of the first principle component, PC1, for the three sets of neural network models it is interesting to see that this component corresponds to taking the mean

$$v_{1j} \propto \frac{1}{Z} \quad \forall j = \{1 \dots Z\}. \tag{5.46}$$

All other components correspond to extremely small variance i.e. there is a high degree of collinearity. However, some of these low variance components may include important information [232] about the structure, as is the case here; when including more of the low variance components the price approximation improves. The addition of the collinearity in these components works as a corrector for the bias from the MC training data and improves the ensemble performance; this shows that the non-convex linear regression is able to extract additional information about the function being approximated and that when using the mean of creating ensembles this information is not included.

Although it can be argued that this may possibly be avoided by creating more diversity within the ensemble, the collinearity and strong presence of a bias can also be present as an artefact of the training data; this is shown to be the case when looking at the bias of the estimation in more depth.

Bias Decomposition

It is interesting to see that the a linear combination can be achieved when the weights are non-convex. An important point to note is that for the weight vectors obtained here it was seen that $\sum \beta_i = 1$; using this fact it is possible to extract more information about the neural network approximation.

For a given target value y there is a regression approximation $\hat{y}$, assuming for each ith member of the ensemble the given approximation of y can be decomposed into a bias approximation of the target value, $\tilde{y}_i$, plus an error term, $\hat{y}_i = \tilde{y}_i + \varepsilon_i$, the linear combination of the ensemble can be decomposed as

$$\hat{y} = \sum_i \beta_i \hat{y}_i = \sum_i \beta_i \tilde{y}_i + \sum_i \beta_i \varepsilon_i. \tag{5.47}$$

Given $\sum_i \beta_i = 1$ this implies that for $\hat{y}$ to produce an accurate estimation of y then the estimators are unbiased and $\tilde{y}_i = y$, $\forall i$. The ensemble approximation reduces down to

$$\hat{y} = y + \sum_i \beta_i \varepsilon_i, \tag{5.48}$$

implying that for an exact approximation there is a linear dependance between the errors, ε_i, of each regressor to minimise $\sum \beta_i \varepsilon_i$. Assuming that ε_i is the sole source of error then for an accurate unbiased ensemble it must hold that

$$\hat{y} - y - \sum \beta_i \varepsilon_i = 0. \tag{5.49}$$

When applying this to the non-convex linear model obtained here it was found in contrast that $\|\hat{y} - y - \sum \beta_i \varepsilon_i\| = 0.13$, which indicates that there is a secondary source of error. This second source of error is assumed to be present as the bias of the regressors for the target value i.e. $\tilde{y}_i = y + b$, adding a bias term, b, to the decomposition now gives

$$\hat{y} = y + b + \sum \beta_i \varepsilon_i \tag{5.50}$$

and rearranging, an expression for the bias, b, can now be found. This decomposition has been applied to the non-convex linear model. The bias for each point with respect to the moneyness is given in Figure 5.13; from this it can be seen that neural networks have a negative bias which constantly underestimate the prices, moreover the bias has a functional form similar to the option price function being approximated and further suggests that the bias is a structural part of the model and not just due to noise. A possible source of the bias could be from the Monte Carlo training data, sources of bias are from discretisation error and non-linear function of means [128]. If a similar pattern of negative bias is present in the MC training data used for both the neural network training and the linear weight training then this can explain some of the origins of the observed bias in the neural networks model.

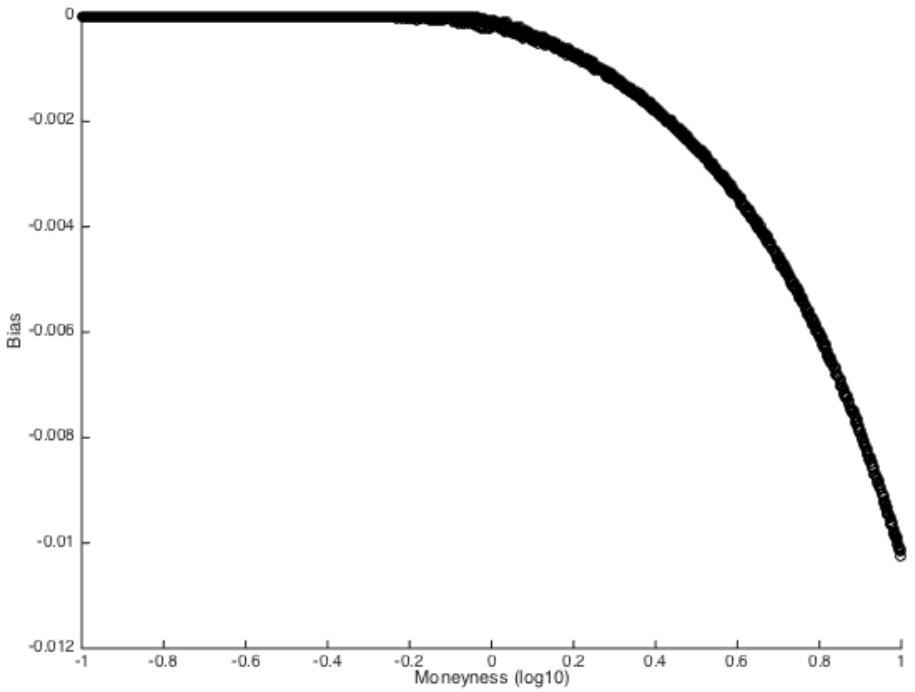


Figure 5.13: Bias of the neural network price approximations obtained using Equation 5.50 for the non-convex ensemble of the unweighted-training set of neural networks.

5.3.5.4 Constrained Regression Using Data Transforms

One major concern remains, that the unconstrained non-convex linear models with negative weightings still perform the best with regards to the acceptable error rates,

although they perform extremely well the use of negative weights causes some concern over their reliability. Similar to the motivation for using a transform during the neural network training another transform is here applied to the ensemble regression data. The aim of this transform is to make certain regions more important. All the ensembles models looked at so far tend do well for DOTM and OTM options with respect to the acceptable error rate, the constrained convex linear weighting ensembles begin to struggle more when accurately pricing ATM, ITM and DITM options. The focus here is on boosting the ATM prices as in practice this tends to be the most important region.

To achieve the desired boosting the log-like Inverse-Hyperbolic-Sine (IHS) transform is used, [1]

$$\mathrm{IHS}\left(x,\theta\right)=\frac{\sinh^{-1}\left(\theta x\right)}{\theta}\;;\;\theta>0 \tag{5.51}$$

where θ is the control parameter. This transformation is often applied within econometrics in the case of proportional variance in the data [233]. The features that make IHS suitable to this application are: the degree of the range of magnitude boosting can be controlled by the θ parameter; the mapping is bijective for $x\geq 0$ and $\mathrm{IHS}(0)=0$.

The ensemble regression to obtain the weight set, β, is now done using the elementwise transformations of the neural network outputs $\mathbf{N}$ and target values $\mathbf{y}$ for the ensemble training data

$$\beta=\mathrm{IHS}\left(\mathbf{N},\theta\right)^{+}\mathrm{IHS}\left(\mathbf{y},\theta\right)\;;\;\theta>0. \tag{5.52}$$

IHS Ensemble Results

The IHS transform constrained regression has been applied, using $\theta=1$, to the three sets of neural networks (unweighted, wt-rand, and wt-atm), the acceptable error rates (AER) are shown in Table 5.9. In comparison to the convex ensembles obtain

[1] The SP10-transform was additionally tested but resulted in worse pricing error and are not reported here.

without the IHS transform it can be seen that the IHS transform is able to improve the AER for all of three sets of neural networks, the most significant improvements are for ATM and ITM options. Based on the results previously seen for the mean and convex ensembles it is unsurprising that wt-atm performs the best when using the IHS transform, the wt-atm set will be used throughout the rest of this discussion.

The effect of the IHS parameter, θ, is shown in Table 5.10. The effect of θ in the IHS transform is that the larger θ becomes the larger the values with smaller magnitudes become, θ can be viewed as controlling how small of magnitude a log-like transform is applied to the value. It can be seen in Table 5.10 that as θ increases the AER of ITM and DITM decreases due to the significance of the magnitude of these prices being reduced, however the decrease in AER for ITM and DITM is disproportionate for the increase seen for the AER for ATM options. in regards to this tradeoff between ATM, ITM and DITM accuracy the value of $\theta = 1$ is chosen to give the best balance. Compared to the AER of the non-convex ensemble for wt-atm there are improvements for OTM and ATM AER, from 0.92 up to 1.00 and 0.89 up to 0.94 respectively, in fact the OTM AER of 1.00 obtained here is better than the best OTM AER out of all of the previously discussed ensembles. However, the tradeoff is that there is about a 4% and 10% drop for ITM and DITM AERs.

Table 5.11 gives the cumulative distributions of the absolute relative error (ARE) for the IHS transform regression $\theta = 1$ for the wt-atm set of neural networks. The distribution shows that for OTM options the cumulative frequency of instances for ARE $< 10\%$ is 57% and this decreases to only 12% for $< 1\%$, whilst for MC it is 98% and 64% respectively. However, it was seen previously that all the non-convex weighted ensembles also struggled for OTM options, and the results here for the convex IHS transform are similar to the non-convex weighted ensembles. This highlights that there is still a lot of room for improvement with respect to accurately pricing low valued options, this is not reflected in the high AER given the relative high level of tolerance in the absolute error boundary with respect to these option prices. In a generalised setting with realistic error tolerances these prices suffice; although improvements may be able to be achieved by using a larger θ is

increase the relevance of these low prices, but as seen this comes at the larger cost of disproportionately worse ITM and DITM performance.

The two regions of improvement in comparison to the convex ensembles without the IHS transform are the ITM and ATM options. For ITM for ARE¡0.3% the performance is slightly better than MC by about 5%, and becomes a lot better than MC when considering other higher degrees of accuracy. Though, when compared to the non-convex ensembles the performance is not as good as either wt-atm or the unweighted set with the IHS transform convex ensemble being around 10% worse for ARE¡0.3%, but for higher degrees of accuracy the performance between the ensembles becomes the same. This shows that for a small percentage of ITM options that although the AER is only around 5% worse an additional 5% are also not priced as accurately as the non-convex ensembles.

Compared to MC IHS transform convex ensemble show a significantly better degree of accuracy with 94% of instances having ARE$<0.3\%$ and still over half the prices acheive ARE$<0.1\%$, and around a quarter acheive ARE$<0.03\%$. Compared to the non-convex ensembles the IHS transform convex ensemble is better for the wt-atm set, although it is slightly worse than the non-convex ensemble for the unweighted set, but using a slightly larger θ may be able to acheive the same level accuracy. These results show that the IHS transform convex ensemble exhibits extremely good accuracy for what may be considered the most important region. However, it can be seen from Figure 5.14.a that a set of large outlier errors still exist within this region and showing that there are still cases, in particular low volatility (see Section 5.3.6), that need improvement.

Applying the IHS transform to the constrained regression significantly increases the accuracy of OTM, ATM and ITM option price approximations. This increase in accuracy also comes with the welcomed advantage of providing a more reliable and robust model in the sense that the weights are non-negative and the range of the approximation is now bounded, Equation 5.41, by the individual neural network outputs. Using convex weights also allows for significant pruning and here the ensemble only required 20% of the available neural networks from the set,

	DOTM	OTM	ATM	ITM	DITM
unwted	1.00	0.85	0.82	0.60	0.48
wt-rand	0.99	0.78	0.68	0.79	0.47
wt-atm	1.00	1.00	0.94	0.91	0.44

Table 5.9: Acceptable error rates for the inverse-hyperbolic-sine transform convex ensembles

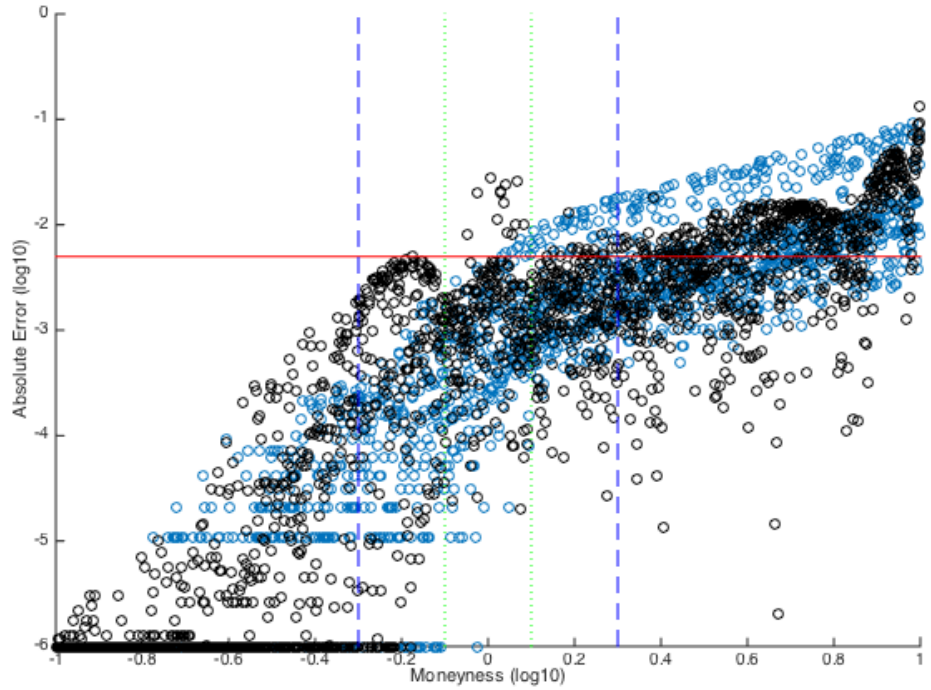
	DOTM	OTM	ATM	ITM	DITM
wt-atm					
No Transform	1.00	0.99	0.78	0.70	0.41
$\theta = 1$	1.00	1.00	0.94	0.92	0.44
$\theta = 2$	1.00	1.00	0.94	0.93	0.43
$\theta = 5$	1.00	1.00	0.94	0.95	0.37
$\theta = 10$	1.00	1.00	0.95	0.85	0.27

Table 5.10: Acceptable error rates for the IHS transform, Equation 5.51, linear regression ensembles using the of the wt-atm set of neural networks. θ is the IHS control parameter.

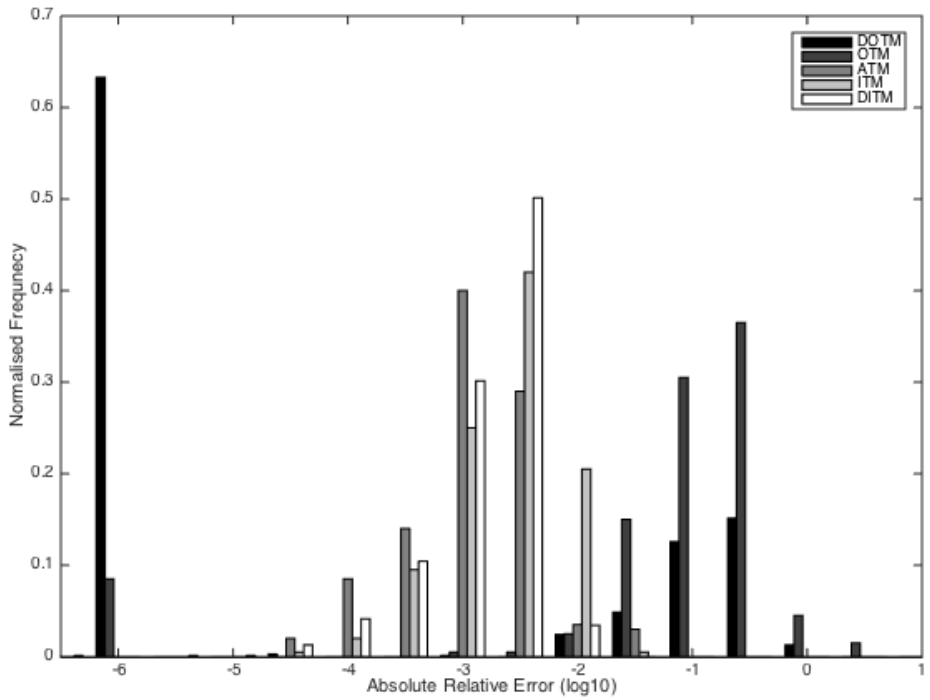
$\theta = 1$	Absolute Relative Error								
	$\leq 0.01\%$	$\leq 0.03\%$	$\leq 0.1\%$	$\leq 0.3\%$	$\leq 1\%$	$\leq 3\%$	$\leq 10\%$	$\leq 30\%$	$\leq 100\%$
DOTM	0.636	0.636	0.637	0.637	0.661	0.710	0.836	0.987	1.000
OTM	0.085	0.085	0.090	0.095	0.120	0.270	0.575	0.940	0.985
ATM	0.105	0.245	0.645	0.935	0.970	1.000	1.000	1.000	1.000
ITM	0.025	0.120	0.370	0.790	0.995	1.000	1.000	1.000	1.000
DITM	0.059	0.163	0.464	0.966	1.000	1.000	1.000	1.000	1.000
ALL	0.256	0.305	0.445	0.683	0.753	0.813	0.894	0.989	0.999

Table 5.11: Cumulative frequencies of the absolute relative errors for the IHS transform regression convex weighted ensemble of the wt-atm set of neural networks.

whilst the non-convex models rely on the full set. Though, the considerable gains in accuracy were only seen for the *wt-atm* neural networks, the lower mutual information and higher diversity of this set of neural networks makes it more suitable for the constrained regression compared to the other sets of neural networks. Overall the IHS constrained regression applied to the wt-atm neural networks presents a more accurate, robust and smaller generalised function approximator than previous methods discussed.



(a) Absolute errors.



(b) Absolute relative error distribution

Figure 5.14: Absolute pricing error and distribution distribution of absolute relative pricing errors (ARE) for the convex neural network ensemble trained using the inverse hyperbolic sine transform regression.

5.3.6 Volatility Effect and the Payoff Function

One aspect that has been so far neglected in the preceding discussions is the effects of the volatility and interest rate inputs, the focus has primarily been on the moneyness as this is the principle factor. Even though the accuracy has been mainly discussed with respect to the moneyness there still exists respectively large outliers within the ATM region. From Figures 5.11 and 5.12 it has been seen that a cluster of large errors occur for high volatility ATM options.

As the time-to-maturity, in this case measured by the volatility and interest rates, become smaller, in particular the volatility, there is less diffusion in the asset price and in the limiting case of $\sigma = 0$ the price C takes on the form of the final payoff function, $I(S) = (S-K)^+$,

$$C = e^{-r}(S-K)^+ \tag{5.53}$$

Therefore the difficulty arises when approximating the discontinuity, which occurs for low volatility at ATM options $S = K$ or with respect to moneyness $m = 1$.

Considering the neural network model used here where $\hat{C} = T^{-1}[N(\Omega)]$, where T^{-1} denotes the inverse transform of the neural network output. Using the moneyness, m, the payoff function is approximated as

$$I(S) = T^{-1}[T[(m-1)^+]]. \tag{5.54}$$

Firstly note that

$$x^+ = xH(x) \tag{5.55}$$

where $H()$ is the Heaviside step function. Writing $g(m) = (m-1)$ the payoff function can be written as

$$I(S) = T^{-1}[T[g(m)H(g(m))]]. \tag{5.56}$$

Assuming the neural network is aiming to approximate this functional form of the payoff function, it can be seen that the neural network needs to be able to accurately approximate the Heaviside function. It is the error of approximating the Heaviside function that can cause the observed errors for the ATM and OTM options.

5.3.6.1 Approximation Error

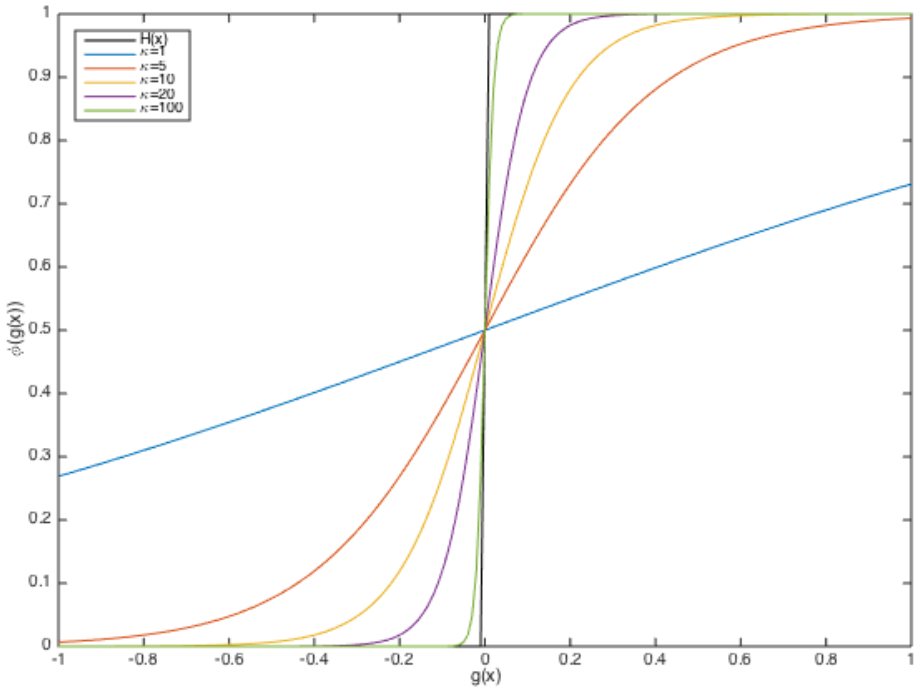
Based on a neural network with a sigmoid transfer function, the Heaviside function will have to be approximated using a sigmoid function, $\phi(x,\kappa) = \frac{1}{1+e^{-\kappa x}}$, where κ is the rate parameter and determines the gradient of the sigmoid at $x = 0$. Figure fig:happrox.a shows the value of the sigmoid function for different rate parameters compared to the Heaviside function. The error of the sigmoid approximation to the Heaviside function can result in significant errors in the price output. Consider an OTM option, where $H(g(m)) = 0$ and $g(m) < 0$; in the sigmoid approximation however, $\phi(g(m),\kappa) > 0$, and it can then be seen that as a result it is possible to obtain negative values, $g(m)\phi(g(m),\kappa) < 0$, this is illustrated in Figure fig:happrox.b. Furthermore, when the sigmoid approximation is inverted via the output transform, T^{-1}, to obtain the price approximation this can result in a positive price and a high relative error. This also shows that without a suitable output transform it would be possible to obtain negative price approximation.

Definition 5.2. Define the error of the sigmoid approximation of the Heaviside function $H(x)$ as $\varepsilon_H(x)$ where

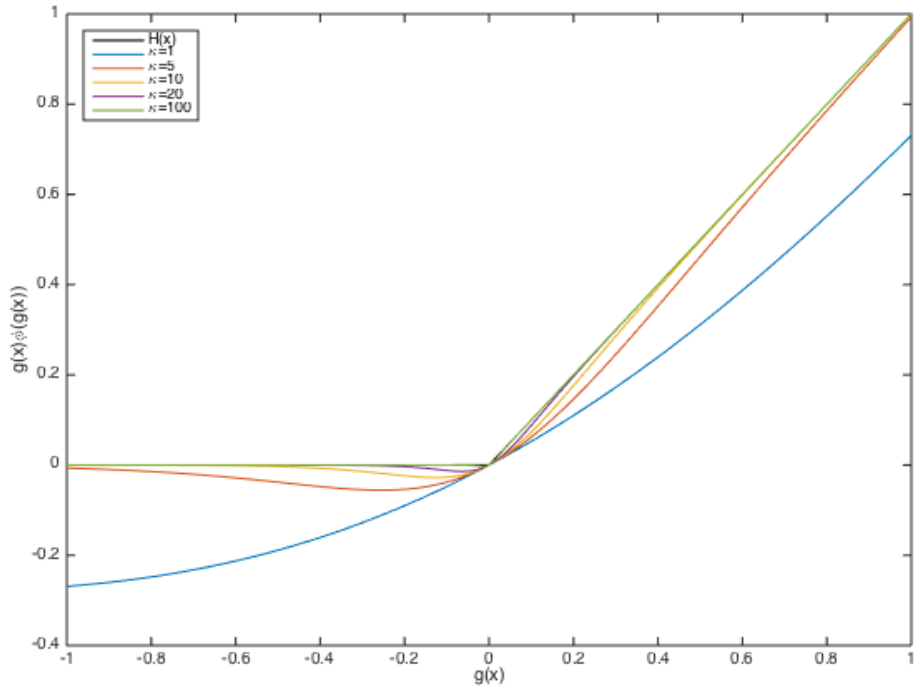
$$\varepsilon_H(x) = \phi(x,\kappa) - H(x). \tag{5.57}$$

The error for approximating the payoff function $g(m)H(g(m))$ can be seen to be $g(m)\varepsilon_H(g(m))$; it can be observed from this that for exact ATM options where $g(m) = 0$, and assuming that $g(m)$ is accurately approximated, then the approximation error of the sigmoid has no impact. The approximation errors occur for small increments around $g(m) = 0$, which explains the small cluster of large errors observed for high volatility options around $\log(m) = 0$.

A further insight into the error behaviour can be derived. For simplicity it shall



(a) Sigmoid functions, $\phi(x,\kappa)$, with different rate parameters κ compared to the Heaviside function $H(x)$.



(b) Calculating the function $g(x)\phi(x)$ using sigmoid functions, $\phi(x,\kappa)$, with different rate parameters κ, compared to using the Heaviside function $H(x)$.

Figure 5.15: Approximating the Heaviside function and related values using sigmoid function approximations.

be assumed that the output transform function T is the function $\log(x+a)$ where a is small positive constant, this is a reasonable assumption as $\lim_{x\to 0} T_{\text{sp10}} \to \log(x)$. Using the Heaviside form of the payoff function the approximation error, $E_{N(m)}$, can be given as

$$E_{N(m)} = \log\left(g\left(m\right)\phi\left(g\left(m\right)+a\right)\right) - \log\left(g\left(m\right)H\left(g\left(m\right)\right)+a\right) \tag{5.58}$$

$$= \log\left(\frac{g\left(m\right)\left(H\left(g\left(m\right)\right)+\varepsilon_H\right)+a}{g\left(m\right)H\left(g\left(m\right)\right)+a}\right) \tag{5.59}$$

for OTM options where $H(g(m)) = 0$ this becomes

$$E^{\text{OTM}}_{N(m)} = \log\left(\frac{g\left(m\right)\varepsilon_H + a}{a}\right) \tag{5.60}$$

and for ITM options where $H(g(m)) = 1$

$$E^{\text{ITM}}_{N(m)} = \log\left(\frac{g\left(m\right)\left(1+\varepsilon_H\right)+a}{g\left(m\right)+a}\right). \tag{5.61}$$

This shows that with respect to the threshold value a used, errors for OTM prices can quickly explode if the sigmoid approximation to the Heaviside function is not sufficient, whereas for ITM options $g(m)$ becomes the dominant term and the errors are more stable. This can also explain why the OTM options were seen to be harder to accurately price than other regions.

This shows that there is an inherent issue when approximating the initial payoff function using sigmoid transfer functions, especially for OTM options as they can be given a small price value. One possible solution could be to explicitly add Heaviside transfer functions or using the piecewise linear-rectifier function to add discrete behaviour and to remove the approximation error, when using gradient-based training methods this introduces additional challenges but does not pose any issues when using heuristic optimisation based training methods.

Approximation Error Bounds

A loose upper bound for $\varepsilon_H(x)$ can be given as

$$|\varepsilon_H(x)| < 0.5, \quad \forall x \in \mathbb{R} \tag{5.62}$$

given that H(0)=0, whilst $\phi(0,\kappa) = 0.5$. But this will largely overestimate the error as the value of $|g(m)| > 0$, it is more appropriate to use an upper bound for the known smallest increment of $|g(m)|$. It is also possible to use the bounds of the Hausdorff distance between the sigmoid approximation and the Heaviside function given by by Kyurkchiev [234]

$$d_l = \frac{1}{\frac{\kappa}{2}+2} < d(\kappa) < \frac{\ln(\kappa+1)}{\kappa+1} + \frac{\ln\ln(\kappa+1)}{(\kappa+1)\frac{\ln\ln(\kappa+1)}{1-\ln(\kappa+1)} - 1} = d_r(\kappa). \tag{5.63}$$

Using the results of Kyurchiev the Heaviside approximation error can be bounded by

$$\frac{d_l}{2} < |\varepsilon_H(g(m))| < \frac{d_r}{2} \tag{5.64}$$

Although it is possible to here to see theoretically here how price approximation errors can occur due to issue of approximating the discontinuous payoff function it is not possible to accurately calculate the actual observed effect of this error due to the fact that it is unclear which neuron or neurons are being used to approximate this part of the function. For a sigmoid neural network the quantity $-\kappa x$ is determined by the weighted inputs into the neuron. For the sigmoid to accurately approximate the Heaviside component large weights are required to give a large κ. Instead of relying upon the neural network to approximate these type of discontinuous functions it is possible to instead embed them directly into the solution.

Trial Solutions

It is worth briefly mentioning here, although out of the scope of implementation of this current work, the possibility of reducing errors of this type by using trial solutions. In the trial solution approach the neural network $N(\omega)$ is used as part of a solution, $U(\omega)$, that automatically satisfies some of the boundary conditions. For example a simple trial solution of

$$U(\omega) = e^{-r}\left((S-K)^{+} + \sigma N(\omega)^{2}\right), \tag{5.65}$$

would be able to automatically satisfy the payoff function without the neural network having to approximate any discontinuous functions.

5.4 Path Dependent Options - Examples

To further demonstrate the applicability of the discussed methodology it is now applied to exotic path dependent option contacts. Given the notably high accuracy of the non-convex linear weighting this method is therefore demonstrated here.

5.4.1 Geometric Asian options

The unweighted-training methodology along with unconstrained shortest-path linear regression to produce of the final ensemble price approximation used for European style options will now been applied to geometric Asian options. Asian options present the next level in complexity where the payoff is now path dependent. The payoff is a function of the geometric mean of the asset price path over the time-to-maturity (for more details see Section 2.3, Equation 2.30). Additionally, for geometric Asian options there is a known analytical solution which allows for direct testing of the neural network's performance.

Results and Discussion

The acceptable error rate for geometric Asian options, Table 5.12, is less informative than for European options, as it can be seen that nearly all regions, except for neural network ATM, have an acceptable error rate of 100% for both the neural network model and MC. This is partially due to the overall lower prices of Asian

	DOTM	OTM	ATM	ITM	DITM
MC	1.00	1.00	1.00	1.00	1.00
NN	1.00	1.00	0.99	1.00	1.00

Table 5.12: Acceptable error rates for MC and neural network ensemble price approximations.

options, but also due to the smaller effects of volatility on the price which make the options easier to approximate; using a function of the mean asset price in the payoff reduces the effect of the volatility and in turn results in lower prices in comparison to European options. It can be seen from Figure 5.16 that the ATM AER of 99% for the neural network ensemble is only due to two high volatility instances. This is similar to what has been observed for European options where for ATM options high volatility options prove to be more difficult to price due to the increased functional convexity. It can also been seen similar to European options that lower value options (DOTM and OTM) with low volatility have larger errors than those with high volatility. Due to the geometric averaging these volatility effects occur to a lesser extent than seen for European options.

For DOTM both the neural network and MC were able accurately price all the low valued options, the majority of these prices tend towards zero and hence for the neural network the function approximation becomes more of a classification problem. Both MC and the neural network ensemble have more difficultly with OTM options, although up until ARE$<30\%$ MC has twice as many instances than the neural network. MC has 50% of instances for ARE $< 0.3\%$ whilst the neural network only achieves this same percentage of instances for ARE $< 10\%$. Compared to European options both MC and the neural network perform a lot better for higher degrees of OTM accuracy ARE$< 1\%$, this likely due to a higher proportion of OTM options being valued as zero for the Asian options.

For ATM options the neural network ensemble is significantly more accurate than MC, the neural network approximations has ARE $< 0.1\% = 94\%$ whilst MC is only 50% for the same ARE, and then only achieves a percentage of instances $< 90\%$ for ARE$< 1\%$. Down to ARE $< 0.01\%$ both methods have a similar error

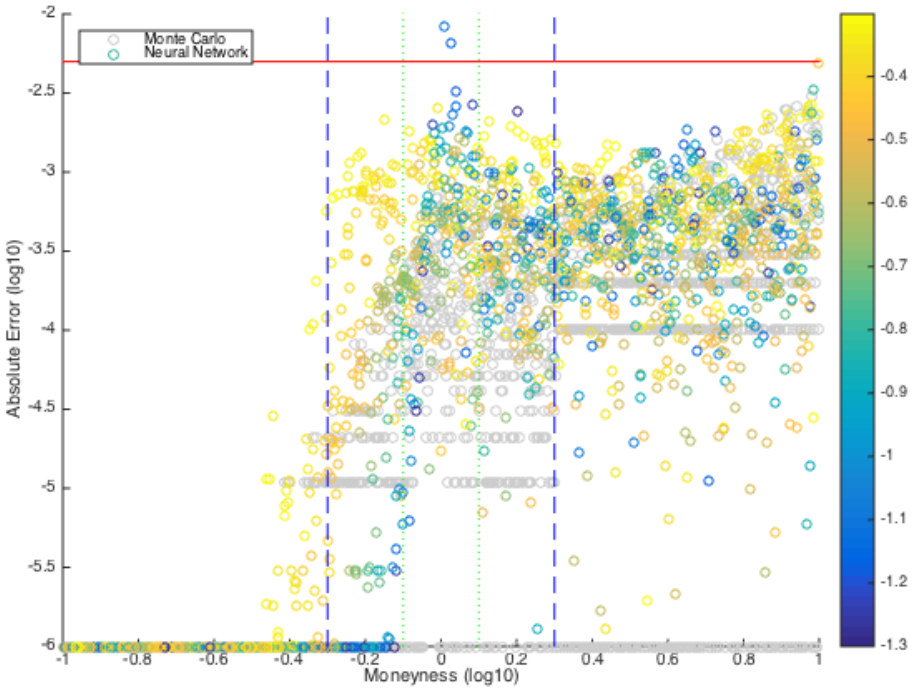


Figure 5.16: Scatter plots showing the absolute relative error for the 10 median-aggregated neural network European pricing models for each of the neural network architectures explored, 2-Layer (20N,Res=10^{-6}).

	Absolute Relative Error										
	$\leq 0.001\%$	$\leq 0.003\%$	$\leq 0.01\%$	$\leq 0.03\%$	$\leq 0.1\%$	$\leq 0.3\%$	$\leq 1\%$	$\leq 3\%$	$\leq 10\%$	$\leq 30\%$	$\leq 100\%$
DOTM											
MC	0.976	0.976	0.976	0.976	0.976	0.976	0.976	0.979	0.983	1.000	1.000
NN	0.941	0.946	0.946	0.946	0.946	0.946	0.951	0.959	0.967	0.990	1.000
OTM											
MC	0.475	0.475	0.475	0.475	0.480	0.515	0.705	0.875	0.950	0.995	1.000
NN	0.255	0.255	0.255	0.255	0.265	0.270	0.295	0.360	0.515	0.890	1.000
ATM											
MC	0.140	0.165	0.220	0.300	0.500	0.780	0.980	0.990	1.000	1.000	1.000
NN	0.025	0.105	0.240	0.605	0.940	0.990	1.000	1.000	1.000	1.000	1.000
ITM											
MC	0.155	0.365	0.600	0.870	1.000	1.000	1.000	1.000	1.000	1.000	1.000
NN	0.015	0.050	0.155	0.450	0.835	0.995	1.000	1.000	1.000	1.000	1.000
DITM											
MC	0.240	0.464	0.809	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000
NN	0.083	0.243	0.597	0.950	1.000	1.000	1.000	1.000	1.000	1.000	1.000
ALL											
MC	0.503	0.605	0.754	0.856	0.890	0.921	0.960	0.979	0.989	1.000	1.000
NN	0.382	0.445	0.582	0.742	0.807	0.847	0.878	0.907	0.933	0.983	1.000

Table 5.13: Cumulative frequencies of the absolute relative errors for the non-convex weighted neural network (NN) ensembles and Monte Carlo (MC) price approximations for geometric Asian options.

rate of around 20%, although below this MC becomes better for a small number of cases.

For ITM options both methods have a high percentage of instances for a high degree of accuracy of ARE $< 0.3\%$ and still for ARE $< 0.1\%$, although compared to ATM options at this level of accuracy the MC has improved whilst the neural network ensemble is slightly worse. Furthermore, for higher degree of accuracy, ARE $< 0.03\%$, the number of neural network instances drops by almost 40% down to 45%, whilst MC only drops by around 13% down to 87%, at an even higher degree of accuracy for ARE $< 0.01\%$ the neural networks drops further to only 15% whilst MC remains quite high at 60%. It is similar for DITM where both show similarly high frequencies for ARE $< 0.03\%$, but below this the NN accuracy degrades faster than the MC. Looking at the overall AREs (ALL), MC shows a higher degree of overall accuracy, however for an ARE$> 0.1\%$ for all price regions, apart from OTM, the neural network and MC are comparable. It was seen that for ATM options the neural network ensemble is significantly better, for overall generalised performance MC may be the better the choice, but given there is a high weighting on the practical significance of ATM prices the NN options may instead be preferable.

These results show that the current methodology is able to produce viable prices for geometric Asian options, and for the case of ATM options the neural network ensemble used here is better than MC with respect to the absolute relative errors, although for other regions there is room for further refinement, in particular for OTM and ITM options. Knowing that this method can produce reliable geometric option prices this method should also be able to provide an accurate pricing model for Arithmetic Asian options which do not yet have a known analytical solution, and this methodology could provide a useful desktop tool for traders. For Asian options further investigations should be made using the wt-atm training methodology, it was seen for European options this method showed significant increases in accuracy for the approximations OTM and ITM options by being a less biased estimator of the pricing function, which is needed here for the neural network to be truly competitive

against the MC methods in the sense of overall precision.

5.4.2 American options

American options present another level of difficulty because of the early exercise feature of the payoff function; the early exercise property means that there currently no known analytical solution for this type of option contract, therefore relying heavily upon numerical pricing methods. It should be noted that for American options it is assumed the time-to-maturity is constant, $\tau = 1$, as Theorem 5.2 has not been proven for the case of American options, the difficulty arises because the discounting term is dependent on the time of early execution. The expectation term can be scaled, but without knowing the time of early execution it is not possible to determine the discount factor; further work can explore how to extract this information from the neural network as a second output, or find a functional form to predict the time of early exercise τ^*. The second modification to the main methodology is that because unnormalised prices are being obtained with respect to the strike price the training data is multiplied by a factor of 10, so that the unnormalised price approximation will retain the same level of accuracy as the unnormalised prices.

Due to the fact there is no known analytical solution to compare prices to results for the American put options priced here are compared to a small set of 6 highly accurate numerical prices from the literature [235] given in Table 5.14. Options 1-4 are ITM options whilst options 5 and 6 are OTM, for the same strike prices the volatility σ is also varied for either low, 0.2, or high, 0.4, volatility regimes. Table 5.14 compares two different numerical methods, fine-grain finite difference and the high performance GPU based Longstaf-Schwarz Monte Carlo model implemented by NVIDIA, the precision of the methods being measured up to 3 decimal places.

	σ	r	S	K	FD Price	GPU LSMC
1	0.2	0.06	36	40	4.473	4.478
2	0.4	0.06	36	40	7.098	7.101
3	0.2	0.06	38	40	3.248	3.250
4	0.4	0.06	38	40	6.138	6.148
5	0.2	0.06	44	40	1.112	1.110
6	0.4	0.06	44	40	3.944	3.948

Table 5.14: American Put options tested with $\tau = 1$, fine grid finite difference (FD) and GPU Longstaff-Schwarz MC (LSMC) price from [235]

To create the neural network ensemble model 300 independent neural networks are trained on low resolution MC simulations (5000 replications, 365 periods) using the Longstaff-Schwarz method [115] for the early exercise price; less replications compared to European options are used for the American training data due to the additional computational time required by the Longstaff-Schwarz method, as well as the previously mentioned benefits of having slightly noisy training data.

Results and Discussion

The average absolute error and absolute relative errors of all 300 trained networks for the 6 test options are given in Table 5.4.2, it can be seen that on average the neural networks largely deviate from the reference price by between 1-10%, and in particular for option 5 the neural network prices deviates around 75%. With respect to the mean absolute error the average pricing errors exceed the acceptable pricing error boundary (AEB) of 0.005, and from looking at the mean model in Table 5.16 it can be seen that for ITM options the average price approximation is an overestimate whilst for OTM options the neural networks underestimate. The larger errors occur for the low volatility options, and in particular the low volatility OTM option, this is the same previously seen for European and Asian options and is related to how well the neural network call model the discontinuous payoff function. Though, it can be seen from the minimum absolute error that there is potential for the neural network models to accurately estimate the price within the AEB, and hopefully the linear regression ensemble method can extract this information.

	Ref Price	MAE	Std	Min	MRE	Std
1	4.473	0.464	0.155	0.001	0.104	0.035
2	7.098	0.201	0.116	0.001	0.026	0.019
3	3.248	0.229	0.138	0.000	0.058	0.058
4	6.138	0.151	0.109	0.000	0.016	0.026
5	1.112	0.846	0.233	0.013	0.762	0.210
6	3.944	0.446	0.390	0.003	0.064	0.136

Table 5.15: Average pricing errors of the 6 test cases for the 300 individual neural network models.

Table 5.16 gives the prices generated by mean and non-convex weighted linear ensembles for the 6 test options. The overall error-norm of the ensemble prices for the the 6 test cases are measured as the Euclidean distance between the price approximations and the reference prices. The error-norm shows that the non-convex linear ensemble significantly improves the overall price approximations compared to the mean ensemble. The non-convex ensemble tends to overestimate the majority of the prices as is also seen for the mean ensemble, this shows that there is a strong positive bias present in the neural network models. The largest improvements compared to the mean ensemble are for option 1, ITM with low volatility, and option 5, OTM with low volatility; for all the low volatility options the linear ensemble has improved the prices indicating that the regression was able to improve the fit to the final payoff function $(S-K)^+$). The linear ensemble is still most accurate for the for deeper ITM options which are typically the easier region to fit due it being the most linear function, the option 6 would have the highest degree of functional convexity which is reflected in it having the largest absolute error.

With respect to the relative errors they are all low between 1-4%, although when compared to European options for ITM options the majority of prices approximations had relative errors of $< 0.3\%$, which this level is only seen for option 2 for American options. For OTM options European price approximations were only significantly accurate for relative errors $< 10\%$ whilst a level of relative errors of around 3% were achieved for the two OTM American samples. Although it is not possible to draw any firm conclusions due to the small sample size of American options it may be speculated that the higher value of American options may make it

	Ref Price	Mean NN	Lin NN	AE (Lin)	RE (Lin)
1	4.473	4.942	4.543	0.070	0.016
2	7.098	7.287	7.120	0.022	0.003
3	3.248	3.441	3.288	0.040	0.012
4	6.138	6.244	6.051	-0.087	-0.014
5	1.112	0.264	1.155	0.043	0.039
6	3.944	3.694	4.068	0.124	0.031
Error Norm		1.042	0.178		

Table 5.16: Price estimations comparisons for the 6 test cases for the neural network (NN) ensembles (mean and non-convex linear weights). The error norm is given as the RMSE with respect to the reference price over all 6 test cases.

easier for neural networks to price American OTM options compared to Europeans OTM options, the larger values means it is less likely for the neural networks to make large relative errors and misclassification for near zero valued options. Although the relative errors seem to be good, the larger magnitude of the normalised prices presented here means that the absolute errors are still outside of the exchange quote AEB by at least one decimal place, in this respect when compared to European options the American options are not as accurately priced, for which nearly 100% of the European price approximations were within the AEB. Further improvements such as using more training samples or using wt-atm weighted training may be able to sufficiently improve the neural network model for American options.

Overall more work needs to be done to further explore this method for American style options, and the neural network models need to be compared to more samples of high accuracy numerical pricing results to fully understand the limitations and strengths of this methodology. Even though the accuracy of this method still needs to be improved with respect to the numerical methods used for the 6 test cases, the speed and processing power advantages of the neural network method are obvious; the numerical results for the 6 test cases required a GPU whilst evaluating a neural network can be done easily and quickly on any ordinary CPU. The results presented show that there is a lot of potential for this method to produce comparably accurate option prices to numerical methods, but with a fraction of the speed for evaluating the model once trained, and provides a fruitful avenue for research.

5.5 Conclusions

This work presented an empirical study of the use of neural networks to approximate the parameterised option price $V(S,K,\sigma,r,\tau)$. Firstly it was shown that using known relations between the variables the input parameter space of the neural network model can be reduced from the original five to three. One current limitation of this dimensional reduction is for the case of American options, where the output price cannot be fully generalised to any time-to-maturity due to the unknown early execution discount factor $e^{-r\tau^*}$.

The methodology developed involved training a set of neural networks on Monte-Carlo generated price data, and then ensembling the set of neural network models into one single approximate. It was concluded that a simple multi-layer-percetron (MLP) network gave a better function approximation than the more elaborate multi-stage architecture; the MLP was able to learn the overall shape of the function better, shown by the approximation of the Greeks, whilst the more elaborate multi-stage architecture over-trained on just the price output.

In addition to normal unweighed MLP training two other sets of neural networks were trained using two different weighted-training methods (wt-rand and wt-atm) aimed at increasing the diversity and localised accuracy. The normal unweighted trained MLPs showed overall better accuracy for European options, although the wt-atm trained neural networks showed better accuracy for the targeted ATM region. The wt-atm training also had a second advantage that the neural networks interpolated the pricing function better and gave rise to a less biased estimator than the unweighted trained neural networks; this allowed the wt-atm networks to produce considerably more accurate convex weighted ensembles that can be considered a more reliable, and, due to pruning, in addition a smaller final model.

Considering that the domain of the inputs is closed, and the neural networks are not being used to extrapolate outside of this domain, the non-convex linear weights are considered as the most accurate method presented without too much risk to the reliability being introduced by the associated negative weights. For European options it was seen that this method rivalled the Monte-Carlo (10000 replications, 365

time steps) numerical prices. With respect to exchange quoted accuracy of two decimal places both neural network and MC methods were equally accurate for deep-out-the-money options (DOTM), out-the-money options (OTM) and at-the-money options (ATM); the neural network was more accurate for in-the-money (ITM) and deep-in-the-money (DITM) options. Looking at higher degrees of accuracy, with respect to the absolute relative error, it was seen that the neural network was worse for OTM and DOTM options, but showed significantly more accurate prices for ATM and ITM options. While for both methods there was a trade-off between accuracy for either DOTM and OTM or ATM and ITM, in practice it could be argued that ATM are the most important and the neural network method could therefore be the more favourable.

The neural network method has additionally been demonstrated for path dependent options (geometric Asian call options, and American put options), for which similar studies [201] had significant issues with. In the work here it was shown that the neural network methodology is able to produce acceptable prices for these exotic options. For the geometric Asian options the MC and NN prices provided similar degrees of accuracy with respect to exchange quoted prices but and for overall absolute relative errors of $> 0.1\%$, furthermore the neural network method was considerably more accurate for ATM option prices. American options proved to be more difficult and for the cases examined although the neural network method was able to generate reasonable price approximations, unlike [201], the relative error of the price approximations were between 1-4%, however additional work is required to improve the method that bring the errors down by one decimal place to be within the exchange quoted prices, but this work is able to show that it is possible to accurately price American options using neural networks.

One element not yet emphasised it the computational speed-up and efficiency of the neural network method. Generating the MC prices requires a heavy computational load, as illustrated by the amount of work within the literature focusing of parallelisation and high performance computing methods applied to MC. On the other hand, the neural network model, once trained, provides fast and computation-

ally efficient prices as it only requiries forward passes through shallow MLP neural networks, with the same or better levels of accuracy than provided by MC. This provides a compelling case for the neural network methodology to be further refined and investigated, and this work has provided a successful exploration supporting the use of neural networks for accurately pricing options contracts.

Final Remark

This work has shown the practicality of applying novel numerical methodologies for improving the accuracy and efficiency of derivatives pricing. The following chapter looks at how rather than better software, specialised hardware can instead be utilised to improve the computational efficiency of numerical methods for derivatives pricing.

Chapter 6

Options Pricing using Hardware Acceleration

This chapter presents the design and implementation of the Thomas algorithm optimised for hardware acceleration on an FPGA. The hardware based algorithm combined with the custom data flow and low level parallelism available in an FPGA reduces the overall complexity from 8N down to 5N serial arithmetic operations and almost halves the overall latency by parallelising the two costly divisions, and memory costs for reduced down to 2N. Using a data streaming interface, the Thomas Core developed allows for multiple independent tridiagonal systems to be continuously solved in parallel, providing an efficient accelerator for many computations. Finally the Thomas solver core is applied for derivatives pricing problems using implicit finite difference schemes on an FPGA accelerated system and investigates the use and limitations of fixed-point arithmetic.

6.1 Introduction

6.1.1 FPGAs

Field programmable gate arrays (FPGAs) provide an integrated circuit that can be reconfigured on the fly or 'in the field' in the form of a chip. FPGAs provides a flexible and cost effective way to develop and implement custom hardware designs. The core component that allows an FPGA to be reconfigurable is a look-up table (LUT). A LUT produces an output/outputs as a function on the digital inputs, these

functions are determined when the FPGA is configured and provides the desired logic by the inputs controlled via the programmable cells. The other key component that helps to increase the performance of FPGAs are on-chip block memory (BRAM) which can provide fast local memory caches. Some FPGA chips may also offer other additional features such as high speed digital signal processors (DSP) and multipliers. The FPGA chip is then usually placed on a circuit board and connected to additional peripherals such as DDR memory, USB ports, ethernet, PCI express and VGA ports to provide the complete heterogeneous computing system.

6.1.2 Finite Difference Schemes and Tridiagonal Systems

Finite difference (FD) schemes are an important tool for solving parabolic partial differential equations (PDEs) numerically. In financial engineering FD methods [129] [210] are commonly employed to solve PDEs that are used to model derivatives, such as the famous Black-Scholes equation (BSE).

Finite difference schemes begin by discretising the problem domain into a mesh/grid over the time interval $[0,1]$ and in basic cases, the asset price interval $[0,S_{max}]$. The domain is discretised into N asset price steps and M time steps, given by:

$$\Delta S = \frac{S_{max}}{N}, \tag{6.1}$$

$$\Delta t = \frac{1}{M}. \tag{6.2}$$

The spatial derivative terms are approximated using central difference and backward difference for the time derivative. These discretizations are then substituted into the PDE to produce the discrete difference equation, for example the BSE equation gives:

$$V_n^m = a_n V_{n-1}^{m-1} + b_n V_n^{m-1} + c_n V_{n+1}^{m-1}, \tag{6.3}$$

with problem dependant stencil coefficient values a_n, b_n, c_n. This is the basic implicit scheme used for one dimensional problems, the resultant system of equations

can be written in matrix form and needs to be solved the for the price vector at the current time-step,V^{t-1}, where the vector V^t is known from the previous implicit step,

$$\mathbf{A}\mathbf{V}^{t-1} = \mathbf{V}^t. \tag{6.4}$$

This can be more generally written as the matrix inversion problem $\mathbf{A}x = y$, where the coefficient matrix $\mathbf{A}$ takes on the banded tridiagonal form shown below:

$$\mathbf{A} = \begin{bmatrix} b_0 & c_0 & 0 & 0 & 0 & \dots & \\ a_1 & b_1 & c_1 & 0 & 0 & \dots & \\ 0 & a_2 & b_2 & c_2 & 0 & \dots & \\ 0 & 0 & a_3 & b_3 & c_3 & 0 & \dots \\ . & & & & & & \\ . & & & & & & \\ . & & & & & & \\ 0 & & & & \dots & 0 & a_N & b_N \end{bmatrix}. \tag{6.5}$$

Furthermore, when introducing numerical schemes for pricing multidimensional derivatives, such as basket options or under stochastic volatility, another class of finite difference schemes known as alternating-direction-implicit schemes [236] may be used. These schemes solve the PDE in an implicit manner within multiple dimensions. These methods can be computational challenging as they require the solution to multiple tridiagonal systems at each time step, thus a lot of effort has gone into creating fast parallel solvers on devices such as GPUs [237] [238].

6.1.3 Thomas Algorithm

Algorithm 6.1 Thomas Algorithm (a,b,c,y) Pseudo Code

```
d[0] = b[0]
z[0] = y[0]
for  i = 1 to N do
   prev = i - 1
   l_i = a[i]/d[prev]
   d[i] = b[i]-l_i*c[prev]
   z[i] = y[i] - l_i*z[prev]
end for
z[N] = z[N]/d[N]
for  i = N-1 to 0 do
   x[i] = (z[i]-c[i]*x[i+1])/d[i]
end for
return x[i]
```

The Thomas algorithm [239] is the simplest method used to solve a tridiagonal system of equations and is commonly employed on serial devices such as a CPU. The Thomas algorithm is a specialised case of gaussian elimination and can be derived from the LU decomposition of the matrix A. This reduces the system down to the solution of two bi-diagonal systems which can then be solved via gaussian elimination. The first system is solved via forward substitution and the second system is solved via backward substitution. These two stages will be referred to as the forwards and backwards iterations. The Thomas algorithm is given in Algorithm 6.1, it has a complexity of $O(N)$ and requires a total of 8N arithmetic operations to solve an N-tridiagonal system.

In parallel computing the Thomas algorithm is usually less favoured than algorithms such as recursive-doubling [240], cyclic-reduction [241] and parallel cyclic-reduction [242], since although these algorithms have a larger number of arithmetic operations some of the operations be can parallelised on devices such as GPUs [243] resulting in an overall lower algorithmic complexity. With a recent increased interest in FPGA acceleration attempts have been made to port tridiagonal solvers onto FPGAs [244] [245] [246] [247]; in this application the simplicity of the Thomas algorithm makes it well suited to the task when compared to cyclic-reduction which

maybe too complex for efficient data flow FPGA implementation.

6.2 Algorithmic Optimisation and Low Level Parallelism

Figures 6.1 and 6.2 depict the data dependency of the Thomas algorithm; it can be observed that in the forward iteration there are two separate branches of computation, one for calculating d_n and the other for z_n and hence this provides the first level of parallelism extracted. A similar approach has been taken by both Oliveira et al [244] and Warne et al [245]. This optimisation reduces the effective serial arithmetic operations down from 8N to 6N.

The problem with this simple optimisation is that although there is a reduction in serial operations, it has only reduced a multiply and a subtract which are computational cheap when compared to divisions. Consequently a competitive speed-up over faster clocking devices such as CPUs may not be obtained [245]. We thus introduce a simple algorithmic rearrangement that can allow for the two divisions

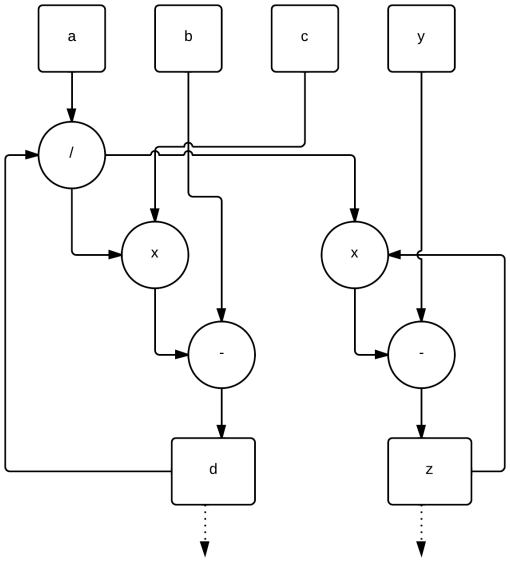


Figure 6.1: Data dependency graph for the forward iteration of the Thomas algorithm

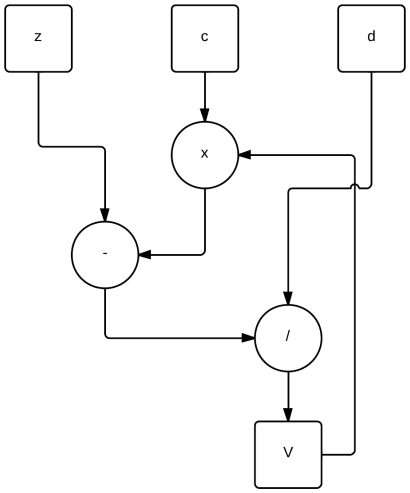


Figure 6.2: Data dependency graph for the backwards iteration of the Thomas algorithm

from the backwards and forward iterations to be effectively parallelised. Equation 6.6 shows the factorisation of the backwards iteration calculation where we now treat the divisions of z_n and c_n by d_n individually.

$$\frac{z_n - c_n V_{n+1}}{d_n} = \frac{1}{d_n} z_n - (\frac{1}{d_n} c_n) V_{n+1} \tag{6.6}$$

In a serial implementation this would add an extra division to the total number of arithmetic operations, which is not usually desirable, but as shown in figure 6.3 the data dependence is in fact here reduced, we can now treat these two divisions in parallel with each other whilst also performing them in parallel with the forward iteration calculations. This reduces the serial arithmetic operations down from the original 8N to 5N.

For FPGA implementations this rearranged algorithm has two advantages:

1. Total latency of the algorithm is almost halved by parallelising the two lengthy divisions.

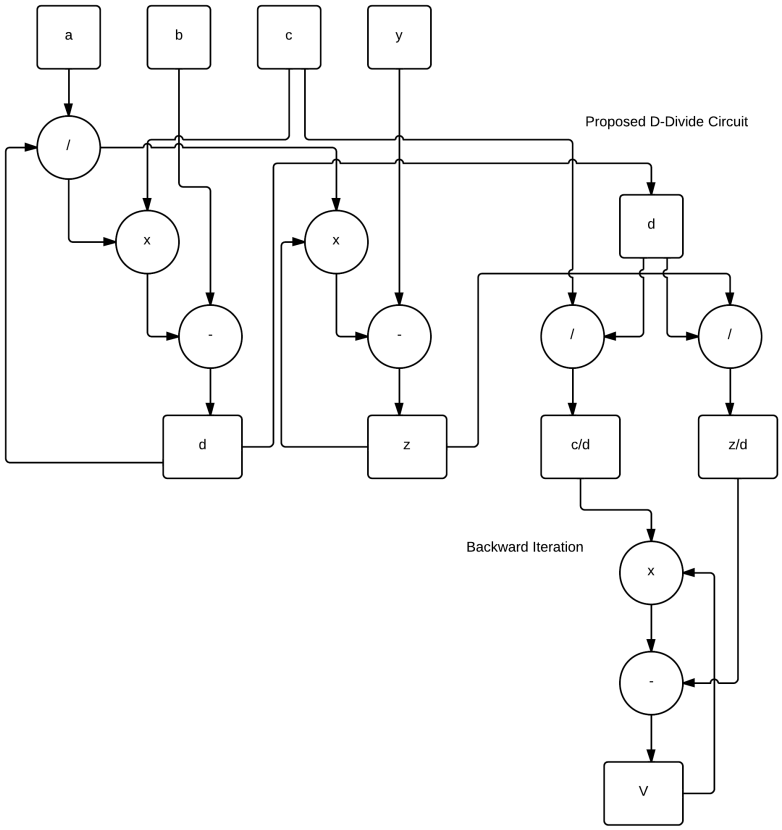


Figure 6.3: Data dependency graph for the proposed Thomas algorithm structure optimised for FPGA implementation.

2. Memory requirements are reduced from the need to save three intermediate data vectors (c, z and d) to two (c/d and z/d).

6.2.1 Pipelining

Further to the low level algorithmic optimisations higher level parallelism can be achieved in two ways: pipelining the data through the computation of the forward and backward iterations; and pipelining the sets of data between the forward and backward iterations, which has commonly been implemented for multiple CPU versions to parallelise the Thomas algorithm [248].

Firstly the computational units themselves can be deeply pipelined, an approach used by Olivera et al [244], which allows for multiple independent tridi-

agonal systems to be computed in the same iteration cycle. For example, if the forward iteration computational unit has P_F pipeline stages then throughout one iteration it is possible to fill each stage of the pipeline with a computation allowing for P_F independent tridiagonal systems to be computed.

The second type of pipelining of the algorithm means that given a set T of pipelined tridiagonal systems for either iteration, as discussed above, we can simultaneously compute the forward and backwards iterations of the two different sets (given the first set has already been through the forward iterations) independently in one Thomas solver. In [246] they look at using OpenCL and Xilinix HLS to build the Thomas solvers, but do not obtain this level of parallelism due to the complex scheduling involved for the pipelines. As such this design has been made in VHDL allowing this desired low level control to be obtained.

6.2.2 Hardware Architecture

The input to the solver core consists of 5 data items a,b,y and id where id is the local id of the system to be solved, this acts as a thread id and is important for addressing the correct memory stacks in the solver. The hardware architecture consists of four main components, the forward iteration core, the d-divider, the backwards iteration core and the stack array. The forward and backward cores contain the pipelined arithmetic for the stages of the algorithm, and the d-divider consists of two dividers to carry out the c/d and z/d computations. The stack array is used for storing the intermediate variables c/d and z/d, a stack can be used due to the nature of the problem which is that the backwards iterations first require the last values calculated by the d-divider, this saves unnecessary complications with memory addressing.

Connecting the forward iterations to the backwards iterations is a queue. This queue allows the problem index to be passed onto the backwards core for computation once the forward iterations have finished, this system allows for efficient independent operation of the forwards and backward iterations. The backwards core checks for space in the pipeline, and reads in the problem to begin computing if there is space, otherwise it remains queued.

In addition to the main Thomas algorithm core, the core has been placed in

a wrapper allowing for easy usability. The wrapper consists of fifo queues for the input data and output results , allowing for variable write and read times to and from the core, as well as the option for floating-point to fixed-point converters for input data and vice-versa for results.

When changing the arithmetic only the arithmetic cores are changed, and the architecture remains constant. The only variability with the arithmetic cores is the pipelining due to the differing latencies, but this is managed via adjustable parameters within the solver VHDL.

6.3 Design Analysis

Here we theoretically analyse the performance of the solver for solving multiple independent tridiagonal systems $\mathbf{T} = \{T_1^{N_1}, T_2^{N_2}, ..., T_M^{N_m}\}$, where M is the total number of independent tridiagonal systems to be solved and N_m is the size of the mth system. The notation is used in this work is as follows:

- $T_m^{N_m}$ is the size of the mth tridiagonal system to be solved of size N_m.
- $\mathbf{T^N}$ is a special case where for all $T_m \in \mathbf{T}$, $N_m = N$.
- $C^D_{\{+,-,/,\times\}}$ is the number of clock cycles taken for that arithmetic operation using data format D.
- $C_{F,B,A}$ is the number of clock cycles taken for a single forwards, F, and backwards, B, iteration and administration costs A.
- f is the clock frequency of the FPGA system.

The number of cycles taken for the iteration stages are:

$$C_F = C_{/}^{D} + C_{\times}^{D} + C_{-}^{D} \tag{6.7}$$

$$C_B = C_{\times}^{D} + C_{+}^{D} + C_{/}^{D} \tag{6.8}$$

with C_A being a constant determined by the programming of the algorithm.

To fully harness the power of the pipelined design it is desired that maximal throughput should be achieved by scheduling groups of independent computations.

Definition 6.1. The number of computational blocks, B, is defined as the number of subsets of independent tridiagonal systems to be solved. The set of of blocks given by $\mathbf{B} = \{\mathbf{b}_1^{m_1}, \mathbf{b}_2^{m_2}, ...\mathbf{b}_B^{m_B}\}$, where $\mathbf{b}_i^{m_i} \subset \mathbf{T}$ of size m_i such that:

$$\cup_{i=1}^{B} \mathbf{b}_i^{m_i} = \mathbf{T} \quad ; \cap_{i=1}^{B} \mathbf{b}_i^{m_i} = \emptyset \tag{6.9}$$

Thus for a given M the time to compute $\mathbf{T^N}$ is then given by:

$$t_{\mathbf{T^N}} = \frac{NB\left(C_F + C_A\right) + BC_{/} + NC_B + 2\sum_{b=1}^{B}\left(m_b - 1\right)}{f} \tag{6.10}$$

The partitioning of $\mathbf{T}$ into the set of blocks $\mathbf{B}$ can be effected by the data transfer rate r_d between the solver and the host system. The rate of computation, r_c, of the Thomas solver is given by:

$$r_c = \frac{5D}{f} \tag{6.11}$$

where D is the number of bits used to represent a number in the given format. This value is the rate at which data can be processed by the Thomas solver, the solver requires 5 inputs, a,b,c,y and id and can process a row every clock cycle.

The optimal number of blocks B can be obtained if the rate of transfer is quicker than the rate of computation, i.e. the solver can receive all the 5 values in

one clock cycle or less:

$$B_{opt} = floor\left(\frac{M}{C_F}\right) \; ; r_d \geq r_c \tag{6.12}$$

It maybe the case that the data transfer rate is slower than the rate of computation and hence the solver has to be stalled whilst waiting for the data. It is therefore desirable to compute the maximum number of tridiagonal systems in a block **b** in the pipeline without stalling for data. The number of blocks B is given by:

$$m_{opt} = ceil\left(\frac{r_c}{r_d}\right) \; ; r_d < r_c \tag{6.13}$$

$$B = floor\left(\frac{M}{m_{opt}}\right) \tag{6.14}$$

Maximum throughput for the solver can be obtained if the set of tridiagonal systems to be solved completely fills the pipeline of the solver:

$$M\%C_F = 0 \tag{6.15}$$

$$B > 1 \tag{6.16}$$

6.4 Numerical Bounds

To maximise performance it maybe required that custom data formats are used in the FPGA design. Fixed-Point arithmetic often provides faster and smaller FPGA designs, for example [249] [250], but at the cost of the loss of some precision in the results and a higher risk arithmetic overflow. Therefore it is important to know the range of values the solver is expected to use in the algorithm to allow for the custom data formats to be optimised for the problem. The preceding theorems presented require additional conditions that $b(n)$ is a positive monotonically increasing function of the row index, n i.e. $b_n < b_{n+1}$ and $|a_n| < 1$ and $|c_n| < 1 \; \forall i \leq N$. These theorems will be useful later for range bounding the implicit pricing problem. In these

following results the L-Infinity norm of the set of coefficients a, b or c, denoted by $\|x\|_\infty$, is used, the value of this norm is the largest absolute value in a set.

Theorem 6.1. *Given A is diagonally dominant by row or columns, and let A have LU factorisation $A = LU$. Then $\|d\|_\infty \leq 3\|b\|_\infty$*

Proof. Given in the proof of $|L||U| \leq 3|A|$ found in [251] pg.175 the following result is used:

$$|l_n c_{n-1}| + |d_n| \leq 3|b_n| \tag{6.17}$$

Simple rearrangement and the observation that the max will occur at the maximum absolute value gives the result of Theorem 6.1 ■

Theorem 6.2. *Given A is diagonal dominant by row, and let A have LU factorisation $A = LU$ then $|d_n| > |b_0 - \frac{\|a\|_\infty}{|b_0|}\|c\|_\infty| \quad \forall n$, given that $b(n)$ is a positive monotonically increasing function of the row index, i and $\Delta b \leq \|c\|_\infty$.*

Proof.

$$d_0 = b_0 \tag{6.18}$$

$$d_1 = b_1 - \frac{a_1}{b_0} c_0 \tag{6.19}$$

$$b_1 - \frac{\|a\|_\infty}{|b_0|}\|c\|_\infty \leq d_1 \tag{6.20}$$

Under the assumption that $b(n)$ is a positive monotonically increasing function then

$$b_0 - \frac{\|a\|_\infty}{|b_0|}\|c\|_\infty \leq b_1 - \frac{\|a\|_\infty}{|b_0|}\|c\|_\infty \leq d_1 \tag{6.21}$$

finally for this to hold over all cases it must enforced that $\|l\|_\infty \leq \frac{\|a\|_\infty}{|b_0|}$, which implies that:

$$b_0 \leq b_1 - \frac{\|a\|_\infty}{|b_0|}\|c\|_\infty \tag{6.22}$$

thus for this for hold $\Delta b \leq \|c\|_\infty$, given that $\|a\|_\infty < |b_0|$.

■

A more general theorem for all functions of $b(n)$ will be investigated, though Theorem 6.2 currently suffices for this work as will be seen later. The approach for finding a more general theorem should look at conditions for when a certain sequence of b provides a minimum for d_0 when compared to all other possible permutations.

Theorem 6.3. *Given A is diagonal dominant by row, and let A have LU factorisation $A = LU$ then $\|l\|_\infty < \frac{\|a\|_\infty}{|b_0 - \frac{\|a\|_\infty}{|b_0|}\|c\|_\infty|} < \frac{\|a\|_\infty}{|b_0|}$, given that $b(n)$ is a positive monotonically increasing function of the row index, n, and $\Delta b \leq \|c\|_\infty$.*

Proof. Using Theorem 6.2, the maximum value of l must be achieved when the largest value of a is divided by the smallest value of d.

■

In fact, although Theorem 6.1 provides an upper bound for the value of d, using the previous theorems a tighter more accurate bound can now be defined.

Theorem 6.4. *Given A is diagonal dominant by row, and let A have LU factorisation $A = LU$ then $d \leq \|b\|_\infty + \|l\|_\infty \|c\|_\infty$, given that $b(n)$ is a monotonically increasing function of the row index, n.*

6.4.1 Bounding the Thomas Algorithm

The first section describes various bounds for the LU decomposition of a matrix A. This forms the basis of the well known Thomas Algorithm used for solving tridiagonal inversion problems of the form $Tx = y$, where T is a tridiagonal matrix. The first stage of the algorithm is to apply LU decomposition to the matrix and then solve to auxiliary equations using forwards and backwards substitution.

Theorem 6.5. *Given a tridiagonal matrix T is diagonally dominant by row and let T have LU factorisation $T = LU$ with $\|l\|_\infty < 1$, then solving the first auxiliary equation of the inversion problem $Lz = y \quad ; z = Ux$, then $\|z\|_\infty < \|y\|_\infty \left(\frac{1}{1-\|l\|_\infty}\right)$*

Proof. First the term for z_N is expanded and using Theorem 6.3 it is possible to replace the individual l_i terms with the upper bound $\|l\|_\infty$,

$$\|z\|_\infty \leq |y|_N + \sum_{k=1}^{N-1} \|l\|_\infty^k |y_k|. \tag{6.23}$$

It is then possible to compact the telescopic sum into a geometric sequence, which has a maximum value when $i = N$ i.e using all of the terms. Given that the index of the the largest y value may not be known a larger bound can be formed by including this in the geometric sum as the final term.

In fact we can further loosen the bound by assuming that all values are the max, so

$$\|z\|_\infty \leq |y_N| + \sum_{k=1}^{N-1} \|l\|_\infty^k |y_k| \leq \|y\|_\infty \left(1 + \sum_{k=1}^{N-1} \|l\|_\infty^k\right) \tag{6.24}$$

using the formulas for the sum of a geometric sequence the max bound becomes

$$\|z\|_\infty \leq \|y\|_\infty \left(1 + \frac{\|l\|_\infty \left(1 - \|l\|_\infty^{N-1}\right)}{1 - \|l\|_\infty}\right) \tag{6.25}$$

and finally in the case that $\|l\|_\infty < 1$ a simpler form using the infinite geometric sum can be used:

$$\|y\|_\infty \left(1 + \frac{\|l\|_\infty \left(1 - \|l\|_\infty^{N-1}\right)}{1 - \|l\|_\infty}\right) < \|y\|_\infty \left(\frac{1}{1 - \|l\|_\infty}\right). \tag{6.26}$$

■

The hardware optimised algorithm presented here requires the calculation of two additional intermediates $\frac{c}{d}$ and $\frac{z}{d}$.

Theorem 6.6. *Given a tridiagonal matrix T is diagonally dominant by row and let T have LU factorisation $T = LU$ with $\|l\|_\infty < 1$, and previously stated conditions the intermediate value $|\frac{c}{d}| \leq \frac{\|c\|_\infty}{|b_0|}$.*

Theorem 6.7. *Given a tridiagonal matrix T is diagonally dominant by row and let*

T have LU factorisation $T = LU$ with $\|l\|_\infty < 1$ the intermediate value $|\frac{z}{d}| \leq \frac{\|z\|_\infty}{|b_0|}$.

Finally it is possible to then derive the bounds for the final values.

Theorem 6.8. *Given a tridiagonal matrix T is diagonally dominant by row and let T have LU factorisation $T = LU$ with $\|l\|_\infty < 1$, and previously stated conditions with $b_n > 1$ and $c_n < 1$ $\forall n \leq N$ then $\|v\|_\infty < \frac{\|z\|_\infty}{|b_0|-1}$*

Proof.

$$v_N = \frac{z}{d} \tag{6.27}$$

$$|v_N| < \frac{\|z\|_\infty}{|b_0|} \tag{6.28}$$

the recursion begins at v_{N-1}, it is possible to see that

$$|v_{N-1}| < \frac{\|z\|_\infty}{|b_0|} + V_N \frac{\|c\|_\infty}{|b_0|} < \frac{\|z\|_\infty}{|b_0|} + \frac{\|z\|_\infty}{|b_0|}\frac{\|c\|_\infty}{|b_0|} \tag{6.29}$$

expanding the recursion in this manner the result for a sequence given by

$$|v_n| < \sum_{i=0}^{N-n} \frac{\|c\|_\infty^i \|z\|_\infty}{|b_0|^{i+1}} \tag{6.30}$$

then assuming $|b_0| > 1$ and $\|c\|_\infty < 1$ the limit of this sequence is given by:

$$|v_n| < \sum_{i=0}^{N-n} \frac{\|c\|_\infty^i \|z\|_\infty}{|b_0|^{i+1}} < \frac{\|z\|_\infty}{|b_0| - 1} \tag{6.31}$$

■

Combing the previous theorems it is now possible to define a set of conditions that can ensure the absolute value of any variable in the algorithm does not exceed a certain bound. This will prove extremely useful when applying the fixed-point designs to given problems.

Theorem 6.9. *For a given integer Z ; $0 < Z$ there exists a set of conditions such that all intermediate variables in the Thomas algorithm can be bounded by Z, given that $b(n)$ is a positive monotonically increasing function of the row index, and $|a_n| < 1$, $|c_n| < 1$ and $|b_n| > 1$ $\forall n \leq N$. The following conditions are sufficient but not necessary:*

1. $\|l\|_\infty < 1$ *this implies* $\|a\|_\infty < |b_0|$
2. $\|y\|_\infty < Z\frac{|b_0| - \|a\|_\infty}{|b_0|+1}$
3. $\|c\|_\infty < |b_0|$
4. $\Delta b \leq \|c\|_\infty$

6.5 Hardware Implementation

The Thomas Solver hardware will be tested using the ZedBoard Xilinx Zynq7020 Evaluation Kit. The Zynq7020 is a system-on-chip which consists of two ARM-A9 processors connected to Xilinx Artix-7 FPGA fabric, allowing a high-speed interface between CPU and FPGA. Using the Zynq7020 the system of equations will be formulated in floating-point on the ARM-A9 CPU, these will then be transferred to the FPGA via AXI interfaces and solved using the FPGA Thomas solver. The fixed-point results are then converted back to floating-point and compared to the results for the same problem solved using floating-point arithmetic. A driver was also

	Arithmetic			
Resource	Floating	Fixed[2,30]	Fixed[2,22]	Fixed[2,14]
Flip-Flops	25721 (24%)	15369 (14%)	17224 (16%)	10711 (10%)
LUT	27204 (51%)	20722 (39%)	16998 (32%)	11894 (22%)
Mem-LUT	10547 (61%)	8683 (50%)	6174 (35%)	4294 (25%)
BRAM	35 (25%)	3 (2%)	3 (2%)	3 (2%)
DSP	6 (3%)	15 (7%)	9 (4%)	6 (3%)
Buft	1 (3%)	1 (3%)	1 (3%)	1 (3%)
Clock	100MHz	200MHz	200MHz	200MHz
Power (W)	1.932	1.788	1.648	1.568

Table 6.1: FPGA resources used for each design and percentages of resources used on the Xilinx Zynq7020

	Arithmetic			
Operation	Floating	Fixed[2,30]	Fixed[2,22]	Fixed[2,14]
Div (Radix-2)	28	61	52	36
Multiplier	12	6	6	6
Subtractor	4	2	2	2
Core				
Thomas Forward	44	69	60	44
Thomas Backward	16	8	8	8
Administration	3	3	3	3

Table 6.2: Clock cycle latency for each of the arithmetic cores on the FPGA, and the total latency of the Thomas solver forward and backward cores.

developed in Python to allow for the use of the FPGA device via a serial connection, although a serial connection is slow it provides a proof of concept that with further development this hardware could be connected via PCI-express.

6.5.1 FPGA Resource Usage

The results in table 6.1 were obtained post-implementation from the Vivado Design Suite, the base design used has $N_{max} = 512$ with 10 threads ($M_{max} = 10$), and variable arithmetic. A floating point design and three fixed-point solvers with the following data representation, [integer bits, fractional bits], are tested, 32bit[2,30], 24bit[2,22], 16bit[2,14]. For the arithmetic cores the provided Xilinx base IP cores were used, and set to make maximum usage of DSPs, and the Radix-2 divider algorithm is used for as part of the fixed-point divider.

Each of the solver designs have the same magnitude of latency, with the

	Max Throughput	Min Throughput
CPU(2.6GHz)	0.02000ms(1x)	0.020(ms)(1x)
Floating	0.00120ms(16x)	0.063ms(0.31x)
Fixed[2,30]	0.00055ms(36x)	0.040ms(0.50x)
Fixed[2,22]	0.00055ms(36x)	0.036ms(0.55x)
Fixed[2,14]	0.00057ms(35x)	0.028ms(0.72x)

Table 6.3: The average time(ms) for computing the solution to tridiagonal systems (N=100) on a desktop CPU and the implemented FPGA Thomas solver

floating-point design providing the lowest total latency, although the fixed-point designs maybe sped up by using higher radix divider algorithms. The disadvantage of the higher radix divider algorithms is that the maximum through-put is reduced due to the iterative nature of the algorithms, but this is useful if it is not possible to achieve maximum throughput of processing on tridiagonal system per clock. The main advantage of the fixed point solvers is the reduced resource usage, which provides the opportunity to maximise coarse grain parallelism by allowing more solver cores to fit onto a device and also increasing the maximum number of pipelined tridiagonal systems each core can solve. As can be expected the amount of memory resources is proportional to the total data width used for the fixed-point designs, whilst the floating-point solver, although 32bits wide, uses significantly more memory resources (BRAM and memory LUTs).

6.5.2 Performance

The latency performance of the solver can be evaluated using Equation 6.10 once the implemented FPGA clock speed is known. The floating-point was only able to achieve a maximum clock frequency of 100MHz whilst the fixed-point designs where able to achieve double this at 200MHz. Therefore although the fixed-point designs may have slightly higher latency in terms of clock cycles, the speed of computation is considerably faster due to this higher clock rate.

The average time in milliseconds per tridiagonal system is shown in Table 6.3 for minimum throughput, a single tridiagonal system, and maximum throughput, where the pipeline is completely full. These results are compared to the average time taken for a 2.6GHz on a top of the range desktop machine. If the solver was

to be used for single tridiagonal systems the speed is fractionally less than a top of the range 2.6GHz CPU, but this is not taking advantage of the pipelined design. At maximum throughput it is possible to achieve up to a 36x speed-up and 16x speed-up over a 2.6GHz CPU for FPGA fixed-point and floating-point designs respectively. In terms of cost of computing power the basic $200 FPGA board used here can outperform, in terms of speed, a $1000+ desktop computer, as well as also using considerably less power. This is due to the deep pipelining and custom data paths possible on an FPGA.

6.6 Implementation for Implicit Finite Difference Schemes

Here it is intended to evaluate the accuracy of the fixed-point Thomas solvers within the context of options pricing. When pricing options via implicit finite difference methods it often results in a tridiagonal or many systems of tridiagonal equations to be solved. As an example the solver will be used for solving tridiagonal systems arising in implicit finite difference scheme for european options using the Black-Scholes model.

6.6.1 Scaling For Fixed-Point Designs

The motivation is to use fixed-point arithmetic as previously discussed is that it results in smaller and faster designs when compared to floating-point. The tridiagonal coefficients for pricing a European option via implicit finite difference are given by:

$$
\begin{aligned}
a_n &= -(n^2\sigma^2 - nr)dt \\
b_n &= 1 + (n^2\sigma^2 + r)dt \\
c_n &= -(n^2\sigma^2 + nr)dt \\
a_N &= Nrdt \\
b_N &= 1 - (Nr - r)dt \\
y_n &= (S_n - K)^+
\end{aligned}
$$

where y is defined by the initial boundary condition of the problem, in this case the payoff function of the option.

Observing the coefficients $b_n > 1 \; \forall i \leq N$, as such two integer bits will be used for the fixed-point representation and a $Z = 2$ to ensure no arithmetic overflow. Using proposition 1 it is possible to show that the coefficients of implicit Black-Scholes pricing the algorithm can be bounded so that $Z = 2$ after applying and basic grid constraint to bound coefficient values and an appropriate transformation for the y values to meet condition 2 of Theorem 6.9. This is more formally expressed in Theorem 6.10.

Theorem 6.10. *Given a Black-Scholes Implicit pricing problem, A_I, it is possible to ensure that the supremum of the algorithm i.e. all values calculated in the algorithm, $sup(|A_I|) < Z$, given the following grid constraint and suitable linear transform on the problem domain.*

$$dt < \frac{1}{\sigma^2 N^2} \tag{6.32}$$

$$\hat{y}_n = f(y_n) \tag{6.33}$$

$$f(y_n) = y_n Z \frac{|b_0| - \|a\|_\infty}{(|b_0| + 1)\|y\|_\infty} \tag{6.34}$$

6.6.2 Fixed-Point Solver Accuracy

The fixed-point solver designs are tested over a sample of 5000 randomly selected tridiagonal equations generated by random option pricing problems. The two market dependant parameters, interest rate r and volatility σ, are randomly chosen for each option sample to generate a new sample of tridiagonal equations to solve; $r = U[0.01, 0.05]$, $\sigma = U[0.10, 0.30]$. The finite difference grid parameters are selected to meet the constraint in equation 6.32 in the case of maximum market parameter values, this resulted in $dt = 0.001$. Finally to meet the final for $\|y\|$ a linear transform constant of $\frac{0.45Z}{\|y\|_\infty}$ was used, calculated using equation 6.34, to meet this condition the problem was chosen so that $S_N = 2$ and $K = 1$.

Table 6.4 gives the expected rounding error for the number of fractional bits used in the fixed-point design. The expected rounding error $e_{rnd}(x, f)$, where e_{rnd}

	Fractional Width		
	30	22	14
Expected Rounding Error	2.33E-10	5.95E-08	1.52E-05
Maximum FPGA Error	4.06E-08	4.88E-07	1.23E-04

Table 6.4: Comparison of expected rounding error and maximum absolute error from the FPGA implementation.

is the rounding error and f is the number of fractional bits, for rounding a floating-point number x to a fixed-point representation is given by:

$$\mathbb{E}(e_{rnd}(x,f)) = \frac{2^{f-1}}{2}\ ; x \in U[0,\infty] \tag{6.35}$$

It has been assumed that the rounding error is uniformly distributed white noise over $x \in \mathbb{R}$ [252]. If an error obtained is smaller than this value indicates that the fixed-point value was rounded to 0, and the actual value is smaller than is possible to represent in the fixed-point representation. Errors within a similar magnitude as the magnitude of the expected error indicate that the fixed-point result is on average as accurate as is possible for the given fixed-point representation.

Figure 6.4 shows the absolute error with respect to the floating point result for the fixed-point solver using 30 fractional bits. The most striking feature of this plot is how the error resembles the shape of the payoff function indicating that the magnitude of the option price plays a role in the error function. A worst case error function has been derived from the observation that an option price, V , $V < S_n$, i.e the european option price must be at least less than the asset price due the the effect of the strike. The maximum error is then a function of the asset price, minimum expected rounding error and n to take into account error prorogation factors through the iterations.

$$E(S_n) = nS_n\frac{2^{f-1}}{2} \tag{6.36}$$

Figures 6.5 and 6.6 show the absolute errors for the fixed-point solver with 22 and 14 fractional bits respectively. Unlike the 30 fractional bits both of sets of errors show a shape differing from the one observed for 30 fractional bits, with a

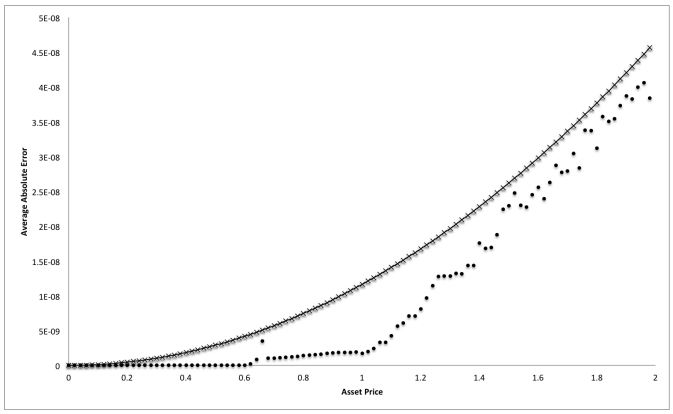


Figure 6.4: Average absolute error over 5000 tridiagonal systems of the fixed-point results using 30 fractional bits with respect to floating-point results. -x- - estimated maximum error bound using equation 6.36.

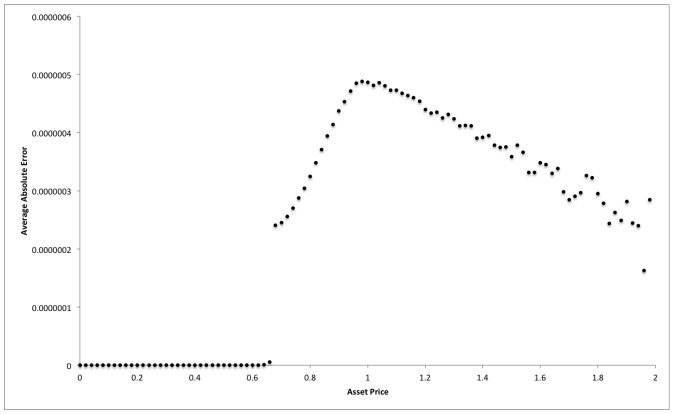


Figure 6.5: Average absolute error over 5000 tridiagonal systems of the fixed-point results using 22 fractional bits with respect to floating-point results.

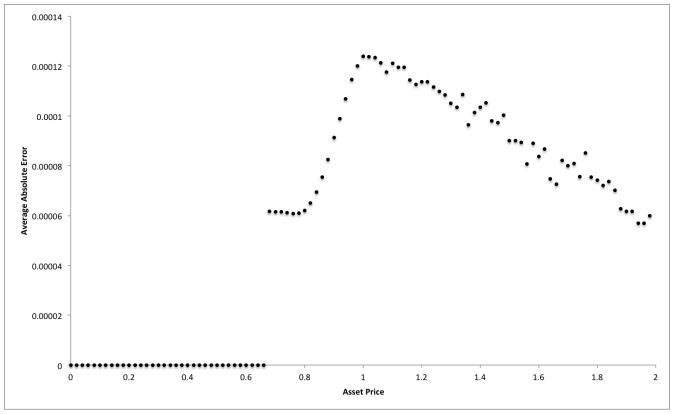


Figure 6.6: Average absolute error over 5000 tridiagonal systems of the fixed-point results using 14 fractional bits with respect to floating-point results.

peak near the strike price and then descending again. Although they both show and different shape to the 30 fractional bits, their respective absolute errors with respect to their minimum fractional resolution is a lot better, with the largest magnitude of error being of the same order, this is up to 100 times smaller in magnitude than the error predicted by equation 6.36.

These results show that the the fixed-point arithmetic is accurate up to a given decimal place, which then after the accuracy begins to degrade. This explains why the 30 fractional bit errors were a lot larger than its respective minimum fractional resolution and not so for the 22 and 14 fractional bits.

6.7 Conclusion

This work has proposed and introduced a prototype design for a high performance FPGA based tridiagonal solver. Fixed-point designs can be used to minimise resource usage and obtain higher clock rates compared to floating point designs. When compared to a 2.6GHz CPU on a top of the range desktop it was possible to achieve up to a 36x speed-up and 16x speed-up for the fixed-point and floating-point designs respectively. For the fixed-point designs the errors introduced in the results due to the limited fractional resolution was investigated. Overall in the implicit option pricing example the errors were well behaved with the maximum for the 22 and 14 bit fractional representations only being 10x that of the expected rounding error, and 50x for 30 fractional bits. It is intended that this work is further integrated into a larger FPGA based implicit pricing system to achieve a high speed and low cost solution for accelerating options pricing.

Chapter 7

Conclusions and Future Work

This thesis has addressed a number of problems in computational finance, characterised by their high demand for computation time and resources. The aim of the work has been to offer alternative methodologies to solve these problems, applying a combination of evolutionary algorithms (EAs) and neural networks, with a possibility of further acceleration using custom hardware.

In Chapter 3, Breeding Particle Swarm Optimisation (BrPSO), a novel variant of particle swarm optimisation (PSO) using mutation, was introduced. This provided a competitive optimisation algorithm when compared to other state of the art EAs on sets of benchmark functions. BrPSO was further shown, when combined with neural networks, to be a powerful function approximator, as evidenced by its performance on a problem in marine engineering (approximating the function that describes the water resistance offered to a ship with a novel hull-form). Thus BrPSO would appear competitive as a candidate algorithm for optimisation problems in computational finance

In Chapter 4 various forms of PSO and state of art differential evolution algorithms (DEs, a form of EA) were used for calibration of the Heston model. BrPSO was seen to be the best of the PSO algorithms, though L-SHADE, a form of DE, proved to be by far the most accurate optimiser. Strengths and weaknesses were however seen in both categories of algorithm, DEs displaying good final exploitation behaviour whilst PSO showed good initial exploration, so being more likely to locate the global basin. Nelder-Mead (NM) local search was added, with the hope

that this would enhance final exploitation, being combined in a PSO-L-SHADE-NM hybrid; this was by far the most efficient algorithm, in terms of fitness function evaluations to achieve a given level of accuracy. This Chapter also contained an important observation pertaining to the use of numerical integration, which can create apparent local minima, a topic which in relation to the Heston model calibration problem has been highly debated. The work here concludes that the calibration surface is in fact an inherently unimodal valley, as previously shown by Cui *et al* [127], but that local minima can be introduced as an artefact by the numerical integration scheme. It is therefore recommended that safeguards are used to protect against possible instability in the fitness function.

Chapter 5 applied BrPSO and neural networks to the problem of providing approximate solutions for options pricing. A two-layer MLP was used, with linear regression to create ensembles. Initial investigation focused on European options because the results of the approximator could be compared not only to MC but also to the analytic solution provided by Black-Scholes. The mean prices obtained from the ensembles were better than prices from their component networks but there was room for improvement as the results were not yet competitive with the MC used here. An improvement was obtained by using linear regression to derive the ensemble weights. Two forms of linear regression were considered: shortest path least squares, which produces non-convex weights; and constrained non-negative least squares, which leads to convex weights. Non-convex weights provided most the accurate results and were more competitive with MC (in fact being better than MC for ATM options); however the presence of negative weights causes some concerns as to the method's robustness (the range of the ensemble output being unbounded). The convex weights were less competitive; however with the use of the inverse hyperbolic sine transform their performance was improved. Convex weights have the benefit that because the ensemble output is bounded they are more robust, and in addition the ensembles using them can be pruned so as to make them smaller. However it should be noted that despite these advantages ensembles with convex weight; are still not as accurate for this options pricing problem as ones using non-convex

weights. Ensembles with non-convex weights were therefore chosen to be applied to exotic (path-dependent) options, specifically Asian and American options, with good pricing results, though small improvements are still needed in the case of American options in order to reach acceptable error tolerances.

Chapter 6 explored the use of custom hardware for accelerating numerical options pricing, in particular the application of finite difference methods. Tridiagonal systems of equations very frequently occur in finite difference calculations and are the main computational bottleneck of these methods. One algorithm used to solve a tridiagonal system of equations is the Thomas algorithm. Using field programmable gate arrays (FPGAs) a low level parallelised implementation of the Thomas algorithm was developed. Using a data flow design paradigm and fixed-point arithmetic the FPGA implementation of the Thomas algorithm was able to calculate the solution of many tridiagonal systems simultaneously. A mathematical analysis was able to provide bounds for the fixed-point arithmetic calculations involved in the Thomas algorithm allowing the accuracy of the solver to be maximised with respect to the number of bits used. The accuracy of the FPGA Thomas solver was tested for sets of example tridiagonal systems generated from implicit finite different schemes for European options pricing; it was seen that the FPGA implementation was able to accurately and efficiently solve these systems of equations.

The primary result of this thesis that was presented in Chapter 5, shows that neural networks and EAs can be used, after a one-time, offline training period, to provide very fast approximate solutions to options pricing problems, with accuracies comparable to traditional Monte Carlo methods. Overall the work presented here has shown promise in the application of new methodologies to computationally demanding problems in finance, though there is scope for further work, as will be briefly discussed below.

7.0.1 Future Work

The work presented in this thesis offers many opportunities for further development and exploration; these will be considered on a chapter-by-chapter basis.

Chapter 3. It would be valuable to explore improving BrPSO by applying crossover operators hybridised with more advanced PSO algorithms. It is also worthwhile investigating the effects the crossover rate and breeding probability have on the search capability for artificial benchmark function optimisation problems, and potentially incorporating these parameters into the self adaptation mechanism. Further research for the self-adaptation mechanism could look to use elements inspired from the historical archive mechanisms used in the SHADE algorithms [73].

Chapter 4. In addition to calibrating the Heston model, more complex multi-factor models such as Heston-Bates model [6] should be investigated. The Heston-Bates model is sensitive to the jump parameters, and gradient-descent methods often struggle, the power of BrPSO and L-SHADE can be used to better search this highly multimodal search space. Further investigation should also be pursued in the understanding of how local minima structures form in the parameter search space as an artefact of numerical integration, and explore the use of other numerical integration and pricing methods.

Chapter 5. The work here showed that neural networks accurately price European and Asian options, however the methodology presented showed further scope for improvement with regards to American options. Further work should investigate how these early exercise features can be better learnt or incorporated into the neural network models. Other types of multi-factor options such a basket options should also be investigated. Basket options rely on multiple asset price simulations and hence an accurate neural network solution can provide even greater gains in efficiency. Apart from investigating different types of options contracts more complex multi-factor models such as the Heston model should be further explored. To enhance the presented methodology evolving of self-adapting neural network architectures should be investigated, this will allow optimal architectures with respect to size and approximation accuracy to be found. It may also be worthwhile investing

the use of training methods such as negative correlation learning [220] to increase ensemble diversity. Another important avenue is to explore how the use of trial solutions could further assist the neural network learning by reducing the complexity of the target function and automatically satisfying boundary conditions. Finally work looking at how neural networks can be used to directly solve the resultant initial boundary partial differential equations should be explored, as this removes any dependancy on other numerical methods.

Chapter 6. This work can be advanced by exploring FPGA implementations of the more inherently parallel cyclic reduction algorithms used for solving tridiagonal systems, and overall efficiency when tested as part of a 3D alternate-direction-implicit (ADI) solver. Finally one could build the neural network models from Chapter 5 on the custom hardware devices to provide efficient specialised pricing hardware.

These extensions to the work of the thesis would be of value to pursue. However the work presented here already suggests there is a promising future for the use of EAs, neural networks - and possibly, in combination with these algorithmic methodologies, custom hardware - in computational finance.

Appendix A

Additional Mathematical Results

The well known results, used in Section 3.3, for the volume of an n-ball, and the length of a side of the maximum inscribed hypercube inside the n-ball are given bellow:

Corollary A.1. *The maximum side length L for an n-hypercube* $\mathbf{H}_L$ *fully inscribed inside an n-ball with radius r is*

$$L = \frac{r}{\sqrt{n}}. \tag{A.1}$$

This gives a maximum n-volume of

$$V_n(\mathbf{H}) = (\frac{2r}{\sqrt{n}})^n \tag{A.2}$$

Corollary A.2. *The the n-volume of an n-ball with radius r is*

$$V_n(\mathbf{S}_r) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)} r^n \tag{A.3}$$

Appendix B

Benchmark Functions

The test functions used here are a set of well known unimodal, $f1 - f2$, and multimodal problems, $f3 - f8$; the complexity of the set of test functions is increased by adding a set of rotated-multimodal problems, $f9 - f14$ [253]. The rotated set of test functions are created for functions $f3 - f8$, applying random orthogonal rotations to the input vectors generating test functions $f9 - f14$. The rotations are applied using an orthogonal matrix, $\mathbf{M}$. This is then applied to the $\mathbf{x}$ vector which now creates dependancies between the x_i values in each dimension in terms of the new input vector $\mathbf{y}$ whilst retaining the shape and minima of the original function.

$$f_R(\mathbf{y}) = f_k(\mathbf{M}\mathbf{x}). \tag{B.1}$$

The addition of the inter-dimensional dependancies due to rotation makes the problems much harder due to the fact that the D-dimensional search problem can now not be directly solved using D independent searches.

In the CEC'05 [253] set of benchmark functions, the rotated functions are additionally shifted such that the minimum no longer lies at $\mathbf{0}$; this stops any center biased algorithms or algorithms that may naturally converge to $\mathbf{0}$ having an inbuilt advantage. In addition, composite functions are used to introduce a set of random multimodal functions; the composite functions are formed as a mixture of basic functions, such as the sphere function, $f1$, with shifted optima. These produce

	Name	Function	Search range	Initialisation range	$\hat{\mathbf{x}}$
f1	Sphere	$\sum_{i=1}^{D} x_i^2$	[-100, 100]	[-100, 50]	**0**
f2	Rosenbrock	$f_2(x) = \sum_{i=1}^{D-1}(100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2)$	[-2.048, 2.048]	[-2.048, 2.048]	**1**
f3	Ackely	$-20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2})$ $-\exp(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)) + 20 + e$	[-32.768, 32.768]	[-32.768, 16]	**0**
f4	Griewank	$1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)$	[-600, 600]	[-600, 200]	**0**
f5	Weistrass	$\sum_{n=1}^{D}\left(\sum_{k=0}^{kmax}\left(a^k\cos\left(2\pi b^k(x_n + 0.5)\right)\right)\right)$ $-D\sum_{k=0}^{\text{kmax}}\left(a^k\cos\left(2\pi b^k(x_n + 0.5)\right)\right)$ $a = 0.5 b = 3, kmax = 20$	[-0.5, 0.5]	[-0.5, 0.2]	**0**
f6	Rastrigin	$\sum_{n=1}^{D}\left(x_n^2 - 10\cos(2\pi x_n) + 10\right)$	[-5.12, 5.12]	[-5.12, 2]	**0**
f7	Non.Cont. Rastrigin	$\sum_{n=1}^{D}\left(y_n^2 - 10\cos(2\pi y_n) + 10\right)$ $y_n = \begin{cases} x_n, & \lvert x_n\rvert < 0.5 \\ \frac{round(2x_n)}{2}, & \lvert x_n\rvert >= 0.5 \end{cases}$	[-5.12, 5.12]	[-5.12, 2]	**0**
f8	Schwefel	$418.9829D - \sum_{n=1}^{D} x_n \sin\left(\lvert x_n\rvert^{0.5}\right)$	[-500, 500]	[-500, 500]	**420.96**
f9	Rot Ackely	$f3(\mathbf{Mx})$	[-32.768, 32.768]	[-32.768, 16]	**0**
f10	Rot Griewank	$f4(\mathbf{Mx})$	[-600, 600]	[-600, 200]	**0**
f11	Rot Weistrass	$f5(\mathbf{Mx})$	[-0.5, 0.5]	[-0.5, 0.2]	**0**
f12	Rot Rastrigin's	$f6(\mathbf{Mx})$	[-5.12, 5.12]	[-5.12, 2]	**0**
f13	Rot NCnt. Rastrigin	$f7(\mathbf{Mx})$	[-5.12, 5.12]	[-5.12, 2]	**0**
f14	Rot Schwefel	$418.9829D - \sum_{n=1}^{D} z_n$ $z_n = \begin{cases} y_n \sin\left(\lvert y_n\rvert^{0.5}\right) & \lvert y_n\rvert <= 500 \\ 0.001\left(\lvert y_n\rvert - 500\right)^2 & \lvert y_n\rvert > 500 \end{cases}$ $\mathbf{y} = \mathbf{Mx}$	[-500, 500]	[-5.12, 5.12]	**420.96**

Table B.1: Set of non-rotated and rotated (rot) benchmark functions used.

a multi-modal function with one randomly placed global optimum and numerous deep local optima. The functions were designed with a local optimum placed at $\mathbf{x} = \mathbf{0}$; this is important for catching algorithms that converge towards $\mathbf{0}$, which may be seen to work well on the standard set of test functions but may not work so well in more complex search spaces.

Appendix C

Additional Calibration Results

Standard deviations for mean Euclidian distances										
	1	2	3	4	5	6	7	8	9	10
PSO-gB	3.64e-03	1.04e+00	5.22e-03	1.84e-04	1.77e-03	5.80e-02	3.24e-02	2.20e-02	6.00e-02	1.31e-01
PSO-gB-cf	3.92e-04	1.19e+00	7.94e+00	4.93e-09	8.83e-07	1.71e-02	7.88e-03	4.31e-03	3.73e-02	1.04e-01
PSO-lB-cf	1.77e-03	4.60e-01	9.92e-04	3.31e-06	1.58e-04	4.47e-02	3.19e-02	3.26e-02	8.99e-02	1.03e-01
BrPSO	2.16e-06	2.06e+00	1.08e+01	2.13e-12	3.06e-10	1.74e-03	5.60e-04	7.51e-05	2.83e-03	3.18e-02
BrPSOSAM	6.78e-05	9.71e-01	2.52e-05	2.29e-08	2.08e-06	6.57e+00	6.41e-03	9.66e-03	5.28e-02	1.08e-01
CLPSO	1.97e-01	4.11e-01	1.84e-01	2.24e-01	3.76e-02	3.87e-01	1.66e-01	1.47e-01	1.36e-01	2.11e-01
CPSO	1.04e+00	1.56e+00	1.06e+00	9.35e-01	4.64e-01	1.62e+00	1.11e+00	8.39e-01	7.28e-01	4.28e-01
UPSO	5.96e-03	5.69e-01	2.22e-02	1.31e-05	1.02e-03	6.50e-02	6.10e-02	3.63e-02	1.07e-01	1.82e-01
wFIPS	5.41e-02	6.01e-01	2.92e-02	6.23e-03	3.91e-03	1.51e-01	1.03e-01	7.70e-02	7.22e-02	1.01e-01
FDR	1.75e-03	1.33e+00	7.54e-04	1.67e-05	2.03e-04	4.11e-02	2.58e-02	1.88e-02	5.81e-02	9.93e-02
DE/r/1/b	2.16e-03	5.04e+01	5.19e+01	7.73e-05	5.33e-06	5.24e+01	2.28e-05	8.34e-06	1.24e-05	3.11e-05
DE/r/1/e	4.87e-03	3.31e+01	1.61e+00	4.80e-04	1.07e-05	3.35e+01	3.41e-05	2.82e-05	1.54e-05	1.96e-05
JADE	2.62e-11	1.81e-01	4.31e-12	2.84e-11	1.58e-11	9.71e-04	2.37e-11	6.40e-12	1.16e-12	1.17e-10
jDE	3.23e-08	9.83e-02	4.92e-08	3.80e-08	9.58e-09	7.42e-03	8.42e-06	6.15e-08	4.95e-07	7.98e-07
SHADE	4.63e-12	1.67e-01	2.63e-12	1.21e-12	8.12e-13	5.51e-10	7.71e-12	1.02e-12	1.94e-12	3.22e-12
L-SHADE	1.64e-13	2.16e-01	6.95e-13	3.36e-13	2.26e-14	6.20e-13	6.02e-14	2.71e-14	1.76e-14	2.49e-14
PSO-gB-NM	1.36e-05	1.73e+00	7.67e-06	2.44e-05	8.69e-06	1.19e-05	4.27e-05	1.19e-05	1.36e-05	1.89e-05
DE/r/1/b-NM	1.11e-05	2.74e+00	1.09e-05	5.17e-05	7.36e-06	1.20e-05	5.84e-05	1.48e-05	1.38e-05	2.13e-05

Table C.1: Respective standard deviations for the mean Euclidian distance metrics given in Table 4.3.

	Mean			Q75		
	Mean Std	Log Mean Std	SLM Std	Mean Std	Log Mean Std	SLM Std
PSO-gB	5.25e-02	-1.66	0.61	7.00e-02	-1.53	0.69
PSO-gB-cf	2.54e-01	-2.69	0.28	3.87e-02	-3.27	0.14
PSO-lB-cf	4.66e-02	-2.14	0.40	6.56e-02	-2.03	0.44
BrPSO	3.57e-01	-4.14	-0.05	6.80e-03	-5.00	-0.20
BrPSOSAM	1.65e-01	-2.97	0.22	3.96e-02	-3.11	0.20
CLPSO	2.69e-01	-0.59	0.80	3.86e-01	-0.45	0.86
CPSO	1.14e+00	0.06	1.06	1.55e+00	0.18	1.15
UPSO	8.09e-02	-1.72	0.54	1.15e-01	-1.65	0.59
wFIPS	1.14e-01	-1.13	0.68	1.41e-01	-1.02	0.74
FDR	4.58e-02	-2.01	0.50	6.42e-02	-1.88	0.59
DE/r/1/b	8.02e+00	-2.78	0.51	7.80e+00	-3.35	0.44
DE/r/1/e	5.22e+00	-2.81	0.45	7.78e+00	-3.39	0.22
JADE	1.97e-05	-9.43	-1.44	3.12e-10	-10.10	-1.57
jDE	6.34e-04	-6.00	-0.59	1.24e-03	-6.03	-0.56
SHADE	2.83e-11	-10.45	-1.76	1.30e-11	-10.39	-1.69
L-SHADE	2.77e-13	-11.84	-2.09	3.75e-13	-11.76	-1.99
PSO-gB-NM	2.27e-05	-4.20	-0.06	3.13e-05	-4.09	-0.03
DE/r/1/b-NM	3.13e-05	-4.10	-0.05	4.16e-05	-4.00	-0.02

Table C.2: Standard deviations for the total error measures, mean, log mean and standardised log mean (SLM) given in Table 4.4.

	FE = 1000			FE = 5000		
	Mean Std	Log Mean Std	SLM Std	Mean Std	Log Mean Std	SLM Std
PSO-gB	0.54	0.18	0.67	0.28	0.32	0.37
PSO-gB-cf	0.80	0.18	0.40	0.56	0.59	0.35
PSO-lB-cf	0.77	0.17	0.44	0.54	0.42	0.30
BrPSO	1.03	0.26	0.58	0.74	0.85	0.47
BrPSOSAM	0.96	0.21	0.79	0.85	0.38	0.57
CLPSO	0.46	0.14	0.39	0.34	0.30	0.24
CPSO	0.78	0.13	0.65	0.65	0.20	0.41
UPSO	0.85	0.19	0.54	0.55	0.45	0.15
wFIPS	0.70	0.17	0.42	0.49	0.30	0.25
FDR	0.59	0.20	0.42	0.49	0.34	0.30
DE/r/1/b	1.33	0.19	0.51	1.07	0.79	0.80
DE/r/1/e	1.22	0.18	0.45	0.82	0.75	0.61
JADE	0.56	0.07	0.59	0.41	0.09	0.33
jDE	0.51	0.06	0.65	0.40	0.10	0.35
SHADE	0.50	0.20	0.39	0.35	0.76	0.16
L-SHADE	0.66	0.23	0.44	0.49	1.07	0.09
PSO-gB-NM	0.60	0.28	0.72	0.58	1.75	0.75
DE/r/1/b-NM	1.24	0.32	1.05	0.80	1.79	1.20

Table C.3: Standard deviations for the total error measures, mean, log mean and standardised log mean (SLM) given in Table 4.5.

$\sqrt{v_0}$	$\sqrt{\theta}$	ρ	κ	σ	Fitness
0.190	0.134	0.938	2.270	0.179	28.166
0.195	0.141	0.983	3.197	0.199	27.011
0.179	0.539	0.959	0.000	0.143	37.232
0.190	0.150	0.865	3.197	0.211	28.144
0.187	0.145	0.689	3.554	0.312	31.865
0.189	0.166	0.715	8.389	0.342	30.788
0.188	0.147	0.798	3.063	0.224	29.153
0.189	0.152	0.850	3.468	0.209	28.380
0.190	0.135	0.981	2.324	0.168	28.109
0.194	0.158	0.889	4.889	0.231	27.382
0.185	0.154	1.000	2.593	0.152	29.306
0.194	0.000	0.860	0.463	0.253	36.530
0.194	0.154	0.897	4.750	0.219	27.254
0.201	0.141	0.940	4.557	0.246	26.506
0.196	0.162	0.762	7.965	0.328	28.726
0.193	0.161	0.908	5.455	0.218	27.371

Table C.4: Local optima where $\rho > 0$ found by the evolutionary algorithms for calibrating Heston parameter 2.

Bibliography

[1] G. Poitras. The Early History of Option Contracts. In *Vinzenz Bronzin's Option Pricing Models*, pages 487–518. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[2] BIS. Statistical Release: OTC Derivatives Statistics at End-June 2018. Technical report, 2018.

[3] I. de Pinto. An Essay on Circulation and Credit, in Four Parts, and a Letter on the Jealousy of Commerce;. page 248, 1774.

[4] S. L. Heston. A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *Review of Financial Studies*, 6(2):327–343, apr 1993.

[5] L. A. Grzelak and C. W. Oosterlee. On the Heston Model with Stochastic Interest Rates. *SIAM Journal on Financial Mathematics*, 2(1):255–286, jan 2011.

[6] D. S. Bates. Jumps and Stochastic Volatility: Exchange Rate Processes Implicit in Deutsche Mark Options. *Review of Financial Studies*, 9(1):69–107, jan 1996.

[7] C. W. Reynolds. Flocks, Herds and Schools: A Distributed Behavioral Model. *ACM SIGGRAPH Computer Graphics*, 21(4):25–34, aug 1987.

[8] R. Eberhart and J. Kennedy. A New Optimizer Using Particle Swarm Theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43. IEEE.

[9] M. M. Millonas. *Swarms, Phase Transitions, and Collective Intelligence*. Addison-Wesley, 1992.

[10] J. Kennedy. The Particle Swarm: Social Adaptation of Knowledge. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, pages 303–308. IEEE.

[11] R. Tanabe and A. Fukunaga. Success-History Based Parameter Adaptation for Differential Evolution. In *2013 IEEE Congress on Evolutionary Computation*, pages 71–78. IEEE, jun 2013.

[12] D. Bratton and J. Kennedy. Defining a Standard for Particle Swarm Optimization. In *2007 IEEE Swarm Intelligence Symposium*, pages 120–127. IEEE, apr 2007.

[13] M. Clerc. Initialisations for Particle Swarm Optimisation. Technical report, 2008.

[14] D. N. Wilke, S. Kok, and A. A. Groenwold. Comparison of Linear and Classical Velocity Update Rules in Particle Swarm Optimization: Notes on Scale and Frame Invariance. *International Journal for Numerical Methods in Engineering*, 70(8):985–1008, may 2007.

[15] Y. Shi and R. Eberhart. A Modified Particle Swarm Optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pages 69–73. IEEE.

[16] R. Poli and D. Broomhead. Exact Analysis of the Sampling Distribution for the Canonical Particle Swarm Optimiser and Its Convergence During Stagnation. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation - GECCO '07*, page 134, New York, New York, USA, 2007. ACM Press.

[17] L. Chuan and F. Quanyuan. The Standard Particle Swarm Optimization Algorithm Convergence Analysis and Parameter Selection. In *Third International Conference on Natural Computation (ICNC 2007)*, pages 823–826. IEEE, 2007.

[18] Jong-Bae Park, Yun-Won Jeong, Joong-Rin Shin, and K. Lee. An Improved Particle Swarm Optimization for Nonconvex Economic Dispatch Problems. *IEEE Transactions on Power Systems*, 25(1):156–166, feb 2010.

[19] R. Eberhart and Y. Shi. Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, volume 1, pages 84–88. IEEE.

[20] M. Clerc and J. Kennedy. The Particle Swarm - Explosion, Stability, and Convergence in a Multidimensional Complex Space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[21] Zhi-Hui Zhan, Jun Zhang, Yun Li, and H.-H. Chung. Adaptive Particle Swarm Optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(6):1362–1381, dec 2009.

[22] A. Ismail and A. P. Engelbrecht. Self-Adaptive Particle Swarm Optimization. pages 228–237. Springer, Berlin, Heidelberg, 2012.

[23] J. Ma, J. Zhang, and L. Xu. Staying Together Maybe Better for Particles. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 204–211. IEEE, may 2015.

[24] J. Kennedy. Bare Bones Particle Swarms. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)*, pages 80–87. IEEE.

[25] R. A. Krohling and E. Mendel. Bare Bones Particle Swarm Optimization with Gaussian or Cauchy Jumps. In *2009 IEEE Congress on Evolutionary Computation*, pages 3285–3291. IEEE, may 2009.

[26] D. Schor, W. Kinsner, and J. Anderson. A Study of Optimal Topologies in Swarm Intelligence. In *CCECE 2010*, pages 1–8. IEEE, may 2010.

[27] J. Kennedy. Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, pages 1931–1938. IEEE.

[28] J. Kennedy and R. Mendes. Population Structure and Particle Swarm Performance. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1671–1676. IEEE.

[29] A. Engelbrecht. Particle Swarm Optimization: Global Best or Local Best? In *2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 124–135. IEEE, sep 2013.

[30] K. Parsopoulos and M. Vrahatis. Unified Particle Swarm Optimization for Tackling Operations Research Problems. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 53–59. IEEE.

[31] R. Mendes, J. Kennedy, and J. Neves. The Fully Informed Particle Swarm: Simpler, Maybe Better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210, jun 2004.

[32] J. Liang, A. Qin, P. Suganthan, and S. Baskar. Comprehensive Learning Particle Swarm Optimizer for Global Optimization of Multimodal Functions. *IEEE Transactions on Evolutionary Computation*, 10(3):281–295, jun 2006.

[33] Z.-H. Zhan, J. Zhang, Y. Li, and Y.-H. Shi. Orthogonal Learning Particle Swarm Optimization. *IEEE Transactions on Evolutionary Computation*, 15(6):832–847, dec 2011.

[34] H. Huang, H. Qin, Z. Hao, and A. Lim. Example-Based Learning Particle Swarm Optimization for Continuous Optimization. *Information Sciences*, 182(1):125–138, 2012.

[35] M. Clerc. Standard Particle Swarm Optimisation. 2012.

[36] M. Zambrano-Bigiarini, M. Clerc, and R. Rojas. Standard Particle Swarm Optimisation 2011 at CEC-2013: A Baseline for Future PSO Improvements. In *2013 IEEE Congress on Evolutionary Computation*, pages 2337–2344. IEEE, jun 2013.

[37] M. R. Bonyadi and Z. Michalewicz. SPSO 2011. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation - GECCO '14*, pages 9–16, New York, New York, USA, 2014. ACM Press.

[38] M. R. Bonyadi and Z. Michalewicz. Analysis of Stability, Local Convergence, and Transformation Sensitivity of a Variant of the Particle Swarm Optimization Algorithm. *IEEE Transactions on Evolutionary Computation*, 20(3):370–385, jun 2016.

[39] Eberhart and Yuhui Shi. Particle Swarm Optimization: Developments, Applications and Resources. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, volume 1, pages 81–86. IEEE.

[40] A. Banks, J. Vincent, and C. Anyakoha. A Review of Particle Swarm Optimization. Part I: Background and Development. *Natural Computing*, 6(4):467–484, oct 2007.

[41] N. K. Jain, U. Nangia, and J. Jain. A Review of Particle Swarm Optimization. *Journal of The Institution of Engineers (India): Series B*, pages 1–5, mar 2018.

[42] . Gülcü and H. Kodaz. A Novel Parallel Multi-Swarm Algorithm Based on Comprehensive Learning Particle Swarm Optimization. *Engineering Applications of Artificial Intelligence*, 45:33–45, 2015.

[43] F. VandenBergh and A. Engelbrecht. A Cooperative Approach to Particle Swarm Optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):225–239, jun 2004.

[44] F. V. den Bergh and A. P. Engelbrecht. Cooperative Learning in Neural Networks Using Particle Swarm Optimizers. *South African Computer Journal*, 2000(26):84–90, 2000.

[45] T. Peram, K. Veeramachaneni, and C. Mohan. Fitness-Distance-Ratio Based Particle Swarm Optimization. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)*, pages 174–181. IEEE.

[46] S. Salehizadeh, P. Yadmellat, and M. Menhaj. Local Optima Avoidable Particle Swarm Optimization. In *2009 IEEE Swarm Intelligence Symposium*, pages 16–21. IEEE, mar 2009.

[47] S. Kessentini and D. Barchiesi. A New Strategy to Improve Particle Swarm Optimization Exploration Ability. In *2010 Second WRI Global Congress on Intelligent Systems*, pages 27–30. IEEE, dec 2010.

[48] J. J. Liang, B. Y. Qu, P. N. Suganthan, and Q. Chen. CEC 2015 Competition on Learning-Based Real-Parameter Single Objective Optimization —-Including 15 Benchmark Functions. 2015.

[49] N. H. A. P.N. Suganthan, Mostafa Z. Ali. CEC 2016 Competition on Learning-Based Real-Parameter Single Objective Optimization —-Including 15 Benchmark Functions. 2016.

[50] M. R. Bonyadi and Z. Michalewicz. A Locally Convergent Rotationally Invariant Particle Swarm Optimization Algorithm. *Swarm Intelligence*, 8(3):159–198, sep 2014.

[51] S. Janson and M. Middendorf. On Trajectories of Particles in PSO. In *2007 IEEE Swarm Intelligence Symposium*, pages 150–155. IEEE, apr 2007.

[52] W. M. Spears, D. Green, and D. F. Spears. Biases in Particle Swarm Optimisation.

[53] R. Storn and K. Price. Differential Evolution a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[54] B. Kazimipour, X. Li, and A. K. Qin. A Review of Population Initialization Techniques for Evolutionary Algorithms. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 2585–2592. IEEE, jul 2014.

[55] V. Feoktistov and S. Janaqi. Generalization of the Strategies in Differential Evolution. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, pages 165–170. IEEE.

[56] H.-Y. Fan and J. Lampinen. A Trigonometric Mutation Operation to Differential Evolution. *Journal of Global Optimization*, 27(1):105–129, 2003.

[57] A. W. Iorio and X. Li. Incorporating Directional Information Within a Differential Evolution Algorithm for Multi-Objective Optimization. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation - GECCO '06*, page 691, New York, New York, USA, 2006. ACM Press.

[58] K. V. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution : A Practical Approach to Global Optimization*. Springer, 2005.

[59] D. Zaharie. A Comparative Analysis of Crossover Variants in Differential Evolution. pages 171–181.

[60] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello. A Comparative Study of Differential Evolution Variants for Global Optimization. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation - GECCO '06*, page 485, New York, New York, USA, 2006. ACM Press.

[61] D. Zaharie. Influence of Crossover on the Behavior of Differential Evolution Algorithms. *Applied Soft Computing*, 9(3):1126–1138, 2009.

[62] R. Storn. Differential Evolution Research Trends and Open Questions. In *Advances in Differential Evolution*, pages 1–31. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[63] R. Gamperle and S. D. Uller. A Parameter Study for Differential Evolution.

[64] R. Storn and K. Price. Differential Evolution a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[65] K. V. Price and J. I. Ronkkonen. Comparing the Uni-Modal Scaling Performance of Global and Local Selection in a Mutation-Only Differential Evolution Algorithm. In *2006 IEEE International Conference on Evolutionary Computation*, pages 2034–2041. IEEE.

[66] C. J. F. T. Braak. A Markov Chain Monte Carlo Version of the Genetic Algorithm Differential Evolution: Easy Bayesian Computing for Real Parameter Spaces. *Statistics and Computing*, 16(3):239–249, sep 2006.

[67] D. Zaharie. Critical Values for the Control Parameters of Differential Evolution Algorithms. 2002.

[68] J. Ronkkonen, S. Kukkonen, and K. V. Price. Real-Parameter Optimization with Differential Evolution. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 506–513. IEEE.

[69] F. Peñuñuri, C. Cab, O. Carvente, M. A. Zambrano-Arjona, and J. A. Tapia. A Study of the Classical Differential Evolution Control Parameters. *Swarm and Evolutionary Computation*, 26:86–96, 2016.

[70] A. Qin and P. Suganthan. Self-Adaptive Differential Evolution Algorithm for Numerical Optimization. In *2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1785–1791. IEEE.

[71] A. P. Piotrowski. Review of Differential Evolution Population Size. *Swarm and Evolutionary Computation*, 32:1–24, 2017.

[72] J. Brest and M. Sepesy Maučec. Population Size Reduction for the Differential Evolution Algorithm. *Applied Intelligence*, 29(3):228–247, dec 2008.

[73] R. Tanabe and A. S. Fukunaga. Improving the Search Performance of SHADE Using Linear Population Size Reduction. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 1658–1665, Tanabe2014, jul 2014. IEEE.

[74] F. Neri and V. Tirronen. Recent Advances in Differential Evolution: A Survey and Experimental Analysis.

[75] S. Das, A. Abraham, U. K. Chakraborty, and A. Konar. Differential Evolution Using a Neighborhood-Based Mutation Operator. *IEEE Transactions on Evolutionary Computation*, 13(3):526–553, jun 2009.

[76] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, dec 2006.

[77] J. Brest, B. Bošković, S. Greiner, V. Žumer, and M. S. Maučec. Performance Comparison of Self-Adaptive and Adaptive Differential Evolution Algorithms. *Soft Computing*, 11(7):617–629, feb 2007.

[78] Jingqiao Zhang and A. Sanderson. JADE: Adaptive Differential Evolution with Optional External Archive. *IEEE Transactions on Evolutionary Computation*, 13(5):945–958, oct 2009.

[79] CEC 2016 Competition CEC'14 Test Suite Part a - Including 30 Benchmark Functions Unimodal Functions. 2016.

[80] J. L. J. Laredo, C. Fernandes, J. J. Merelo, and C. Gagné. Improving Genetic Algorithms Performance Via Deterministic Population Shrinkage. In

Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation - GECCO '09, page 819, New York, New York, USA, 2009. ACM Press.

[81] M. El-Abd. Cooperative Co-Evolution Using LSHADE with Restarts for the CEC15 Benchmarks. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 4810–4814. IEEE, jul 2016.

[82] R. Polakova, J. Tvrdik, and P. Bujok. L-SHADE with Competing Strategies Applied to CEC2015 Learning-Based Test Suite. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 4790–4796. IEEE, jul 2016.

[83] P. Bujok and J. Tvrdik. Enhanced SHADE and Real-World Optimization Problmes.

[84] S. Das, S. S. Mullick, and P. N. Suganthan. Recent Advances in Differential Evolution an Updated Survey. *Swarm and Evolutionary Computation*, 27:1–30, 2016.

[85] S.-M. Guo, J. S.-H. Tsai, C.-C. Yang, and P.-H. Hsu. A Self-Optimization Approach for L-SHADE Incorporated with Eigenvector-Based Crossover and Successful-Parent-Selecting Framework on CEC 2015 Benchmark Set. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 1003–1010. IEEE, may 2015.

[86] Z. Hu, Q. Su, X. Yang, and Z. Xiong. Not Guaranteeing Convergence of Differential Evolution on a Class of Multimodal Functions. *Applied Soft Computing*, 41:479–487, 2016.

[87] R. Tanabe and A. Fukunaga. How Far Are We from an Optimial AdaptiveDE.

[88] W. B. Langdon and R. Poli. Evolving Problems to Learn About Particle Swarm Optimizers and Other Search Algorithms. *IEEE Transactions on Evolutionary Computation*, 11(5):561–578, oct 2007.

[89] J.-y. Yan, Q. Ling, and D.-m. Sun. A Differential Evolution with Simulated Annealing Updating Method. In *2006 International Conference on Machine Learning and Cybernetics*, pages 2103–2106. IEEE, 2006.

[90] Y.-F. Li, Z.-H. Zhan, Y. Lin, and J. Zhang. Comparisons Study of APSO OLPSO and CLPSO on CEC2005 and CEC2014 Test Suits. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 3179–3185. IEEE, may 2015.

[91] A. Ratnaweera, S. Halgamuge, and H. Watson. Self-Organizing Hierarchical Particle Swarm Optimizer with Time-Varying Acceleration Coefficients. *IEEE Transactions on Evolutionary Computation*, 8(3):240–255, jun 2004.

[92] M. R. Bonyadi and Z. Michalewicz. Particle Swarm Optimization for Single Objective Continuous Space Problems: A Review. *Evolutionary Computation*, 25(1):1–54, mar 2017.

[93] A. Banks, J. Vincent, and C. Anyakoha. A Review of Particle Swarm Optimization. Part II: Hybridisation, Combinatorial, Multicriteria and Constrained Optimization, and Indicative Applications. *Natural Computing*, 7(1):109–124, mar 2008.

[94] Y. Bengio, P. Simard, and P. Frasconi. Learning Long-Term Dependencies with Gradient Descent Is Difficult. *IEEE transactions on neural networks*, 5(2):157–66, 1994.

[95] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[96] K. Hornik. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks*, 4(2):251–257, jan 1991.

[97] D. J. Montana and L. Davis. Training Feedforward Neural Networks Using Genetic Algorithms. Technical report.

[98] D. Karaboga, B. Akay, and C. Ozturk. Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks. In *Modeling Decisions for Artificial Intelligence*, pages 318–329. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[99] X.-S. Yang and S. Deb. Cuckoo Search: Recent Advances and Applications. *Neural Computing and Applications*, 24(1):169–174, jan 2014.

[100] E. Valian, S. Mohanna, and S. Tavakoli. Improved Cuckoo Search Algorithm for Feedforward Neural Network Training. *International Journal of Artificial Intelligence & Applications (IJAIA)*, 2(3), 2011.

[101] K. Socha and C. Blum. An Ant Colony Optimization Algorithm for Continuous Optimization: Application to Feed-Forward Neural Network Training. *Neural Computing and Applications*, 16(3):235–247, may 2007.

[102] M. Mandischer. A Comparison of Evolution Strategies and Backpropagation for Neural Network Training. *Neurocomputing*, 42(1-4):87–117, jan 2002.

[103] J. Ilonen, J.-K. Kamarainen, and J. Lampinen. Differential Evolution Training Algorithm for Feed-Forward Neural Networks. *Neural Processing Letters*, 17(1):93–105, 2003.

[104] V. Gudise and G. Venayagamoorthy. Comparison of Particle Swarm Optimization and Backpropagation As Training Algorithms for Neural Networks. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)*, pages 110–117. IEEE.

[105] A. P. Piotrowski, M. J. Napiorkowski, M. Kalinowska, J. J. Napiorkowski, and M. Osuch. Are Evolutionary Algorithms Effective in Calibrating Different Artificial Neural Network Types for Streamwater Temperature Prediction? *Water Resources Management*, 30(3):1217–1237, feb 2016.

[106] A. P. Piotrowski, M. Osuch, M. J. Napiorkowski, P. M. Rowinski, and J. J. Napiorkowski. Comparing Large Number of Metaheuristics for Artificial

Neural Networks Training to Predict Water Temperature in a Natural River. *Computers & Geosciences*, 64:136–151, 2014.

[107] A. P. Piotrowski. Differential Evolution Algorithms Applied to Neural Network Training Suffer from Stagnation. *Applied Soft Computing*, 21:382–406, aug 2014.

[108] R. Mendes, P. Cortez, M. Rocha, and J. Neves. Particle Swarms for Feedforward Neural Network Training. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, pages 1895–1899. IEEE.

[109] F. G. E. Alfassio Grimaldi. PSO As an Effective Learning Algorithm for Neural Network Applications. In *Proceedings. ICCEa 2004. 2004 3rd International Conference on Computational Electromagnetics and Its Applications, 2004.*, pages 557–560. IEEE.

[110] A. Rakitianskaia and A. Engelbrecht. Training High-Dimensional Neural Networks with Cooperative Particle Swarm Optimiser. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 4011–4018. IEEE, jul 2014.

[111] A. Rakitianskaia and A. Engelbrecht. Saturation in PSO Neural Network Training: Good or Evil? In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 125–132. IEEE, may 2015.

[112] P. Wilmott, J. Dewynne, and S. Howison. *Option Pricing : Mathematical Models and Computation*. Oxford Financial Press, Oxford, 1993.

[113] J. Hull. *Options, Futures, and Other Derivatives*. Pearson/Prentice Hall, Upper Saddle River N.J., 6th ed. edition, 2006.

[114] G. Barone-Adesi and R. E. Whaley. Efficient Analytic Approximation of American Option Values. *The Journal of Finance*, 42(2):301–320, jun 1987.

[115] F. A. Longstaff and E. S. Schwartz. Valuing American Options by Simulation: A Simple Least-Squares Approach. *Review of Financial Studies*, 14(1):113–147, jan 2001.

[116] J. B. Cohen, F. Black, and M. Scholes. The Valuation of Option Contracts and a Test of Market Efficiency. *The Journal of Finance*, 27(2):399–417, may 1972.

[117] A. Kemna and A. Vorst. A Pricing Method for Options Based on Average Asset Values. *Journal of Banking & Finance*, 14(1):113–129, mar 1990.

[118] J. Gatheral. *The Volatility Surface : A Practitioner's Guide*. John Wiley & Sons, 2006.

[119] F. D. Rouah. *The Heston Model and Its Extensions in Matlab and C#*. John Wiley & Sons, Inc., Hoboken, NJ, USA, aug 2013.

[120] R. Lord and C. Kahl. Complex Logarithms in Heston like Models. *Mathematical Finance*, 20(4):671–694, sep 2010.

[121] P. Carr. Option Valuation Using the Fast Fourier Transform. Technical report, 1999.

[122] M. Attari. Option Pricing Using Fourier Transforms: A Numerically Efficient Simplification. *SSRN Electronic Journal*, 2004.

[123] H. Albrecher, P. Mayer, W. Schoutens, and J. Tistaert. The Little Heston Trap. Technical report.

[124] C. Kahl and P. Jäckel. Not-So-Complex Logarithms in the Heston Model. Technical report.

[125] A. L. Lewis. *Option Valuation Under Stochastic Volatility II with Mathematica Code*.

[126] S. d. B. Rollin, A. Ferreiro-Castilla, and F. Utzet. A New Look at the Heston Characteristic Function. feb 2009.

[127] Y. Cui, S. del Baño Rollin, and G. Germano. Full and Fast Calibration of the Heston Stochastic Volatility Model. *European Journal of Operational Research*, 263(2):625–638, dec 2017.

[128] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer, New York, 2004.

[129] D. Tavella and C. Randall. *Pricing Financial Instruments : The Finite Difference Method.* John Wiley & Sons, 2000.

[130] D. J. Duffy. *Finite Difference Methods in Financial Engineering : A Partial Differential Equation Approach*. John Wiley, Chichester, England ;;Hoboken, NJ :, 2006.

[131] D. Wolpert and W. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, apr 1997.

[132] B. Xin, J. Chen, J. Zhang, H. Fang, and Z. H. Peng. Hybridizing Differential Evolution and Particle Swarm Optimization to Design Powerful Optimizers: A Review and Taxonomy. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 42(5):744–767, 2012.

[133] T. Hendtlass. A Combined Swarm Differential Evolution Algorithm for Optimization Problems. pages 11–18. Springer, Berlin, Heidelberg, 2001.

[134] S. Sarkar and S. Das. A Hybrid Particle Swarm with Differential Evolution Operator Approach (DEPSO) for Linear Array Synthesis. pages 416–423. Springer, Berlin, Heidelberg, 2010.

[135] B. Niu and L. Li. A Novel PSO-DE-Based Hybrid Algorithm for Global Optimization. In *Advanced Intelligent Computing Theories and Applications. with Aspects of Artificial Intelligence*, pages 156–163. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[136] M. Epitropakis, V. Plagianakos, and M. Vrahatis. Evolving Cognitive and Social Experience in Particle Swarm Optimization Through Differential Evolution: A Hybrid Approach. *Information Sciences*, 216:50–92, 2012.

[137] S. Kannan, S. R. Slochanal, P. Subbaraj, and N. P. Padhy. Application of Particle Swarm Optimization Technique and Its Variants to Generation Expansion Planning Problem. *Electric Power Systems Research*, 70(3):203–210, 2004.

[138] Xing Xu, Yuanxiang Li, Shenlin Fang, Yu Wu, and Feng Wang. A Novel Differential Evolution Scheme Combined with Particle Swarm Intelligence. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 1057–1062. IEEE, jun 2008.

[139] S. Das, A. Konar, and U. K. Chakraborty. Improving Particle Swarm Optimization with Differentially Perturbed Velocity. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation - GECCO '05*, page 177, New York, New York, USA, 2005. ACM Press.

[140] A. Moharam, M. A. El-Hosseini, and H. A. Ali. Design of Optimal PID Controller Using Hybrid Differential Evolution and Particle Swarm Optimization with an Aging Leader and Challengers. *Applied Soft Computing*, 38:727–737, 2016.

[141] A. P. Engelbrecht. Particle Swarm Optimization with Crossover: A Review and Empirical Analysis. *Artificial Intelligence Review*, 45(2):131–165, feb 2016.

[142] X. Yu, J. Cao, H. Shan, L. Zhu, and J. Guo. An Adaptive Hybrid Algorithm Based on Particle Swarm Optimization and Differential Evolution for Global Optimization. *TheScientificWorldJournal*, 2014:215472, 2014.

[143] B. Tang, Z. Zhu, and J. Luo. Hybridizing Particle Swarm Optimization and Differential Evolution for the Mobile Robot Global Path Planning. *International Journal of Advanced Robotic Systems*, 13(3):86, jun 2016.

[144] M. Løvbjerg, T. Kiel Rasmussen, and T. Krink. Hybrid Particle Swarm Optimiser with Breeding and Subpopulations.

[145] M. Settles and T. Soule. Breeding Swarms. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation - GECCO '05*, page 161, New York, New York, USA, 2005. ACM Press.

[146] S. Chen. Particle Swarm Optimization with Pbest Crossover. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–6. IEEE, jun 2012.

[147] T. Zhang, T. Hu, X. Guo, Z. Chen, and Y. Zheng. Solving High Dimensional Bilevel Multiobjective Programming Problem Using a Hybrid Particle Swarm Optimization Algorithm with Crossover Operator. *Knowledge-Based Systems*, 53:13–19, 2013.

[148] Y. Miao, Z. Cui, and J. Zeng. Dynamic Population-Based Particle Swarm Optimization Combined with Crossover Operator. In *2009 Ninth International Conference on Hybrid Intelligent Systems*, pages 399–404. IEEE, 2009.

[149] Y. Dong and H. Yang. A New Approach for Reactive Power/Voltage Optimization Control of Regional Grid. In *2010 Asia-Pacific Power and Energy Engineering Conference*, pages 1–5. IEEE, 2010.

[150] A. P. Engelbrecht. Asynchronous Particle Swarm Optimization with Discrete Crossover. In *2014 IEEE Symposium on Swarm Intelligence*, pages 1–8. IEEE, dec 2014.

[151] K. Deb, D. Joshi, and A. Anand. Real-Coded Evolutionary Algorithms with Parent-Centric Recombination. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 1, pages 61–66. IEEE.

[152] K. Deb and N. Padhye. Development of Efficient Particle Swarm Optimizers by Using Concepts from Evolutionary Algorithms. In *Proceedings of*

the 12th Annual Conference on Genetic and Evolutionary Computation - GECCO '10, page 55, New York, New York, USA, 2010. ACM Press.

[153] K. Deb and N. Padhye. Improving a Particle Swarm Optimization Algorithm Using an Evolutionary Algorithm Framework. 2010.

[154] A. Engelbrecht. Particle Swarm Optimization with Discrete Crossover. In *2013 IEEE Congress on Evolutionary Computation*, pages 2457–2464. IEEE, jun 2013.

[155] B. L. M. Ille, D. E. Goldberg, B. L. Miller, and D. E. Goldb. Genetic Algorithms , Tournament Selection, and the Effects of Noise. pages 193–212, 1995.

[156] I. C. Trelea. The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection. *Information Processing Letters*, 85(6):317–325, mar 2003.

[157] D. Jong and K. Alan. An Analysis of the Behavior of a Class of Genetic Adaptive Systems., 1975.

[158] R. Salomon. Re-Evaluating Genetic Algorithm Performance Under Coordinate Rotation of Benchmark Functions. a Survey of Some Theoretical and Practical Aspects of Genetic Algorithms. *Biosystems*, 39(3):263–278, jan 1996.

[159] T. Bäck and H.-P. Schwefel. An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation*, 1(1):1–23, mar 1993.

[160] M. Gang, Z. Wei, and C. Xiaolin. A Novel Particle Swarm Optimization Algorithm Based on Particle Migration. *Applied Mathematics and Computation*, 218(11):6620–6626, feb 2012.

[161] M. S. Arumugam and M. Rao. On the Improved Performances of the Particle Swarm Optimization Algorithms with Adaptive Parameters, Cross-over

Operators and Root Mean Square (RMS) Variants for Computing Optimal Control of a Class of Hybrid Systems. *Applied Soft Computing*, 8(1):324–336, jan 2008.

[162] I. Montalvo, J. Izquierdo, R. Pérez-García, and M. Herrera. Improved Performance of PSO with Self-Adaptive Parameters for Computing the Optimal Design of Water Supply Systems. *Engineering Applications of Artificial Intelligence*, 23(5):727–735, 2010.

[163] Y. Wang, B. Li, T. Weise, J. Wang, B. Yuan, and Q. Tian. Self-Adaptive Learning Based Particle Swarm Optimization. 181(20):4515–4538, oct 2011.

[164] A. P. Engelbrecht. Heterogeneous Particle Swarm Optimization. pages 191–202. Springer, Berlin, Heidelberg, 2010.

[165] F. V. Nepomuceno and A. P. Engelbrecht. A Self-Adaptive Heterogeneous PSO Inspired by Ants. pages 188–195. Springer, Berlin, Heidelberg, 2012.

[166] T. M. Blackwell and P. J. Bentley. Dynamic Search with Charged Swarms. Technical report.

[167] T. Blackwell and J. Branke. Multiswarms, Exclusion, and Anti-Convergence in Dynamic Environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472, aug 2006.

[168] C. L. Müller and I. F. Sbalzarini. Global Characterization of the CEC 2005 Fitness Landscapes Using Fitness-Distance Analysis. pages 294–303. Springer Berlin Heidelberg, 2011.

[169] C. García-Martínez, P. D. Gutiérrez, D. Molina, M. Lozano, and F. Herrera. Since CEC 2005 Competition on Real-Parameter Optimisation: A Decade of Research, Progress and Comparative Analysis's Weakness. *Soft Computing*, pages 1–11, jan 2017.

[170] Y.-W. Shang and Y.-H. Qiu. A Note on the Extended Rosenbrock Function. *Evolutionary Computation*, 14(1):119–126, mar 2006.

[171] T. P. McDonald, R. W. G. Bucknall, and A. R. Greig. Comparing Trimaran Small Waterplane Area Center Hull (TriSWACH), Monohull, and Trimaran Hullforms: Some Initial Results. *Journal of Ship Production and Design*, 29(4):211–220, nov 2013.

[172] E. C. Tupper. *Introduction to Naval Architecture*. Elsevier, Butterworth Heinemann, Amsterdam ;;Boston :, 2004.

[173] P. Couser, A. Mason, G. Mason, C. R. Smith, and B. R. Von Konsky. Artificial Neural Networks for Hull Resistance Prediction.

[174] A. Carter, E. Muk-Pavic, and T. McDonald. Resistance Prediction Using Artificial Neural Networks for Preliminary Tri-SWACH Design. *Transactions of the Royal Institution of Naval Architects Part A: International Journal of Maritime Engineering , Vol. 156 (2013)*, 2013.

[175] N. Chaggar. *Tri-SWACH Resistance Prediction Using Artificial Neural Networks*. Msc thesis, University College London, 2014.

[176] J. Klag and I. McMahon. *Calm Water Resistance Study of a Novel Trimaran*. Doctoral thesis, Webb Institute, Glen Cove, USA, 2007.

[177] R. A. Royce, A. Mouravieff, and A. Zuzick. Trimaran Resistance Artificial Neural Network. 2011.

[178] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient BackProp. pages 9–48. Springer, Berlin, Heidelberg, 2012.

[179] F. Guillaume and W. Schoutens. Calibration Risk: Illustrating the Impact of Calibration Risk Under the Heston Model. *Review of Derivatives Research*, 15(1):57–79, apr 2012.

[180] BLACK and F. Studies of Stock Market Volatility Changes. *1976 Proceedings of the American Statistical Association Bisiness and Economic Statistics Section*, 1976.

[181] J. Pospíšil, M. Mrázek, and T. Sobotka. *On Optimization Techniques for Calibration of Stochastic Volatility Models StochGrid View Project on Optimization Techniques for Calibration of Stochastic Volatility Models*. 2014.

[182] S. Mikhailov and U. Nögel. Heston's Stochastic Volatility Model Implementation, Calibration and Some Extensions. Technical report.

[183] P. Gauthier and P.-Y. H. Rivaille. Fitting the Smile, Smart Parameters for SABR and Heston. *SSRN Electronic Journal*, oct 2009.

[184] F. Guillaume and W. Schoutens. Use a Reduced Heston or Reduce the Use of Heston? *Wilmott Journal*, 2(4):171–192, aug 2010.

[185] M. Gilli and E. Schumann. Calibrating Option Pricing Models with Heuristics. pages 9–37. Springer Berlin Heidelberg, 2011.

[186] I. Vollrath and J. Wendland. Calibration of Interest Rate and Option Models Using Differential Evolution. *SSRN Electronic Journal*, mar 2009.

[187] S. Haring and R. Hochreiter. Efficient and Robust Calibration of the Heston Option Pricing Model for American Options Using an Improved Cuckoo Search Algorithm. jul 2015.

[188] R. Polakova, J. Tvrdik, and P. Bujok. Evaluating the Performance of L-SHADE with Competing Strategies on CEC2014 Single Parameter-Operator Test Suite. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 1181–1187. IEEE, jul 2016.

[189] F. Kilin. Accelerating the Calibration of Stochastic Volatility Models. 2007.

[190] N. Ivkovic. Measuring Performance of Optimization Algorithms in Evolutionary Computation Representation of Big Data (case Study of Text Mining) View Project. 2016.

[191] F. Le Floc'h. Fourier Integration and Stochastic Volatility Calibration. *SSRN Electronic Journal*, feb 2013.

[192] M. Papadrakakis, V. Papadopoulos, and N. D. Lagaros. Structural Reliability Analyis of Elastic-Plastic Structures Using Neural Networks and Monte Carlo Simulation. *Computer Methods in Applied Mechanics and Engineering*, 136(1-2):145–163, sep 1996.

[193] M. Papadrakakis and N. D. Lagaros. Reliability-Based Structural Optimization Using Neural Networks and Monte Carlo Simulation. *Computer Methods in Applied Mechanics and Engineering*, 191(32):3491–3507, 2002.

[194] J. B. Cardoso, J. R. de Almeida, J. M. Dias, and P. G. Coelho. Structural Reliability Analysis Using Monte Carlo Simulation and Neural Networks. *Advances in Engineering Software*, 39(6):505–513, 2008.

[195] J. M. HUTCHINSON, A. W. LO, and T. POGGIO. A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks. *The Journal of Finance*, 49(3):851–889, jul 1994.

[196] J. Bennell and C. Sutcliffe. Black-Scholes Versus Artificial Neural Networks in Pricing FTSE 100 Options. *Intelligent Systems in Accounting, Finance & Management*, 12(4):243–260, oct 2004.

[197] R. Garcia and R. Gençay. Pricing and Hedging Derivative Securities with Neural Networks and a Homogeneity Hint. *Journal of Econometrics*, 94(1-2):93–115, jan 2000.

[198] R. Gencay and Min Qi. Pricing and Hedging Derivative Securities with Neural Networks: Bayesian Regularization, Early Stopping, and Bagging. *IEEE Transactions on Neural Networks*, 12(4):726–734, jul 2001.

[199] N. Gradojevic, R. Gencay, and D. Kukolj. Option Pricing with Modular Neural Networks. *IEEE Transactions on Neural Networks*, 20(4):626–637, apr 2009.

[200] X. Liang, H. Zhang, J. Xiao, and Y. Chen. Improving Option Price Forecasts with Neural Networks and Support Vector Regressions. *Neurocomputing*, 72(13-15):3055–3065, aug 2009.

[201] M. J. Morelli, G. Montagna, O. Nicrosini, M. Treccani, M. Farina, and P. Amato. Pricing Financial Derivatives with Neural Networks. *Physica A: Statistical Mechanics and its Applications*, 338(1-2):160–165, jul 2004.

[202] M. M. Pires. American Option Pricing Using Computational Intelligence Methods. 2005.

[203] S. P. Das and S. Padhy. A New Hybrid Parametric and Machine Learning Model with Homogeneity Hint for European-Style Index Option Pricing. *Neural Computing and Applications*, 28(12):4061–4077, dec 2017.

[204] M. Kohler, A. Krzy Zak, and N. Todorovic. Pricing of High-Dimensional American Options by Neural Networks. 2006.

[205] D. L. Kelly. Valuing and Hedging American Put Options Using Neural Networks. 1994.

[206] Y. Yang, Y. Zheng, and T. M. Hospedales. Gated Neural Networks for Option Pricing: Rationality by Design. *Thirty-First AAAI Conference on Artificial Intelligence*, feb 2017.

[207] M. Pires and T. Marwala. American Option Pricing Using Multi-Layer Perceptron and Support Vector Machine. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, volume 2, pages 1279–1285. IEEE.

[208] N. Gradojevic. Multi-Criteria Classification for Pricing European Options. *Studies in Nonlinear Dynamics & Econometrics*, 0(0):123–139, jan 2015.

[209] N. Yadav, A. Yadav, and M. Kumar. *An Introduction to Neural Network Methods for Differential Equations*. SpringerBriefs in Applied Sciences and Technology. Springer Netherlands, Dordrecht, 2015.

[210] D. J. Duffy. *Finite Difference Methods in Financial Engineering : A Partial Differential Equation Approach*. John Wiley, Chichester, England ;;Hoboken, NJ :, 2013.

[211] Financial Toolbox - MATLAB.

[212] R. C. Merton. *Continuous-Time Finance*. B. Blackwell, Cambridge Mass., 1990.

[213] L. J. McGuffin, K. Bryson, and D. T. Jones. The PSIPRED Protein Structure Prediction Server. *Bioinformatics*, 16(4):404–405, apr 2000.

[214] S. Palmer, D. Gorse, and E. Muk-Pavic. Neural Networks and Particle Swarm Optimization for Function Approximation in Tri-SWACH Hull Design. In *Proceedings of the 16th International Conference on Engineering Applications of Neural Networks (INNS) - EANN '15*, pages 1–6, New York, New York, USA, 2015. ACM Press.

[215] A. Krogh and J. Vedelsby. Neural Network Ensembles, Cross Validation and Active Learning, 1994.

[216] G. Brown and J. Wyatt. The Use of the Ambiguity Decomposition in Neural Network Ensemble Learning Methods. Technical report, 2003.

[217] J. Mendes-Moreira, C. Soares, A. M. Jorge, and J. F. D. Sousa. Ensemble Approaches for Regression. *ACM Computing Surveys*, 45(1):1–40, nov 2012.

[218] Y. Freund and R. E. Schapire. Experiments with a New Boosting Algorithm. Technical report, Machine Learning: Proceedings of the Thirteenth International Conference, 1996.

[219] L. Breiman. Stacked Regressions. Technical report, 1996.

[220] Y. Liu and X. Yao. Ensemble Learning Via Negative Correlation. *Neural Networks*, 12(10):1399–1404, dec 1999.

[221] Z. A. Dindar and T. Marwala. Option Pricing Using a Committee of Neural Networks and Optimized Networks.

[222] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. SIAM, 1995.

[223] Cboe Global Markets.

[224] L. Breiman. Randomizing Outputs to Increase Prediction Accuracy. *Machine Learning*, 40(3):229–242, 2000.

[225] S. Geman, E. Bienenstock, and R. Doursat. Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, 4(1):1–58, jan 1992.

[226] A. Chandra, H. Chen, and X. Yao. Trade-off Between Diversity and Accuracy in Ensemble Generation. In *Multi-Objective Machine Learning*, pages 429–464. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[227] A. Chandra and X. Yao. DIVACE: Diverse and Accurate Ensemble Learning Algorithm. pages 619–625. Springer, Berlin, Heidelberg, 2004.

[228] Y. Liu and X. Yao. Learning and Evolution by Minimization of Mutual Information. pages 495–504. Springer, Berlin, Heidelberg, 2002.

[229] G. Brown, J. L. Wyatt, and P. Tio. Managing Diversity in Regression Ensembles. *Journal of Machine Learning Research*, 6(Sep):1621–1650, 2005.

[230] R. T. Clemen and R. L. Winkler. Limits for the Precision and Value of Information from Dependent Sources. *Operations Research*, 33(2):427–442, apr 1985.

[231] D. W. Bunn. Statistical Efficiency in the Linear Combination of Forecasts. *International Journal of Forecasting*, 1(2):151–163, jan 1985.

[232] I. T. Jolliffe. A Note on the Use of Principal Components in Regression. *Applied Statistics*, 31(3):300, 1982.

[233] J. B. Burbidge, L. Magee, and A. L. Robb. Alternative Transformations to Handle Extreme Values of the Dependent Variable. *Journal of the American Statistical Association*, 83(401):123–127, mar 1988.

[234] N. Kyurkchiev and S. Markov. On the Hausdorff Distance Between the Heaviside Step Function and Verhulst Logistic Function. *Journal of Mathematical Chemistry*, 54(1):109–119, jan 2016.

[235] M. Fatica and E. Phillips. Pricing American Options with Least Squares Monte Carlo on GPUs. In *Proceedings of the 6th Workshop on High Performance Computational Finance - WHPCF '13*, pages 1–6, New York, New York, USA, 2013. ACM Press.

[236] D. Peaceman and H. Rachford, Jr. The Numerical Solution of Parabolic and Elliptic Differential Equations. *Journal of the Society for Industrial and Applied Mathematics*, 3(1):28–41, 1955.

[237] D. M. Dang, C. Christara, and K. Jackson. A Parallel Implementation on GPUs of ADI Finite Difference Methods for Parabolic PDEs with Applications in Finance. *Available at SSRN 1580057*, 2010.

[238] D. Egloff. GPUs in Financial Computing Part III: ADI Solvers on GPUs with Application to Stochastic Volatility. *Wilmott mag., March*, pages 51–53, 2011.

[239] L. Thomas. Elliptic Problems in Linear Differential Equations over a Network. *Watson Sci. Lab Report. Columbia University, New York*, 1949.

[240] H. S. Stone. An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations. *J. ACM*, 20(1):27–38, January 1973.

[241] R. W. Hockney. A Fast Direct Solution of Poisson's Equation Using Fourier Analysis. *J. ACM*, 12(1):95–113, January 1965.

[242] R.W.Hockney and C.R.Jesshope. *Parallel Computers*. Adam Hilger, 1981.

[243] Y. Zhang, J. Cohen, and J. D. Owens. Fast Tridiagonal Solvers on the GPU. *ACM Sigplan Notices*, 45(5):127–136, 2010.

[244] F. Oliveira, C. Santos, F. Castro, and J. Alves. A Custom Processor for a TDMa Solver in a CFD Application. In R. Woods, K. Compton, C. Bouganis, and P. Diniz, editors, *Reconfigurable Computing: Architectures, Tools and Applications*, volume 4943 of *Lecture Notes in Computer Science*, pages 63–74. Springer Berlin Heidelberg, 2008.

[245] D. Warne, N. A. Kelson, and R. F. Hayward. Solving Tri-Diagonal Linear Systems Using Field Programmable Gate Arrays. 2012.

[246] D. J. Warne, N. A. Kelson, and R. F. Hayward. Comparison of High Level FPGa Hardware Design for Solving Tri-Diagonal Linear Systems. *Procedia Computer Science*, 29:95–101, jan 2014.

[247] G. Chatziparaskevas, B. Kienhuis, and J. Walters. An FPGA-Based Parallel Processor for Black-Scholes Option Pricing Using Finite Differences Schemes. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 709–714, 2012.

[248] I. S. Duff and H. A. van der Vorst. Developments and Trends in the Parallel Solution of Linear Systems. *Parallel Computing*, 25(13 14):1931 – 1970, 1999.

[249] Q. Jin, T. Becker, W. Luk, and D. Thomas. Optimising Explicit Finite Difference Option Pricing for Dynamic Constant Reconfiguration. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference On*, pages 165–172, 2012.

[250] X. Tian and K. Benkrid. High-Performance Quasi-Monte Carlo Financial Simulation: FPGa Vs. GPP Vs. GPU. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 3(4):26, 2010.

[251] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, 1996.

[252] C. Barnes, B. Tran, and S. Leung. On the Statistics of Fixed-Point Roundoff Error. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 33(3):595–606, 1985.

[253] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. 2005.