PARALLEL SOLUTION OF DIFFUSION EQUATIONS USING LAPLACE TRANSFORM METHODS WITH PARTICULAR REFERENCE TO BLACK-SCHOLES MODELS OF FINANCIAL OPTIONS

A. M. FITZHARRIS

PH. D

2014

Parallel Solution of Diffusion Equations Using Laplace Transform Methods With Particular Reference to Black-Scholes Models of Financial Options

Andrew Fitzharris

A thesis submitted in partial fulfilment of the requirements of the University of Hertfordshire for the degree of

Doctor of Philosophy

The programme of research was carried out in the
School of Physics, Astronomy and Mathematics
University of Hertfordshire

February 2014

Acknowledgements

In July 1968 my maths teacher, Mrs Catherall, wrote in my end of year school report, "*If Andrew spent as much time working as he does looking out of the window at the other boys playing football, he would be quite good at maths*". Needless to say my parents were not impressed[1]. However, this comment was a fairly accurate reflection of my interest in academic subjects at that time. I remained disinterested until September 1975 when, as a final year engineering apprentice on day-release at Letchworth College of Technology, I was fortunate to have Dr Brian Crawford for maths. He was unlike any maths teacher I had been taught by before. He was young, drove a Jensen sports car and was a fan of The Who and Bruce Springsteen. However, more importantly he taught maths in a way that inspired me and all of the other apprentices in the class. His method was to teach maths in an applied context *i.e.* to show us real-world applications for the maths we were studying. For the first time we saw that topics like algebra, calculus, trigonometry and geometry were not intellectual pursuits as many of us believed but were essential tools for solving problems in engineering and many other areas. His inspirational teaching changed the direction of my life. The following September, encouraged by my late wife Yvonne, I enrolled at Stevenage College to take my 'O'-level in maths and to use an old cliché, "*and the rest is history*". How surprised would Mrs Catherall be to discover that I have had a 30-year career teaching maths at a university, that I have lecturered at the Royal Institution and that I have obtained my Ph.D.

Coming up-to-date, I would also like to thank :

- my supervisors, Dr. Steve Kane and Professor Choi-Hong Lai for their guidance, support and encouragement during this investigation
- Professor Alan Davies for his helpful input and advice
- Professor Bruce Christianson, Dr Abdel Salhi and Dr David Sayers for the friendly and professional way in which they conducted my viva voce examination

Finally, I would like to thank my daughters Laura and Robyn who are the best thing that has ever happened to me and are a constant source of inspiration.

This dissertation is dedicated to Emily Alice Johnson, 1905-1974. With me always ...

[1] Fortunately for me I excelled in P.E. and played several sports at County level.

Abstract

Diffusion equations arise in areas such as fluid mechanics, cellular biology, weather forecasting, electronics, mechanical engineering, atomic physics, environmental science, medicine, *etc.* This dissertation considers equations of this type that arise in mathematical finance.

For over 40 years traders in financial markets around the world have used Black-Scholes equations for valuing financial options. These equations need to be solved quickly and accurately so that the traders can make prompt and accurate investment decisions. One way to do this is to use parallel numerical algorithms. This dissertation develops and evaluates algorithms of this kind that are based on the Laplace transform, numerical inversion algorithms and finite difference methods. Laplace transform-based algorithms have faced a legitimate criticism that they are ill-posed *i.e.* prone to instability. We demonstrate with reference to the Black-Scholes equation, contrary to the received wisdom, that the use of the Laplace transform may be used to produce reasonably accurate solutions (*i.e.* to two decimal places), in a fast and reliable manner when used in conjunction with standard PDE techniques.

To set the scene for the investigations that follow, the reader is introduced to financial options, option pricing and the one-dimensional and two-dimensional linear and nonlinear Black-Scholes equations. This is followed by a description of the Laplace transform method and in particular, four widely used numerical algorithms that can be used for finding inverse Laplace transform values. Chapter 4 describes methodology used in the investigations completed *i.e.* the programming environment used, the measures used to evaluate the performance of the numerical algorithms, the method of data collection used, issues in the design of parallel programs and the parameter values used.

To demonstrate the potential of the Laplace transform based approach, Chapter 5 uses existing procedures of this kind to solve the one-dimensional, linear Black-Scholes equation. Chapters 6, 7, 8, and 9 then develop and evaluate new Laplace transform-finite difference algorithms for solving one-dimensional and two-dimensional, linear and nonlinear Black-Scholes equations. They also determine the optimal parameter values to use in each case *i.e.* the parameter values that produce the fastest and most accurate solutions. Chapters 7 and 9 also develop new, iterative Monte Carlo algorithms for calculating the reference solutions needed to determine the accuracy of the LTFD solutions.

Chapter 10 identifies the general patterns of behaviour observed within the LTFD solutions and explains them. The dissertation then concludes by explaining how this programme of work can be extended. The investigations completed make significant contributions to knowledge. These are summarised at the end of the chapters in which they occur. Perhaps the most important of these is the development of fast and accurate numerical algorithms that can be used for solving diffusion equations in a variety of application areas.

Contents

Appendices

List of Figures

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Chapter 9

List of Tables

Chapter 4

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Chapter 9

Appendix B

Appendix C

Appendix D

Appendix E

Appendix F

Chapter 1

Introduction

1.0 Introduction

The opening chapter of this dissertation provides the background information for this research programme. It explains that diffusion equations arise in a variety of application areas and gives some specific examples. The application area considered in this dissertation is mathematical finance. In the financial sector there is a need to find fast and accurate solutions of option pricing problems. One way to find these solutions is to use parallel numerical algorithms based on the Laplace transform. This chapter gives the aims of this research programme and states my thesis. It concludes by outlining the structure of this dissertation.

1.1 Background

Diffusion equations arise in many areas of science, engineering and commerce. For example, they are used to model :

- the diffusion of one material into another *e.g.* smoke particles into air
- the flow of heat from one part of an object to another
- chemical reactions
- electrical activity in the membranes of living organisms
- the dispersion of populations
- pursuit and evasion in predator-prey systems.

In financial markets around the world traders use mathematical models to value financial options. In order to make prompt and accurate investment decisions traders need :

- a range of solutions so that a variety of financial scenarios can be considered
- fast and accurate numerical algorithms for solving the underlying equations.

An efficient way to obtain these solutions is to use an algorithm in which the time domain is decomposed so that parallel computing methods can be used. One way to do this is to use an algorithm based on the Laplace transform, (Crann *et al.* 2007). However, a legitimate

criticism of these algorithms is that Laplace transform inversion is ill-posed[1] (Epstein and Schotland 2008). As a result, Laplace transform based algorithms have become unfashionable and most recent research into time domain decomposition (*i.e.* parallel) algorithms for solving option pricing problems has focused on the use of Fourier and Fast Fourier transforms. See Leentvaar *et al.* (2008) and Barua *et al.* (2004). However, Laplace transforms have a number of advantages over Fourier and Fast Fourier transforms. Beerends *et al.* (2003) state that Laplace transforms :

- exist for a wider range of inputs and are more generally applicable
- are usually easier to invert
- are better suited for solving *Causal LTC-systems*[2]. Equations modelling financial options fall into this category.
- are more computationally efficient. When using Laplace transforms the initial conditions can be introduced into the solution at an early stage and this reduces the number of subsequent calculations, especially for high-order equations.

Furthermore, significant progress has been made into developing accurate numerical algorithms for inverting Laplace transforms, (Kuhlman 2012). Hence, Laplace transform based algorithms are due for reconsideration.

In this research programme the accuracy of the solutions obtained using the Laplace transform based algorithms is validated by comparing them with solutions obtained using independent methods. This is either the analytical solution of the equation or a solution obtained using a Monte Carlo algorithm. It is left for future work to validate the accuracy of the solutions obtained using corresponding algorithms based on the use of Fourier and Fast Fourier transforms. See 11.2.

In addition to Fourier, Fast Fourier and Laplace transform based methods many other time domain decomposition methods are available *e.g.* mortar methods, balancing domain decomposition methods, Schwarz methods, the Schur complement method and FETI-DP[3] methods. Detailed descriptions of these methods can be found in Quarteroni and Valli (1999).

---

1 Hadamard (1923) states that a problem is *well-posed* if (1) a solution to the problem exists, (2) the solution is unique and (3) the solution depends continuously on the problem data so that small changes in the data produce small changes in the solution. It follows from this definition that a problem is *ill-posed* if any of these conditions do not hold.

2 Linear, Time-invariant and Continuous. A system of this kind is said to be Causal if it remains at rest until time $t \geq 0$.

3 Finite Element Tearing and Interconnect - Dual Primal.

1.2 Aims

The aims of this research programme are to :

- develop and evaluate sequential finite difference algorithms and sequential and parallel Laplace transform based algorithms for solving one-dimensional and two-dimensional, linear and nonlinear diffusion equations. In particular, Black-Scholes equations of these types
- determine the optimal numerical inversion algorithms to use in the Laplace transform based algorithms for solving these equations and the optimal parameter values to use in each case
- provide the evidence to support my thesis *i.e.* to show that the Laplace transform based algorithms produce fast and accurate solutions of the Black-Scholes equations mentioned above.

It is not an aim of this research programme to examine or to provide detailed descriptions of financial markets, products or trading strategies. Interested readers should consult a reference such as Wilmot *et al.* (1999)

1.3 Dissertation Structure

Chapters 2, 3 and 4 of this dissertation provide a review of the supporting literature. Chapter 2 introduces financial options, option pricing and the Black-Scholes model and equations. Chapter 3 describes the Laplace transform method for solving differential equations. It considers the advantages and disadvantages of this method and describes four commonly used numerical algorithms that can be used for finding inverse Laplace transform values. Chapter 4 explains the methodology used. It describes the programming environment used, the measures used to evaluate the performance of the programs and algorithms and the method of data collection used. It then considers issues in the design of parallel programs and gives the parameter values used in the numerical inversion algorithms.

Chapters 5, 6, 7, 8 and 9 provide the evidence to support my thesis. Chapter 5 describes initial investigations into Laplace transform solutions of the one-dimensional, linear Black-Scholes equation. Firstly, the equation is solved using the Laplace transform of its analytical solution. Secondly, the equation is converted into its ordinary differential equation (ODE) boundary-value problem (BVP) form and then solved using Laplace transform and finite difference methods. Chapter 6, 7, 8 and 9 develop and evaluate sequential finite

difference algorithms and sequential and parallel Laplace transform and finite difference algorithms for solving one-dimensional and two-dimensional, linear and nonlinear Black-Scholes equations. These chapters also determine the optimal numerical inversion algorithms to use with the Laplace transform based methods and the optimal parameter values to use in these algorithms. In the nonlinear cases Monte Carlo algorithms are also developed for calculating the reference solutions required for assessing the accuracy of the finite difference and the Laplace transform-finite difference solutions obtained.

The final parts of this dissertation are Chapter 10 and Chapter 11. Chapter 10 identifies and explains the general patterns of behaviour observed when solving Black-Scholes equations. Chapter 11 gives the overall contribution to knowledge and describes how this research programme can be extended.

1.4 Chapter Summary

This chapter has set the scene for this research programme. The next stage will be to familiarise the reader with financial markets and in particular, the Black-Scholes equations that are used widely for calculating option prices.

Chapter 2

The Black-Scholes Model and Equations

2.0 Introduction

Financial markets have existed for hundreds of years and financial options have been traded for even longer. This chapter gives the historical background of this field and describes options of the types traded today. It continues by introducing the Black-Scholes model and by developing the one-dimensional and multi-dimensional, linear Black-Scholes equations. The linear equations assume that financial options can be traded without cost. In reality, each time a quantity of the underlying asset is bought or sold a transaction cost is incurred. When transaction costs are taken into consideration Black-Scholes equations become nonlinear *i.e.* contain non-constant volatilities. The most commonly used volatility models are described and this chapter concludes by describing Black-Scholes equations in which the volatility is modelled by a stochastic process.

2.1 Historical Background

Financial options have been traded for thousands of years. One of the earliest records dates back to the time of the Ancient Greeks. At this time it was common for someone wishing to invest in a commodity like olives, to pay the producer for the right to purchase a particular quantity, for an agreed price at a fixed time in the future, (Pliska 2010). Options of this type are still traded today. The first fully functional commodities exchange was established by the Japanese in the 17th Century initially to enable the elite class, the Samurai, to earn money from the rice trade, (Abraham 2010). During these times the *fair price* of an option was usually determined by negotiation between the buyer and the seller. The first attempt in recent times to use mathematical methods to determine the value of an option dates back to 1900. In his Ph.D dissertation the French mathematician Louis Bachelier derived a closed formula[4] for calculating option prices, (Bachelier 1900). His formula was based upon similar assumptions to modern option pricing formulae and used the same dependent variables, (Benhamou 2008). However, his formula had two major faults. Firstly, it ignored discounting *i.e.* it did not give the present value of an option and secondly, it allowed option prices to be negative. As a result it was not used widely.

[4] A closed formula is one that is expressed in terms of a finite number of "well-known" functions.

Over the next 70 years many other option pricing formulae were developed, many based on the Bachelier approach. A history of the development of these formulae can be found in Benhamou (2008). Unfortunately, all of these formulae had disadvantages. Some were complicated *i.e.* contained numerous parameters, many of which were difficult to estimate. Others did not give investors control over, or information about, the degree of risk involved in purchasing an option. A number failed to give a *universal price* for an option. The price given was related to the degree of risk the investor was prepared to accept. As a result, none of these formulae gained widespread popularity.

The major breakthrough in option pricing came in 1973 when Fischer Black and Myron Scholes published their now famous paper[5]. In this, they introduced the Black-Scholes partial differential equation (PDE). Its solution is a closed formula for pricing options, similar to the one developed by Bachelier in 1900, but without the earlier formulas' disadvantages. The advance made by Black and Scholes was to realise that the expected return of the option price should be the risk-free interest rate and that by holding a particular quantity of stock (called the delta), all risk in the investment could be eliminated, (Benhamou 2008). The Black-Scholes formula also has the following advantages over previous option pricing formulae :

- it is a relatively easy formula to evaluate
- it takes into account the five most important factors in option pricing. These are :
    - the current price of the asset on which the option is based
    - the price at which the option holder has the right to purchase the asset
    - the amount of time left until the contract expires
    - the variability[6] of the asset price so that the investor can predict the value of the asset on the expiry date
    - the current rates on offer for risk-free investments like Government bonds, (Benhamou 2008)
- it allows investors to manage their risk
- it gives a universal option price *i.e.* the same price for all investors, whatever their degree of risk aversion.

---

5 See Black and Scholes (1973).

6 This variability is called the volatility of the option price. This term is defined in more detail in 2.3.

Shortly after its development an empirical study showed that the Black-Scholes formula predicted option prices that were very close to the actual price at which they were being traded. As a result the formula gained popularity and is now used in financial markets all over the world[7].

Fischer Black died in 1995. However, in 1997, Black, Scholes and Robert Merton, who provided a detailed mathematical understanding of the options pricing model, were awarded the Nobel Prize for Economics.

2.2 European Call Options

The simplest financial option is the European call option. This is a financial contract in which the holder may purchase a particular asset, called the underlying asset, for an agreed price, called the exercise price or the strike price, at some time in the future, called the expiry date. The term *may* here indicates that the holder has the right to purchase the underlying asset but does not have an obligation to do so. However, the other party to the contract, the seller, has an obligation to sell the asset if the holder chooses to buy it. The term *European* here indicates that the holder may not purchase the asset *i.e.* exercise the option, until the expiry date. Options in which the holder may exercise the option before the expiry date are called American options.

The holder of a European call option will purchase the underlying asset if it is financially sensible to do so. That is, if the value of this asset is greater than the exercise price at the expiry date. However, if value of the underlying asset is less than the exercise price at the expiry date then the holder will not purchase the asset. This right, without obligation, combined with the need to compensate the seller for the obligation they have assumed, means that an option of this type has a value[8]. The seller must determine this value under the current market conditions so that the option can be sold for the appropriate price.

The most commonly used mathematical model for determining the value of a European call option is the Black-Scholes equation.

---

[7] Black, Scholes and Merton worked as Professors at MIT. Here, they helped to train the next generation of traders, many of whom would go on to work on the New York Stock Exchange on Wall Street. This helped to increase the popularity of the Black-Scholes model further.

[8] The value of an option is the non-refundable premium the holder must pay the seller in order to have the right to buy the underlying asset at the expiry date. It does not include the exercise price of the option.

2.3 The Underlying Assumptions of the Black-Scholes Model

Black and Scholes (1973) state that the underlying assumptions of their model are :

1. The value of the underlying asset follows a lognormal random walk[9]. This behaviour is described by the stochastic differential equation :

$$dS = \mu S dt + \sigma S dW(t) \quad \text{---- (1)}$$

Here :

μSdt	is the deterministic component of the equation
μ	is called the drift. This is a measure of the average rate of growth of the underlying asset value
S	is the current value of the underlying asset
t	is time. dt is a small change in time
$\sigma SdW(t)$	is the stochastic component of the equation
σ	is called the volatility. This is a measure of the variation in the price of the underlying asset over time[10]
$W(t)$	is called a Wiener process or Brownian motion. This is a continuous sequence of random values drawn from a normal distribution with a mean of zero and a variance of dt. $dW(t)$ is a small change in $W(t)$.

2. The volatility σ and the risk-free interest rate *r* are constants or known functions of time
3. There are no transaction costs associated with hedging[11] the portfolio
4. The underlying asset pays no dividends during the life of the option
5. There are no arbitrage opportunities *i.e.* it is not possible for an investor to make more money by investing the value of the underlying asset in a risk-free, no cost investment such as a high interest savings account

---

[9] The log of the asset price follows a normal distribution.

[10] A commonly used measure of volatility is the annualised standard deviation of the daily logarithmic returns *i.e.* $\sigma = S/\sqrt{P}$ where *S* is standard deviation of the daily logarithmic returns and *P* is the time period of the returns. Since there are 252 trading days in a year it is usual to assume that $P = 1/252$. The daily logarithmic return is defined as $r_{\log} = \ln(V_n/V_c)$ where V_c is the current value of the option and V_n is the next value of the option.

[11] A hedge is an investment position used to offset potential losses in a companion investment.

6. Trading in the underlying asset can take place continuously
7. Short selling is permissible and the underlying asset is divisible. Short selling is selling a stock or asset that the seller does not own in the hope that the value of that stock or asset will go down.

2.4 The One-Dimensional, Linear Black-Scholes Equation

2.4.1 Itô's Lemma

Let $V(S,t)$ be a smooth function of a stochastic variable *S* and a deterministic variable *t*. Suppose that *S* and *t* vary by small amounts *dS* and *dt* respectively. Then, by Taylor series, neglecting the high-order terms, the corresponding small change in *V* is given by :

$$dV = \frac{\partial V}{\partial S} dS + \frac{\partial V}{\partial t} dt + \frac{1}{2} \frac{\partial^2 V}{\partial S^2} dS^2 \quad \text{---- (2)}$$

From (1) it follows that :

$$dS^2 = \left(\mu S dt + \sigma S dW(t)\right)^2$$

$$i.e. \quad dS^2 = \mu^2 S^2 dt^2 + 2\mu\sigma S^2 dt dW(t) + \sigma^2 S^2 dW(t)^2$$

The changes in *S* and *t* occur over small intervals of time. Shreve (2010) shows that as $dt \to 0$ in a Brownian motion, $dt^2 \to 0$ and $dW(t)^2 \to dt$. Hence :

$$dS^2 = \sigma^2 S^2 dt \quad \text{---- (3)}$$

Replacing *dS* with its value from (1) and dS^2 with its value from (3) the Taylor expansion (2) becomes :

$$dV = \frac{\partial V}{\partial S}\left(\mu S dt + \sigma S dW(t)\right) + \frac{\partial V}{\partial t} dt + \frac{1}{2} \frac{\partial^2 V}{\partial S^2} \sigma^2 S^2 dt$$

$$i.e. \quad dV = \left(\mu S \frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \right) dt + \sigma S \frac{\partial V}{\partial S} dW(t)$$

This expression is called Itô's Lemma. It is the stochastic calculus equivalent of the total differential in ordinary calculus.

2.4.2 Development of the Equation

Consider a portfolio consisting of an option to short sell a number $-\Delta$ of the underlying asset. The value π of the portfolio is :

$$\pi = V - \Delta S$$

where *V* is the value of the option and *S* is the current value of the underlying asset. The change in the value of this portfolio over a small interval in time *dt* is :

$$d\pi = dV - \Delta dS$$

Replacing *dV* with its value from Itô's Lemma and *dS* with its value from (1) :

$$d\pi = \left(\mu S \frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \right) dt + \sigma S \frac{\partial V}{\partial S} dW(t) - \Delta \left(\mu S dt + \sigma S dW(t) \right)$$

$$i.e. \quad d\pi = \left(\mu S \frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - \Delta \mu S \right) dt + \left(\sigma S \frac{\partial V}{\partial S} - \Delta \sigma S \right) dW(t)$$

The stochastic component in this expression can be eliminated by choosing $\Delta = \frac{\partial V}{\partial S}$. This is called *delta hedging*. In this case, the change in the value of the portfolio becomes :

$$d\pi = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \right) dt \quad \text{---- (4)}$$

Suppose that the value of the portfolio π is invested in a risk-free, no cost investment that produces a return of *r*% per annum. Then, the change in the value of the investment over a small interval in time *dt* is :

$$r\pi dt \quad \text{---- (5)}$$

To prevent arbitrage opportunities the change in the value of the portfolio and the change in the value of the investment must be the same. Equating (4) and (5) :

$$r\pi dt = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \right) dt$$

Since $\pi = V - \Delta S$ and $\Delta = \frac{\partial V}{\partial S}$, $\pi = V - \frac{\partial V}{\partial S} S$. Hence :

$$r\left(V - \frac{\partial V}{\partial S} S \right) dt = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \right) dt$$

$$i.e. \quad rV - rS \frac{\partial V}{\partial S} = \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}$$

$$i.e. \quad \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0 \quad \text{---- (6)}$$

The Black-Scholes equation (6) is a backward parabolic PDE[12], (Smith 2004). Its solution gives the value of the option at the expiry time *i.e.* the future value of the option. To be able to solve this equation uniquely it is necessary to know the final condition and the boundary conditions. For a European call option these conditions are as follows. The final condition is given by the payoff at the expiry time,

$$V(S,T) = \max(S-E,0) \quad \text{---- (7)}$$

where T is the expiry time and E is the exercise price. If $S = 0$ at the expiry time then the value of the option is zero. Hence, the first boundary condition is :

$$V(0,t) = 0 \quad \text{---- (8)}$$

As the current value of the underlying asset increases it becomes increasingly likely that the option will be exercised. In this case the value of the option will tend to the value of the asset. Hence, the second boundary condition is :

$$V(S,t) \approx S \text{ as } S \to \infty \quad \text{---- (9)}$$

The seller will wish to know the current value of the option. To be able to determine this value using the Black-Scholes model, equation (6) must be rewritten as a forward parabolic PDE. This can be done by making the change of variable $\tau = T - t$, where τ is the time to expiry. Using this transformation, the equation becomes :

$$-\frac{\partial V}{\partial \tau} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \quad \text{---- (10)}$$

The final condition (7) is transformed into the initial condition :

$$V(S,0) = \max(S-E,0) \quad \text{---- (11)}$$

The boundary conditions (8) and (9) change to :

$$V(0,\tau) = 0 \quad \text{---- (12)} \quad \text{and} \quad V(S,\tau) \approx S \text{ as } S \to \infty \quad \text{---- (13)}$$

2.4.3 The Analytical Solution

Using a technique called Similarity Reduction, Black and Scholes (1973) and Wilmot *et al.* (1999) show that the analytical solution of equation (10) is :

[12] A backward parabolic PDE is one in which the time derivative and the second spatial derivative have the same sign when they are written on the same side of the equation.

$$V(S,\tau) = SN(d_1) - Ee^{-r\tau}N(d_2) \quad \text{---- (14)}$$

where $N(x)$ is the cumulative distribution function for a standardised normal random variable. This is defined as :

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}y^2} dy \qquad \text{or} \qquad N(x) = \frac{1}{2} erfc\left(-\frac{x}{\sqrt{2}}\right)$$

The terms d_1 and d_2 are defined as :

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

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

2.5 European Put Options

The Black-Scholes equation (10) can also be used for valuing a European put option. This is a financial contract in which the holder may *sell* a particular asset for an agreed price at some time in the future. However, when used to value an option of this type the equation has a different initial condition, different boundary conditions and a different analytical solution. Since the value of a European put option can always be calculated using the call-put parity equation *i.e.*

$$C + \frac{E}{(1+r)^t} = S_0 + P$$

where *C* is the call premium, *P* is the put premium and S_0 is the initial value of the underlying asset (Wilmot 2000), options of this type are not considered during this research programme.

2.6 The Multi-Dimensional, Linear Black-Scholes Equation

Financial options can be written on more than one underlying asset. Options of this type are called *basket options* or *rainbow options*, Geske (1979). The value of an option of this type can be found using the multi-dimensional version of the Black-Scholes equation (10). This is developed in the same way as the one-dimensional equation.

2.6.1 Itô's Lemma in Higher Dimensions

Let $V(S_1, S_2, \ldots, S_d, t)$ be a smooth function of *d* stochastic variables $S_1, S_2, \ldots, S_d$ and a deterministic variable *t*. Suppose that $S_1, S_2, \ldots, S_d$ and *t* vary by small amounts $dS_1, dS_2, \ldots, dS_d$ and dt respectively. Then, by Taylor series, neglecting the high-order terms as before, the corresponding small change in *V* is given by :

$$dV = \sum_{i=1}^{d} \frac{\partial V}{\partial S_i} dS_i + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \frac{\partial^2 V}{\partial S_i \partial S_j} dS_i dS_j + \frac{\partial V}{\partial t} dt \quad \text{---- (15)}$$

Extending assumption 1. of the Black-Scholes model, the change in the value of the *i*th stochastic variable S_i is given by :

$$dS_i = \mu_i S_i dt + \sigma_i S_i dW_i(t) \quad \text{---- (16)}$$

where μ_i and σ_i are the drift and the volatility of the *i*th asset respectively and $dW_i(t)$ is a Brownian motion.

Note
In the multi-dimensional case $dW_i(t)$ and $dW_j(t)$, $i \neq j$, are correlated *i.e.*

$$\mathrm{E}\left(dW_i(t).dW_j(t)\right) = \rho_{ij} dt$$

where ρ_{ij} is the correlation coefficient between the *i*th and *j*th random walks, $\rho_{ij} = \rho_{ji}$ and $\rho_{ii} = 1$. The symmetric, positive definite or positive semi-definite matrix[13] Σ that has ρ_{ij} in the *i*th row and the *j*th column is called the correlation matrix. For example, for an option written on five underlying assets, the correlation matrix is :

$$\Sigma = \begin{bmatrix} 1 & \rho_{12} & \rho_{13} & \rho_{14} & \rho_{15} \\ \rho_{21} & 1 & \rho_{23} & \rho_{24} & \rho_{25} \\ \rho_{31} & \rho_{32} & 1 & \rho_{34} & \rho_{35} \\ \rho_{41} & \rho_{42} & \rho_{43} & 1 & \rho_{45} \\ \rho_{51} & \rho_{52} & \rho_{53} & \rho_{54} & 1 \end{bmatrix}$$

From (16) it follows that :

$$dS_i dS_j = \left(\mu_i S_i dt + \sigma_i S_i dW_i(t)\right)\left(\mu_j S_j dt + \sigma_j S_j dW_j(t)\right)$$

---

[13] Let *M* be an *nxn* real, symmetric matrix and $\underline{y}$ be a column vector. Then *M* is positive definite if $\underline{y}^T M \underline{y} > 0 \ \forall \underline{y} \neq \underline{0}$ and positive semi-definite if $\underline{y}^T M \underline{y} \geq 0 \ \forall \underline{y} \neq \underline{0}$.

i.e. $dS_i dS_j = \mu_i \mu_j S_i S_j dt^2 + \mu_i \sigma_j S_i S_j dt dW_j(t) + \mu_j \sigma_i S_i S_j dt dW_i(t) + \sigma_i \sigma_j S_i S_j dW_i(t) dW_j(t)$

In a Brownian motion $dt^2 \to 0$ as $dt \to 0$, (Shreve 2010). Hence :

$$dS_i dS_j = \sigma_i \sigma_j S_i S_j dW_i(t) dW_j(t)$$

Since $\mathrm{E}\left(dW_i(t).dW_j(t)\right) = \rho_{ij} dt$ it follows that :

$$dS_i dS_j = \sigma_i \sigma_j \rho_{ij} S_i S_j dt \quad \text{---- (17)}$$

Substituting (17) into (15), the multi-dimensional version of Itô's Lemma becomes :

$$dV = \sum_{i=1}^{d} \frac{\partial V}{\partial S_i} dS_i + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_i \sigma_j \rho_{ij} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} dt + \frac{\partial V}{\partial t} dt$$

i.e. $$dV = \left(\frac{\partial V}{\partial t} + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_i \sigma_j \rho_{ij} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} \right) dt + \sum_{i=1}^{d} \frac{\partial V}{\partial S_i} dS_i$$

2.6.2 Development of the Equation

Consider a portfolio consisting of an option to short sell a number $-\Delta_i$ of the *i*th underlying asset. The value π of the portfolio is :

$$\pi = V - \sum_{i=1}^{d} \Delta_i S_i$$

where *V* is the value of the option and S_i is the current value of the *i*th underlying asset. The change in the value of this portfolio over a small interval in time *dt* is :

$$d\pi = dV - \sum_{i=1}^{d} \Delta_i dS_i$$

Replacing *dV* with its value from Itô's Lemma :

$$d\pi = \left(\frac{\partial V}{\partial t} + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_i \sigma_j \rho_{ij} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} \right) dt + \sum_{i=1}^{d} \frac{\partial V}{\partial S_i} dS_i - \sum_{i=1}^{d} \Delta_i dS_i$$

The stochastic component in this expression can be eliminated by delta hedging *i.e.* by choosing $\Delta_i = \frac{\partial V}{\partial S_i}$. The change in the value of the portfolio then becomes :

$$d\pi = \left(\frac{\partial V}{\partial t} + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_i \sigma_j \rho_{ij} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} \right) dt \quad \text{---- (18)}$$

Suppose that the value of the portfolio π is invested in a risk-free, no cost investment that produces a return of *r*% per annum. Then, as before, the change in the value of the investment over a small interval in time *dt* is :

$$r\pi dt \quad \text{---- (5)}$$

Using the arbitrage argument from 2.4.2, the change in the value of the portfolio and the change in the value of the investment must be the same. Equating (18) and (5) :

$$r\pi dt = \left(\frac{\partial V}{\partial t} + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_i \sigma_j \rho_{ij} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} \right) dt$$

$$\textit{i.e.} \quad r\pi = \frac{\partial V}{\partial t} + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_i \sigma_j \rho_{ij} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} \quad \text{----(19)}$$

Since $\pi = V - \sum_{i=1}^{d} \Delta_i S_i$ and $\Delta_i = \frac{\partial V}{\partial S_i}$:

$$\pi = V - \sum_{i=1}^{d} \frac{\partial V}{\partial S_i} S_i$$

Substituting this into (19) :

$$r\left(V - \sum_{i=1}^{d} \frac{\partial V}{\partial S_i} S_i \right) = \frac{\partial V}{\partial t} + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_i \sigma_j \rho_{ij} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j}$$

$$\textit{i.e.} \quad \frac{\partial V}{\partial t} + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_i \sigma_j \rho_{ij} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} + r \sum_{i=1}^{d} \frac{\partial V}{\partial S_i} S_i - rV = 0$$

This equation is the multi-dimensional version of the Black-Scholes equation (6). In the case where $d = 2$, this equation becomes[14] :

$$\frac{\partial V}{\partial t} + \frac{1}{2} \sigma_1^2 S_1^2 \frac{\partial^2 V}{\partial S_1^2} + \frac{1}{2} \sigma_2^2 S_2^2 \frac{\partial^2 V}{\partial S_2^2} + \sigma_1 \sigma_2 \rho_{12} S_1 S_2 \frac{\partial^2 V}{\partial S_1 \partial S_2} + r S_1 \frac{\partial V}{\partial S_1} + r S_2 \frac{\partial V}{\partial S_2} - rV = 0 \quad \text{---- (20)}$$

[14] Since *V* is a smooth function, the mixed partial derivatives are equal.

For a particular value of time *t*, the solution domain for equation (20) can be visualised as :

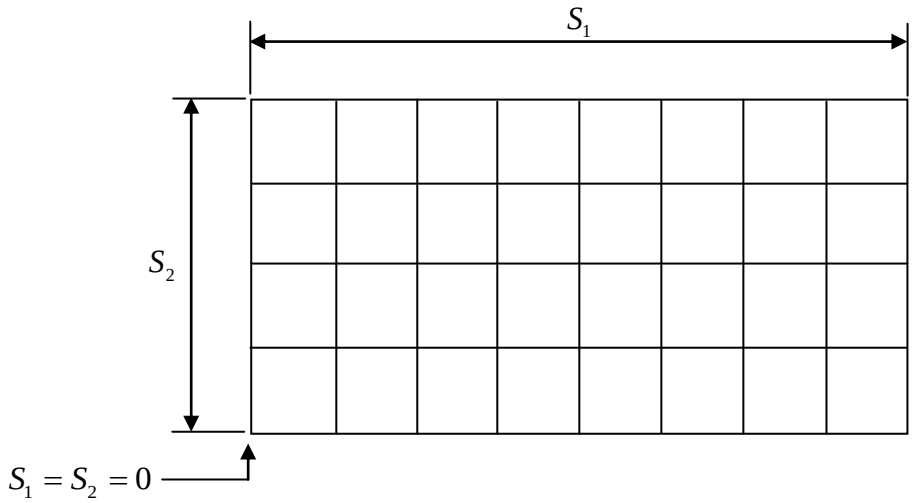

Figure 2.1 The Outline Solution Domain for the Two-Dimensional Black-Scholes Equation

Like its one-dimensional counterpart, equation (20) is a backward parabolic PDE. To be able to solve this equation uniquely it is necessary to know the final condition and the boundary conditions. For a European call option written on two assets these conditions are as follows.

The final condition is the payoff at the expiry time. Hence, at each internal node within the solution domain :

$$V(S_1, S_2, t) = \max\left(\max(S_1, S_2) - E, 0\right) \quad \text{---- (21)}$$

At all points on the bottom boundary $S_2 = 0$. Hence, the value of *V* at each internal point on this boundary is the value of an option written on the first asset only. Similarly, at all points on the left boundary $S_1 = 0$ so that the value of *V* at each internal point on this boundary is the value of an option written on the second asset only. Hence :

$$V(S_1, 0, t) = Vs_1\, t \quad \text{---- (22)} \qquad V(0, S_2, t) = Vs_2\, t \quad \text{---- (23)}$$

where $Vs_1\, t$ and $Vs_2\, t$ are the single asset solutions of the one-dimensional, linear Black-Scholes equation (6) subject to the initial condition (7) and the boundary conditions (8) and (9).

As the values of S_1 and S_2 increase it becomes increasingly likely that the option will be exercised. In this case the value of the option will tend to the value of the most expensive asset *i.e.* the asset with the largest current value. Hence :

$$V(S_1, S_2, t) \to S_1 \text{ as } S_1 \to \infty \quad \text{---- (24)}$$

and :

$$V(S_1, S_2, t) \to S_2 \text{ as } S_2 \to \infty \quad \text{---- (25)}$$

These conditions imply that at each internal point on the right boundary, $V = S_1$ and at each internal point on the top boundary, $V = S_2$.

At the bottom left-hand corner of the solution domain $S_1 = S_2 = 0$. Hence, at this point $V = 0$. The other corner points of the solution domain are discontinuities[15]. A common way of dealing with points of this type is to let their value be the average of the adjacent values on the intersecting boundaries. Hence :

- at the top left-hand corner, $V = \frac{Vs_2\, t + S_2}{2}$
- at the top right-hand corner, $V = \frac{S_1 + S_2}{2}$
- at the bottom right-hand corner, $V = \frac{S_1 + Vs_1\, t}{2}$.

The final condition and the boundary conditions for equation (20) are summarised below in Figure 2.2

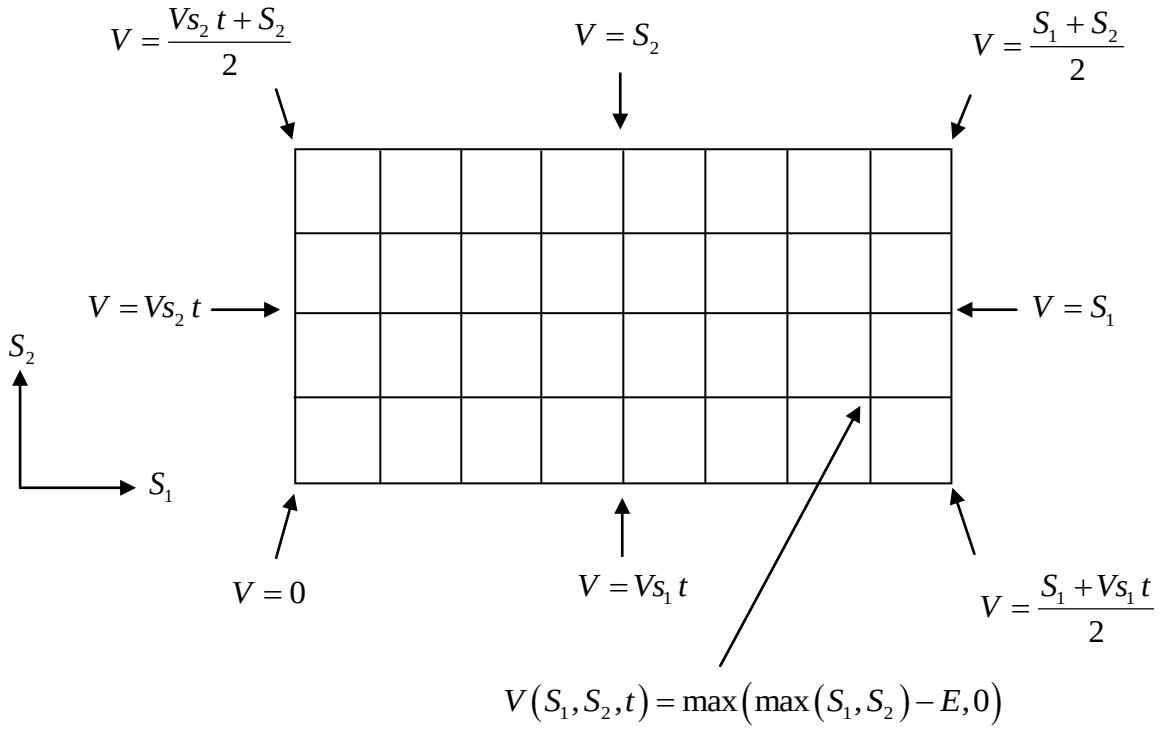

Figure 2.2 The Detailed Solution Domain for the Two-Dimensional Black-Scholes Equation

[15] For the Black-Scholes Equation and the solution technique we use, the discontinuities do not cause problems.

The seller will wish to know the current value of the option. To determine this value using the Black-Scholes model, equation (20) must be rewritten as a forward parabolic PDE. As before, this can be done by making the change of variable $\tau = T - t$, where τ is the time to expiry. Using this transformation, the equation becomes :

$$-\frac{\partial V}{\partial \tau} + \frac{1}{2}\sigma_1^2 S_1^2 \frac{\partial^2 V}{\partial S_1^2} + \frac{1}{2}\sigma_2^2 S_2^2 \frac{\partial^2 V}{\partial S_2^2} + \sigma_1 \sigma_2 \rho_{12} S_1 S_2 \frac{\partial^2 V}{\partial S_1 \partial S_2} + rS_1 \frac{\partial V}{\partial S_1} + rS_2 \frac{\partial V}{\partial S_2} - rV = 0 \quad \text{---- (26)}$$

The final condition (21) is transformed into the initial condition :

$$V(S_1, S_2, 0) = \max\left(\max(S_1, S_2) - E, 0\right) \quad \text{---- (27)}$$

The boundary conditions (22), (23), (24) and (25) change to :

$$V(S_1, 0, \tau) = Vs_1\,\tau \quad \text{---- (28)} \qquad V(0, S_2, \tau) = Vs_2\,\tau \quad \text{---- (29)}$$

$$V(S_1, S_2, \tau) \approx S_1 \text{ as } S_1 \to \infty \quad \text{---- (30)} \qquad V(S_1, S_2, \tau) \approx S_2 \text{ as } S_2 \to \infty \quad \text{---- (31)}$$

The easiest way to find the values of $Vs_1\,\tau$ and $Vs_2\,\tau$ is to use the analytical solution of the Black-Scholes equation (10) *i.e.* expression (14).

2.7 An Alternative Approach

Option prices can be calculated by solving the underlying stochastic differential equations rather than the corresponding Black-Scholes equations. However, Sauer (2012) shows that numerical methods for solving stochastic differential equations arising in finance must incorporate Monte Carlo methods for simulating the Brownian motion terms $dW(t)$ and $dW_i(t)$. Monte Carlo methods are notoriously slow and are not competitive with numerical methods for solving parabolic PDEs *e.g.* finite difference methods.

2.8 Nonlinear Black-Scholes Equations

Assumption 6. of the Black-Scholes model allows investors to continuously buy and sell quantities of the underlying asset in order to offset potential losses they may incur due to variations in the price of that asset. Ankudinova and Ehrhardt (2008) describe this process as "*continuous portfolio adjustment to hedge the position without risk*". Assumption 3. of the Black-Scholes model assumes that this portfolio adjustment can be accomplished without additional costs. However, in practice each time a quantity of the underlying asset is bought or sold a transaction cost is incurred. Since these transaction costs occur continuously they effect the price of the underlying asset over time and hence the volatilities in the

Black-Scholes equations. A number of volatility models incorporating transactions costs have been developed. This research programme considers those described by Lai *et al.* (2005) and Ankudinova and Ehrhardt (2008). It is not an aim of this research programme to give the theoretical development of these models or to evaluate their strengths and weaknesses. Interested readers should consult the references given with each model.

The volatility models described below can all be written in the general form :

$$\tilde{\sigma}^2 = \sigma^2 \left(1 + \nu_{corr}\right)$$

where $\tilde{\sigma}$ is called the modified volatility, σ is the volatility without transaction costs and ν_{corr} is called the volatility correction, (Ankudinova and Ehrhardt 2008). In the absence of transaction costs $\nu_{corr} = 0$. The volatility correction can be a function of time *t*, the time to expiry τ, the current value of the underlying asset *S*, the value of the option *V*, the first and second partial derivatives of *V* or the solution of an initial-value problem, (Lai *et al.* 2005). In these cases the Black-Scholes equation (10) becomes :

$$-\frac{\partial V}{\partial \tau} + \frac{1}{2}\tilde{\sigma}^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \quad \text{---- (32)}$$

Similarly, the Black-Scholes equation (26) becomes :

$$-\frac{\partial V}{\partial \tau} + \frac{1}{2}\tilde{\sigma}_1^2 S_1^2 \frac{\partial^2 V}{\partial S_1^2} + \frac{1}{2}\tilde{\sigma}_1^2 S_1^2 \frac{\partial^2 V}{\partial S_1^2} + \tilde{\sigma}_1\tilde{\sigma}_2\rho_{12}S_1S_2\frac{\partial^2 V}{\partial S_1 \partial S_2} + rS_1\frac{\partial V}{\partial S_1} + rS_2\frac{\partial V}{\partial S_2} - rV = 0 \quad \text{---- (33)}$$

Equations (32) and (33) are nonlinear. Unfortunately, very few analytical solutions exist for equations of this type. Solutions must usually be found using numerical methods.

2.8.1 Volatility Models

2.8.1.1 A Simulated Modified Volatility Model

Boyle and Vorst (1992) assume that the transaction cost is related to the option value *V* and follows a normal distribution. They propose a modified volatility model in the form :

$$\tilde{\sigma}^2 = \sigma^2 \left(1 + a\right)$$

where *a* is the proportional transaction cost[16] scaled by σ and the transaction time. Lai *et al.* (2005) adopt the same approach but assume that the transaction cost follows a pulse-like

[16] A transaction cost that is proportional to the number of transactions made rather than a fixed transaction cost.

distribution. They model the proportional transaction cost using a sine function in the form :

$$a = \sin\left(\frac{\pi V}{E}\right)$$

to produce a simulated modified volatility model.

2.8.1.2 Leland

Leland (1985) assumes that the transaction costs can be minimised if the transactions occur at discrete points in time rather than continuously. Leland's modified volatility model has the form :

$$\tilde{\sigma}^2 = \sigma^2\left(1 + Le\,\text{sign}\left(\Gamma\right)\right)$$

where Le is called the Leland number. This is defined as :

$$Le = \sqrt{\frac{2}{\pi}}\left(\frac{k}{\sigma\delta t}\right)$$

where *k* is the round trip transaction cost per unit dollar of the transaction[17] and δt is the transaction frequency. This is the time interval between successive revisions of the portfolio[18]. The term Γ is the *gamma* of the underlying asset. This is defined as :

$$\Gamma = \frac{\partial^2 V}{\partial S^2}$$

and is a measure of how often or by how much the portfolio needs to be hedged to maintain a risk-free position. Wilmot (2000) shows that for a long option[19] such as a European call, and in the absence of transaction costs, $\Gamma > 0$. If the same assumption is made for a European call in the presence of transaction costs then Leland's modified volatility model becomes :

$$\tilde{\sigma}^2 = \sigma^2\left(1 + Le\right)$$

In this case, the modified volatility is a constant and the Black-Scholes equations (32) and (33) are linear.

---

17 This is defined as :

$$k = \frac{S_{ask} - S_{bid}}{S_{mid}}$$

where S_{ask} is the price the seller is willing to accept for the asset, S_{bid} is the highest price the bidder is willing to pay for the asset and S_{mid} is the mid value of the asset. This is average of the current *bid* and *ask* prices.

18 Since there are 252 trading days in a year, a transaction frequency of 1 day gives $\delta t = 1/252$.

19 A option whose value is expected to increase over time.

2.8.1.3 Boyle and Vorst

Boyle and Vorst (1992) also assume that the transaction costs can be minimised if the transactions occur at discrete points in time rather than continuously. Their modified volatility model has the form :

$$\tilde{\sigma}^2 = \sigma^2 \left(1 + Le \sqrt{\frac{\pi}{2}} \operatorname{sign}(\Gamma) \right)$$

However, the term δt in their definition of the Leland number is the mean length of time for a change in the value of the underlying asset. If it is again assumed that $\Gamma > 0$ for a European call in the presence of transaction costs then this modified volatility is also constant and the Black-Scholes equations (32) and (33) are linear.

2.8.1.4 Barles and Soner

Barles and Soner (1998) propose a modified volatility model in the form[20] :

$$\tilde{\sigma}^2 = \sigma^2 \left(1 + \psi \left(e^{r\tau} \alpha^2 S^2 \Gamma \right) \right)$$

where *r* is the risk-free interest rate, $\alpha^2 = \gamma N a^2$, γ is the risk aversion factor[21], *N* is the number of assets bought or sold, *a* is the proportional transaction cost and $\psi(x)$ is the solution of the initial-value problem :

$$\frac{d\psi}{dx} = \frac{\psi(x) + 1}{2\sqrt{x\psi(x) - x}} \qquad x \neq 0 \quad , \quad \psi(0) = 0 \qquad \text{---- (34)}$$

To calculate the values of $\psi(x)$ in the Barles and Soner model equation (34) must be solved over a suitable range of values of *x*. Then, as each argument $x = e^{r\tau} \alpha^2 S^2 \Gamma$ is calculated, the corresponding value of $\psi(x)$ is found by interpolation.

2.8.1.5 The Risk Adjusted Pricing Methodology

The Risk Adjusted Pricing Methodology (RAPM) model was developed by Kratka (1998) and subsequently refined by Jandačka and Ševčovič (2005). Here, "*the optimal time-lag δt between transactions is chosen to minimise the sum of the rate of the transaction costs and the rate of risk from an unprotected portfolio*", (Ankudinova and Ehrhardt 2008). The

---

[20] The presence of the exponential term in this model means that is can be used only for calculating the modified volatility over short periods of time.

[21] A common measure used for γ is the standard deviation of the returns on the underlying asset.

modified volatility model in this case is :

$$\tilde{\sigma}^2 = \sigma^2 \left(1 + 3 \left(\frac{C^2 M}{2\pi} S \Gamma \right)^{\frac{1}{3}} \right)$$

where C is the risk premium measure[22] and M is the coefficient of transaction costs. This is defined as :

$$M = \frac{k\sigma S}{\sqrt{2\pi}} |\Gamma| \frac{1}{\delta t}$$

where k is the round trip transaction cost per unit dollar of the transaction.

2.9 Black-Scholes Equations with Stochastic Volatility

Volatility is not constant as required by the Black-Scholes model. Empirical studies show that volatility is highly variable, even in the absence of transaction costs. For example, consider the historical volatility data shown in Figure 2.3. This data was calculated by applying the formulae given in the footnote on page 8 to the NASDAQ[23] share price data[24] for the multimedia company Apple for the period 14th January 2013 to 14th January 2014 inclusive.

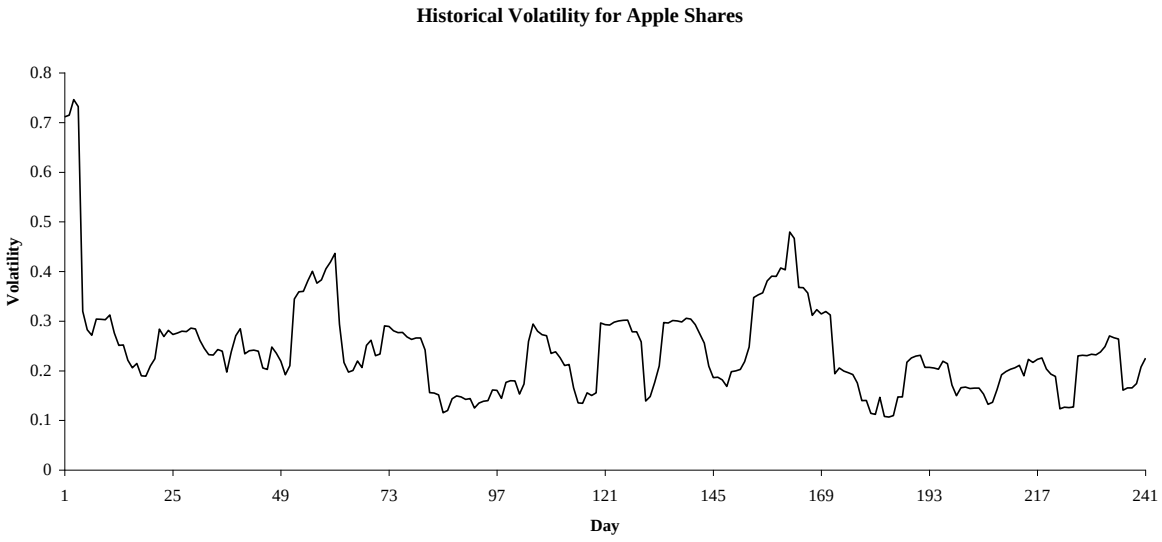

Figure 2.3 Historical Volatility for Apple Shares

Clearly, the volatility for Apple shares is not constant. In this situation it is natural to model the volatility as a stochastic process. Wilmot (2000) states that by modelling volatility in this

22 A measure of the amount by which an assets expected rate of return exceeds the risk-free interest rate.

23 NASDAQ is an acronym for National Association of Securities Dealers Automated Quotations. It is America's second largest stock exchange after the New York stock exchange on Wall Street.

24 The data used to calculate the volatilities was the daily closing share prices in $.

way, options can be valued more accurately.

The general stochastic volatility model has the form :

$$d\sigma = p(S,\sigma,t)dt + q(S,\sigma,t)dB(t) \quad \text{---- (35)}$$

where $p(S,\sigma,t)dt$ is called the *volatility of volatility*, $q(S,\sigma,t)$ is called the *drift of volatility* and $dB(t)$ is a Brownian motion that is correlated with $dW(t)$ in equation (1) with correlation coefficient ρ, (Wilmot 2000).

A number of stochastic volatility models have been developed, each one using different formulae for the functions *p* and *q*. Interested readers should consult a reference such as Jäckel (2005). One of the most commonly used is the Heston model, (Heston 1993). His stochastic volatility model has the form :

$$dv_t = \theta(\omega - v_t)dt + \xi\sqrt{v_t}dB(t)$$

where v_t is a function that models the variance of *S*, the current value of the underlying asset, θ is the rate at which the volatility reverts towards its long-term mean, ω is the mean long-term volatility and ξ is the volatility of volatility. Heston (1993) derives a closed formula for the value of a European call option written on an asset with stochastic volatility modelled by this equation. However, analytical solutions of this type are rare. To find the value of an option with stochastic volatility three methods are commonly used *i.e.*

1. Equation (35) can be used instead of equation (1) to derive the corresponding Black-Scholes equation. Wilmot (2000) uses this approach to derive the equation :

   $$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \rho\sigma S q \frac{\partial^2 V}{\partial S \partial \sigma} + \frac{1}{2} q^2 \frac{\partial^2 V}{\partial \sigma^2} + rS\frac{\partial V}{\partial S} + (p - \lambda q)\frac{\partial V}{\partial \sigma} - rV = 0 \quad \text{---- (36)}$$

   for a European call option[25]. Here, $\lambda(S,\sigma,t)$ is a function called the *market price of volatility risk*. Equation (36) is nonlinear and must be solved using numerical methods. A parallel, Laplace transform based algorithm for solving problems of this type is described in Chapter 9.

2. Statistical procedures can be applied to historical volatility data to produce estimates for the functions $p(S,\sigma,t)$ and $q(S,\sigma,t)$ in the stochastic volatility model. These can then be substituted into the Fokker-Planck equation :

[25] Equation (36) has the same initial and boundary conditions as the Black-Scholes equation (6).

$$\frac{\partial p}{\partial t} = \frac{1}{2}\frac{\partial^2(\beta^2 p)}{\partial \sigma^2} - \frac{\partial(\alpha p)}{\partial \sigma}$$

where α and β are the estimates of $p(S,\sigma,t)$ and $q(S,\sigma,t)$ obtained from the data. This equation "*describes the evolution over time of the probability density function of a random variable described by a stochastic differential equation*", (Wilmot 2000). In this case the closed form solution of this equation[26] is the probability density function for σ. This expression can be used to forecast the values of σ at future points in time, (Wilmot 2000).

3. Time series methods can be applied to historical volatility data to forecast the values of σ at future points in time.

If method 2. or method 3. is used then a set of constant volatility values is available for calculating the required option values. A computer program can simply read the corresponding pairs of *S* and σ values one at a time and then calculate the value of the option using the analytical solution (14). This procedure can be parallelised by assigning each processor a range of volatility values to process. When a program of this kind is used to calculate the values of a European call option written on a single share in Apple, the option value data shown in Figure 2.4 is obtained. The exercise price *E* is taken to be \$380 , the risk-free interest rate *r* is taken to be 0.05 and the required values of τ are calculated within the program.

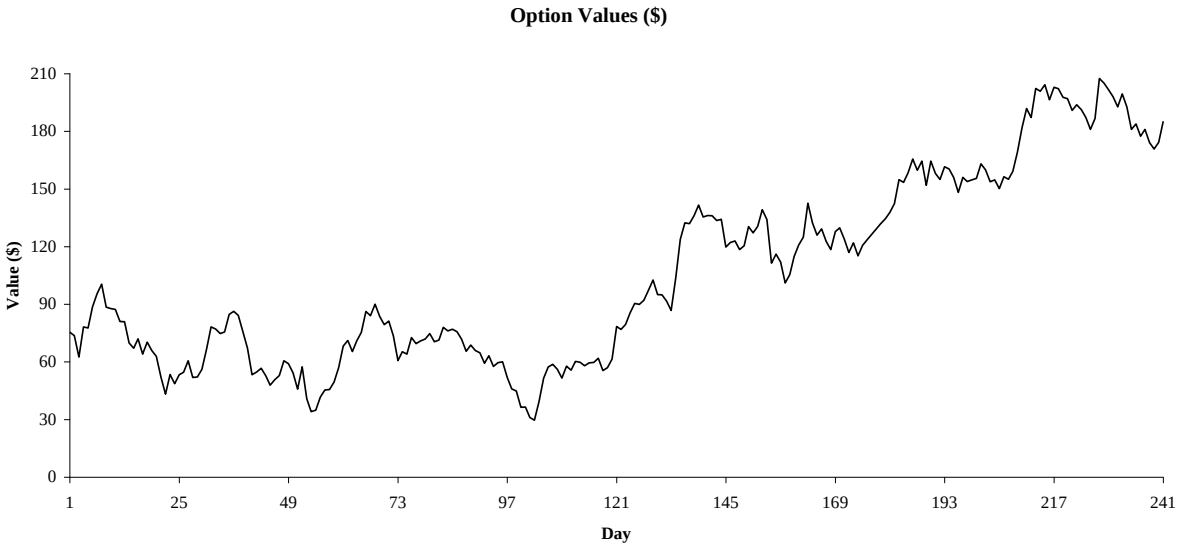

Figure 2.4 Option Values for Apple Shares

[26] Details of how to solve the Fokker-Planck equation can be found in Risken (1996).

It can be seen that the option values follow an upward trend in line with the share price data and the boundary condition $V(S,\tau) \approx S$ as $S \to \infty$. Like the historical volatility data, the option values also exhibit time series like behaviour.

Stochastic volatility is an important issue in option pricing and volatility has become a tradable asset in its own right. For example, a VIX option is a non-equity option in which the underlying asset is the volatility index, (McKhann 2006).

2.10 Chapter Summary

Background information has been provided on financial markets, financial options and the Black-Scholes equations that are used widely for valuing options in the financial sector. In Chapter 1 it was explained that one way to find fast and accurate solutions of these equations is to use parallel numerical algorithms that are based on the Laplace transform. Chapter 3 will therefore describe Laplace transform methods and to explain how these methods can facilitate parallel solutions.

2.11 Contribution to Knowledge

This chapter has developed and implemented a relatively simple procedure for calculating option values in the case where the volatility is stochastic and historical volatility data is available.

Chapter 3

The Laplace Transform Method

3.0 Introduction

The Laplace transform method can be used for solving both ODEs and PDEs. This chapter describes this method and considers its advantages and disadvantages. The advantage most relevant to this research programme is that it allows the time domain in diffusion problems such as Black-Scholes equations to be decomposed so that parallel computing methods can be used for finding their solutions. The main disadvantages are that it can be difficult to find inverse Laplace transforms, especially when tables cannot be used and that Laplace transform inversion is ill-posed. This chapter considers alternative methods for overcoming these problems, in particular numerical inversion algorithms.

3.1 Background

Parabolic PDEs can be solved using a variety of analytical and numerical procedures. Authors such as Edwards and Penney (2008), Morton and Mayers (2008), Smith (2004) and Wilmot *et al.* (1999) describe the most commonly used methods. This research programme considers the Laplace transform method. A description of the history and development of this technique can be found in Deakin (1992).

3.2 Definition

Consider a function $x(t)$. The Laplace transform of $x(t)$ is defined as :

$$\mathrm{L}\left[x(t)\right] = \int_0^\infty e^{-\lambda t} x(t)\, dt = \bar{x}(\lambda)$$

where λ is the transform variable. Widder (1946) states that sufficient conditions for the existence of $\mathrm{L}\left[x(t)\right]$ are that :

- the defining integral converges as $t \to \infty$
- the function $x(t)$ is piecewise continuous[27] on the interval $0 \le t < \infty$

[27] A function is piecewise continuous if it can be divided into a finite number of sections so that it is continuous on the interior of each section and that its value remains bounded as its argument approaches the end points of the sections.

- the function $x(t)$ is of exponential order *i.e.* $\exists$ constants c, $M > 0$ and $T > 0$ such that $|x(t)| \leq Me^{ct} \ \forall t > T$.

The inverse Laplace transform of $\bar{x}(\lambda)$ is given by the Bromwich contour integral :

$$\mathrm{L}^{-1}\left[\bar{x}(\lambda)\right] = \frac{1}{2\pi j}\int_{\gamma-j\infty}^{\gamma+j\infty} e^{\lambda t}\bar{x}(\lambda)\,d\lambda = \begin{cases} x(t) & t > 0 \\ 0 & t < 0 \end{cases}$$

Widder (1946). To ensure that the contour path is within the region of convergence, the constant γ is chosen so that the singularities[28] s_i of $\bar{x}(\lambda)$ lie to the left of the vertical line $\mathrm{Re}(\lambda) = \gamma$ in the complex plane (Laverty 2003) *i.e.*

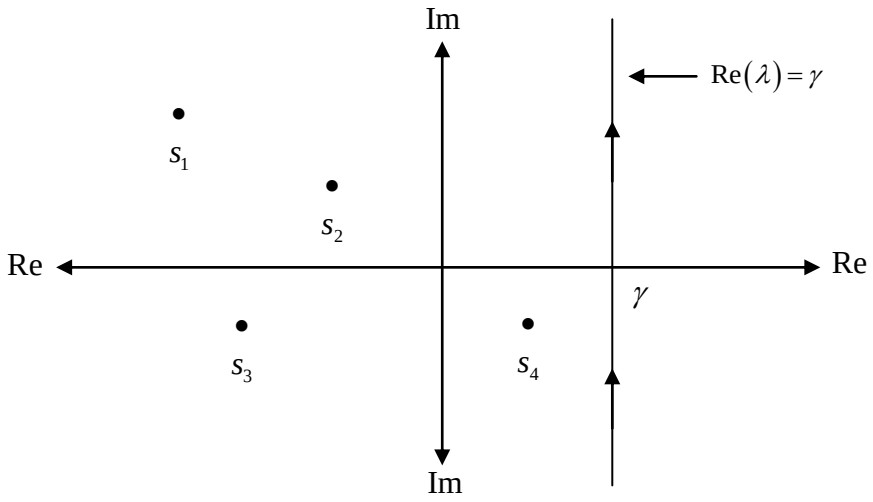

Figure 3.1 The Bromwich Contour

3.3 The Method

In outline, the procedure for solving a time-dependent ODE or PDE using the Laplace transform method is :

- take the Laplace transform with respect to time. This reduces the number of independent variables by one and simplifies the problem being solved *e.g.* ODEs become algebraic equations, one-dimensional PDEs become ordinary differential equations, *etc.*
- substitute the initial condition(s)
- solve the simplified equation to obtain the Laplace transform of the solution

[28] A singularity of a function $f(z)$ is a point *a* such that as $z \to a$, $f(z) \to \infty$.

- find the inverse Laplace transform to give the solution of the original differential equation in the time domain. The inverse Laplace transform of many standard functions can be found from tables such as those given in Davies and Crann (2004).

3.4 Advantages

The Laplace transform method has a number of advantages over time-marching, finite difference methods for solving parabolic PDEs such as those described in Morton and Mayers (2008), Smith (2004) and Buetow and Sochacki (2000). The Laplace transform method :

- avoids the restrictions described in Smith (2004) that must be imposed upon the step length in the case of explicit time-marching methods in order to ensure accuracy and stability, (Davies and Crann 2010)
- is more computationally efficient when the solution is required at a single point in time. The Laplace transform method does not require the solutions to be found at the intermediate values, (Davies *et al.* 2007). This is particularly important in mathematical finance where it is often required to calculate the value of an option at a particular time in the future. By comparing the future value with the value at which the option is currently being traded, dealers can determine whether the option value is likely to increase or decrease in the future and formulate an appropriate trading strategy. The Black-Scholes model is a relatively simple tool for calculating future values
- allows the time domain to be decomposed so that the equation can be solved using parallel computing methods, (Crann *et al.* 2007).

3.5 Disadvantage

The main disadvantage of the Laplace transform method is that, if tables cannot be used then it can be difficult to find the inverse Laplace transform. The Bromwich contour integral is a Fredholm integral equation of the first kind, (McWhirter and Pike 1978). Equations of this kind are ill-posed[29] and hence Laplace transform inversion is prone to instability. Kano (2010) states that the ill-posedness of the Bromwich contour integral is caused by the exponential term in the integrand that magnifies any algorithmic or computational errors in the method being used to find the inverse. However, for a more rigorous explanation of why

[29] Fredholm integral equations of the first kind are ill-posed because they fail the third condition in the test given by Hadamard. See the footnote at the bottom of page 1. With an equation of this kind, small changes in the data produce large changes in the solution *i.e.* the solution does not depend continuously on the data. Fredholm integral equations of the second kind are well-posed.

Laplace transform inversion is prone to instability see Epstein and Schotland (2008).

3.6 Alternative Methods of Laplace Transform Inversion

Authors such as Abate and Whitt (2006), Skachkov (2002) and Davies (2002) have proposed methods for finding the inverse Laplace transform that are based on evaluating the Bromwich contour integral. However, other authors have criticised this approach claiming that it is too difficult to be of practical value (Laverty 2003) and ill-posed, (Crann 2005) and (Wing 1991). An alternative approach is to use a numerical inversion algorithm. For a review of the most commonly used procedures, interested readers should consult references such as Kuhlman (2012), Craddock *et al.* (2000) and Davies and Martin (1979). Although numerical Laplace transform inversion is also ill-posed, this approach has been used successfully in a wide variety of application areas, (NanoDotTek 2007). Furthermore, Kano (2010) suggests that numerical Laplace transform inversion is safe provided that the inversion algorithms are implemented accurately *e.g.* using double precision arithmetic and that a range of algorithms are evaluated to find one that is fast and accurate for the application to which it is being applied. This advice is followed throughout the remainder of this research programme.

To determine the fastest and most accurate numerical inversion algorithm for solving Black-Scholes equations four widely used methods that have been used successfully in other applications are evaluated. These methods are Stehfest's method, Stehfest (1970), the shifted Legendre polynomial method, Zakian and Littlewood (1973), the Jacobi polynomial method, Miller and Guy (1966) and the Laguerre polynomial method, Piessens and Branders (1971) and Weeks (1966). The last of these methods is the one identified by Davies and Martin (1979) as being the best overall performer. A number of performance comparisons have been completed previously. However, this research programme extends this work by evaluating the numerical inversion algorithms in a financial context.

3.6.1 Stehfest's Method

The Stehfest inversion method is based upon a stochastic inversion process described by Gaver (1966). Here, the numerical inverse Laplace transform is given by a weighted sum of the Laplace transform values :

$$V \approx \frac{\ln 2}{\tau} \sum_{j=1}^{m} \omega_j \bar{V}\left(\lambda_j\right)$$

where m must be even. The values of the transform variable λ_j are calculated using :

$$\lambda_j = j\frac{\ln 2}{\tau} \quad j = 1, 2, \ldots, m$$

and the weights ω_j are given by :

$$\omega_j = (-1)^{\frac{m}{2}+j} \sum_{k=\frac{1}{2}(1+j)}^{\min\left(j,\frac{m}{2}\right)} \frac{k^{\frac{m}{2}}(2k)!}{\left(\frac{m}{2}-k\right)!k!(k-1)!(j-k)!(2k-j)!}$$ (Stehfest 1970).

3.6.2 The Shifted Legendre Polynomial Method

The shifted Legendre polynomial (SLP) method is a member of a class of numerical inversion algorithms in which the inverse Laplace transform is given by a weighted sum of exponential functions :

$$V \approx \frac{1}{\kappa} \sum_{k=0}^{m} C_k P_k(z)$$

The values of the transform variable λ_k are calculated using :

$$\lambda_k = \frac{k+1}{\kappa} \quad k = 0,1,\ldots,m$$

The Legendre polynomials $P_k(z)$ are given by :

$$P_k(z) = a_{ko} + a_{k1}z + a_{k2}z^2 + \ldots + a_{kk}z^k$$

with :

$$z = e^{-\frac{\tau}{\kappa}}, \quad a_{kj} = (-1)^{k+j} \binom{k+j}{k} \binom{k}{j} \quad 0 \le j \le k \; ; k = 0,1,\ldots,m \quad \text{and} \quad \binom{n}{r} = \frac{n!}{r!(n-r)!}$$

The coefficients C_k are given by :

$$C_k = (2k+1) \sum_{j=0}^{k} a_{kj} \bar{V}(\lambda_j)$$

(Zakian and Littlewood 1973). The term κ is an arbitrary parameter[30]. Aral and Gülçat (1977) discuss possible values for this parameter and suggest setting κ to the value of the independent variable *e.g.* $\kappa = \tau$. However, (Crann 2005) found that using $\kappa = 1$ gave good results and that varying κ made little difference to the accuracy of the results obtained.

[30] Aral and Gülçat (1977) and Crann (2005) use the symbol τ to denote this parameter. The symbol κ is used here to avoid confusion with the use of τ to denote the time to expiry.

3.6.3 The Jacobi Polynomial Method

The Jacobi polynomial method is a member of a class of numerical inversion algorithms in which the inverse Laplace transform is given by a weighted sum of orthogonal polynomials[31], (Laverty 2003) :

$$V \approx \sum_{n=0}^{N-1} c_n P_n^{(0,\beta)}\left(2e^{-\delta\tau} - 1\right)$$

where β and δ are arbitrary parameters. The values of the transform variable λ_n are calculated using :

$$\lambda_n = \delta\left(\beta + n + 1\right) \quad n = 0,1,\ldots,N-1$$

The coefficients c_n are given by :

$$c_0 = \left(\beta+1\right)\delta\bar{V}\left(\lambda_0\right) \qquad c_1 = \left(\beta+2\right)\left(\beta+3\right)\left[\delta\bar{V}\left(\lambda_1\right) - \frac{c_0}{\beta+2}\right]$$

$$c_n = \frac{\prod_{j=1}^{n+1}\left(n+\beta+j\right)}{\prod_{j=0}^{n-1}\left(n-j\right)}\left[\delta\bar{V}\left(\lambda_n\right) - \sum_{m=0}^{n-1} c_m \frac{\prod_{j=0}^{m-1}\left(n-j\right)}{\prod_{j=1}^{m+1}\left(n+\beta+j\right)}\right], \qquad n \geq 2$$

The Jacobi polynomials $P_n^{(0,\beta)}\left(x\right)$ are given by :

$$P_n^{(0,\beta)}\left(x\right) = \frac{\left(-1\right)^n}{2^n n!}\left(1+x\right)^{-\beta}\frac{d^n}{dx^n}\left[\left(1-x\right)^n\left(1+x\right)^{n+\beta}\right]$$

(Miller and Guy 1966, Laverty 2003). For computational purposes it is more convenient to use the formula for the Jacobi polynomials given in (Abramowitz and Stegum 1972) *i.e.*

$$P_n^{(0,\beta)}\left(x\right) = \frac{1}{2^n}\sum_{m=0}^{n}\binom{n}{m}\binom{n+\beta}{n-m}\left(x-1\right)^{n-m}\left(x+1\right)^m \quad \text{where} \quad \binom{n}{r} = \frac{n!}{r!\left(n-r\right)!}.$$

Miller and Guy (1966) suggest that the most accurate results are obtained when the parameters β and δ are assigned values in the ranges $-0.5 \leq \beta \leq 5.0$ and $0.05 \leq \delta \leq 2.0$. However, a preliminary investigation conducted during this research programme found that varying the values of β and δ made little difference to the accuracy of the results obtained.

---

[31] Suppose that two polynomials *f* and *g* are evaluated at regular intervals to produce approximating vectors $\underline{f}$ and $\underline{g}$ respectively. Then, *f* and *g* are orthogonal if these vectors are perpendicular *i.e.* if $\underline{f} \cdot \underline{g} = 0$.

3.6.4 The Laguerre Polynomial Method

In the Laguerre polynomial method the Laplace transform is approximated using a bilinear transformation of a complex variable[32] (Davies and Martin 1979) :

$$V \approx t^{\alpha} e^{-ct} \sum_{k=0}^{N} a_k \frac{k!}{(\alpha+k)!} L_k^{\alpha}\left(\frac{t}{T}\right)$$

where α, c, N and T are parameters. The Laguerre polynomials can be calculated using the recursive formulae :

$$L_0^{\alpha}(t) = 1$$

$$L_1^{\alpha}(t) = 1 + \alpha - t$$

$$n L_n^{\alpha}(t) = (2n + \alpha - 1 - t) L_{n-1}^{\alpha}(t) - (n - 1 + \alpha) L_{n-2}^{\alpha}(t)$$

The coefficients a_k can be calculated using :

$$a_0 = \frac{1}{N+1} \sum_{j=0}^{N} h(\theta_j)$$

$$a_k = \frac{2}{N+1} \sum_{j=0}^{N} h(\theta_j) \cos(k\theta_j)$$

where :

$$\theta_j = \left(\frac{2j+1}{N+1}\right)\frac{\pi}{2}$$

and :

$$h(\theta) = \mathrm{Re}\left(\left(\frac{1+\cot\left(\frac{\theta}{2}\right)}{2T}\right)^{\alpha+1} \bar{V}\left(\frac{1}{2T} + c + i\left(\frac{\cot\left(\frac{\theta}{2}\right)}{2T}\right)\right)\right), \quad i = \sqrt{-1}$$

The complex expression within the $\bar{V}$ bracket in the formula for $h(\theta)$ is the transform variable for this algorithm.

---

[32] A bilinear transformation Z of an independent complex variable z is a relationship in the form :

$$Z = \frac{a + bz}{c + dz}$$

where $a, b, c, d \in \mathbb{C}$.

Authors such as Weeks (1966), Piessens and Branders (1971) and Davies and Martin (1979) make recommendations for the optimal values of the parameters α, c, N and T. However, while investigating this algorithm it was found that the optimal parameter values *i.e.* the values that gave the most accurate numerical inverse, depended upon the function to which the method was being applied. For example, when inverting the Laplace transform of the exponential function e^{-t}, the optimal parameter values were found to be $\alpha = 0$, $c = 0.7$, $N = 10$ and $T = 3.1$. When inverting the Laplace transforms arising in the solution of the Black-Scholes equation (10), the optimal parameter values were found to be $\alpha = 0$, $c = 2.4$, $N = 6$ and $T = 0.1$[33].

3.7 Chapter Summary

This chapter has described the Laplace transform method, considered its advantages and disadvantages and described four numerical algorithms that can be used for finding inverse Laplace transform values. Before proceeding to use Laplace transform based methods for solving Black-Scholes equations, the methodology used in the investigations that follow will be described.

3.8 Contribution to Knowledge

This chapter has established the optimal parameter values to use in the Laguerre polynomial method when the algorithm is used to invert the Laplace transforms arising in the solution of the Black-Scholes equation (10).

[33] The optimal parameter values were determined by experimentation *i.e.* by calculating the numerical inverse for parameter values in the ranges $\alpha = 0$, $c = 0.1(0.1)5.1$, $N = 1(1)20$, $T = 0.1(0.1)6.1$ and choosing the combination that gave the most accurate solution.

Chapter 4

Methodology

4.0 Introduction

The methodology explains how the investigations described later in this dissertation are conducted. It describes the programming environment used, the measures used to evaluate the performance of the algorithms developed, the method of data collection used, issues in the design of the parallel programs and the parameter values used in the numerical inversion algorithms.

4.1 Programming Environment

The numerical algorithms developed in this research program are implemented in Fortran 90. This programming language was chosen because :

- it contains a range of built-in mathematical functions and provides facilities to support accurate numerical computation *e.g.* double precision arithmetic, (Metcalf and Reid 2006)
- it can be linked to parallel development environments (Snir *et al.* 1996).

The numerical algorithms are implemented sequentially and/or in parallel. All programs are run on a 96-node cluster/blade system. The full specification of this system is given in Appendix A.

Commonly used parallel development environments are PVM (Parallel Virtual Machine), Sunderam (1990) and the MPI (Message Passing Interface), Snir *et al.* (1996) and Gropp *et al.* (1999). Geist *et al.* (1996) compare the features, strengths and weaknesses of each system. They conclude that when using a cluster/blade system, the MPI should be used because it provides a larger set of communication functions in particular, asynchronous communication, faster inter-processor communication and produces code that is more portable across different platforms. The first two advantages are particularly important in the context of this research programme. MPI parallelism is illustrative and many other means of obtaining parallel computation exist. Some may be more or less apposite - for example GPU or the XEON PHI accelerator *etc.*

4.2 Measures of Performance

The numerical algorithms are evaluated in terms of their speed and accuracy.

4.2.1 Measures of Speed

Three measures of speed are used. For a measure of absolute speed Burkardt (2010) recommends using the program wall time. This is the difference between the CPU time at the beginning of the program and the CPU time at the end of the program. The MPI provides a built-in function for collecting data of this type. The measures of relative speed used are :

- the parallel/sequential speed up. Magoules(2010) defines this as :

$$\text{Parallel/Sequential Speed Up} = \frac{\text{Execution Speed of Sequential Program}}{\text{Execution Speed of Parallel Program}}$$

- the parallel/sequential speed up per processor[34].

4.2.2 Measure of Accuracy

For a measure of accuracy, the normalised root mean square deviation (NRMSD) between the numerical solutions n_i and the analytical solutions a_i is used. This is defined as :

$$\text{NRMSD} = \frac{1}{\left(n_{\max} - n_{\min}\right)} \sqrt{\frac{\sum_{i=1}^{m} \left(n_i - a_i\right)^2}{m}}$$

where *m* is the number of pairs of solutions. This is the measure of accuracy used by Davies and Martin (1979) and is a commonly used measure of the differences between the values predicted by a model and the values actually observed, (Schiller *et al.* 2008).

4.3 Method of Data Collection

The programs implementing the numerical algorithms are largely CPU-bound *i.e.* perform no input and minimal output. Hence, the load on the cluster should not have a significant effect upon the program wall times. However, to allow accurate performance comparisons to be made, all programs are run at the same time *i.e.* under the same load conditions. Each program is also run 100 times. The wall time data collected is then used to calculate summary statistics. As in all empirical research the data collected contains the occasional statistical outlier *e.g.* a program wall time for a complicated algorithm being smaller than the program wall time for a simple one. Values of this type should be ignored and only the general trends in the data should be considered.

[34] This is also a measure of the efficiency of a parallel numerical algorithm.

4.4 Parallel Program Design

The most obvious reason for implementing a numerical algorithm in parallel is the potential for increased execution speed. This is particularly important in areas such as mathematical finance where rapid results can give traders a competitive edge over their rivals. However, this is not the only reason. Many algorithms can be described more naturally as a number of simultaneously executing tasks rather than as a sequence of individual steps, (Magoules 2010).

The parallel programs are designed using the master/slave model. Here, each parallel program contains a master processor and a number of slave processors. The master processor is responsible for allocating work and data to the slave processors and for calculating and presenting the final results. The slave processors are each responsible for performing part of the overall computation and for returning their results to the master processor. Baldo *et al.* (2005) state that this is the most common way to design parallel programs implemented on shared data/message passing systems like the MPI and is the method that produces the fastest execution speeds.

4.4.1 Inter-Processor Communication

Communication between processors is achieved using synchronous and asynchronous calls as appropriate. If the receiving processor cannot proceed without the data being sent *e.g.* when it is waiting for the weights/parameters to use in the numerical inversion algorithm or the range of t (or τ) values for which it is responsible then synchronous calls such as *mpi_send*, *mpi_receive* and *mpi_bcast* are used. However, if the receiving processor can be doing other work *e.g.* receiving and processing data from another processor then asynchronous calls such as *mpi_reduce* are used.

4.4.2 Functional Decomposition Verses Domain Decomposition

To parallelise a numerical algorithm a choice is available between a functional decomposition and a domain decomposition (Fitzharris *et al.* 2012), (Grama *et al.* 2003). In the context of numerical Laplace transform inversion this means a choice between assigning each slave processor part of the each inversion calculation *e.g.* the calculations associated with a particular weight/term in the inversion formula or assigning each slave processor part of the solution domain *e.g.* the calculations associated with a range of t (or τ) values. To determine the best method to use in terms of speed and accuracy, the numerical inversion algorithms were implemented using each decomposition method. Performance data was then collected for the range of test functions and parameter values used by Davies and Martin (1979). The

number of weights/terms used in the numerical inversion algorithms was varied in the range 6(2)(16). In the programs based upon a functional decomposition the number of processors used was $n+1$, where n is the number of weights/terms used in the numerical inversion algorithm. In the programs based upon a domain decomposition the number of processors used was always the same as the number used in the corresponding functional decomposition program. This ensured that an accurate performance comparison could be made between the two decomposition methods.

The results presented below were obtained by numerically inverting the Laplace transform of the function $x(t) = e^{-t}$ for values of t in the range 0(0.01)100. However, almost identical results were obtained for the other functions tested.

4.4.2.1 NRMSD Values

The tables below give the NRMSD values for each numerical inversion algorithm and each decomposition method.

Stehfest's Method

Number of Weights :	6	8	10	12	14	16
Sequential Program	0.0142322545	0.0036774285	0.0008157130	0.0002046827	0.0000594815	0.0000160370
Functional Decomposition	0.0142308042	0.0036768908	0.0008156957	0.0002047592	0.0000595126	0.0000160453
Domain Decomposition	0.0142242159	0.0036733632	0.0008146399	0.0002046388	0.0000594235	0.0000160071

Table 4.1 NRMSD Values for Stehfest's Method

The SLP Method

Number of Weights :	6	8	10	12	14	16
Sequential Program	0.0000000000	0.0000000000	0.0000000001	0.0000000001	0.0000000223	0.0000246744
Functional Decomposition	0.0000000000	0.0000000000	0.0000000000	0.0000000001	0.0000000223	0.0000008250
Domain Decomposition	0.0000000000	0.0000000000	0.0000000001	0.0000000001	0.0000000223	0.0000246229

Table4.2 NRMSD Values for the SLP Method

The Jacobi Polynomial Method

Number of Terms :	6	8	10	12	14	16
Sequential Program	0.0000000007	0.0000001333	0.0000265723	0.0024033179	0.1361245659	0.4325653824
Functional Decomposition	0.0000000007	0.0000001333	0.0000265799	0.0024042109	0.1361903713	0.4326578113
Domain Decomposition	0.0000000007	0.0000001332	0.0000265412	0.0024028027	0.1359877938	0.4317798724

Table 4.3 NRMSD Values for the Jacobi Polynomial Method

The Laguerre Polynomial Method

Number of Terms :	6	8	10	12	14	16
Sequential Program	0.0655779344	0.0587723753	0.0498944950	0.0469411261	0.0476426049	0.0486373446
Functional Decomposition	0.0655779344	0.0587723753	0.0498944950	0.0469411261	0.0476426049	0.0486373446
Domain Decomposition	0.0655420217	0.0587101715	0.0498289249	0.0469159920	0.0475790506	0.0485350214

Table 4.4 NRMSD Values for the Laguerre Polynomial Method

The zero NRMSD values shown in Table 4.2 are numbers that are smaller than 10^{-10}. It can be seen from these tables that the choice of decomposition method does not appear to significantly effect the accuracy of the inverse Laplace transform values obtained.

The data in these tables suggest that the SLP method is the most accurate numerical inversion algorithm. However, this was not generally the case. For all other test functions Stehfest's method gave the most accurate results[35].

4.4.2.2 Execution Speeds

The graphs below summarise the wall time data and the parallel/sequential speed up data collected. The minimum wall time is considered to be the most accurate measure of absolute speed.

Stehfest's Method

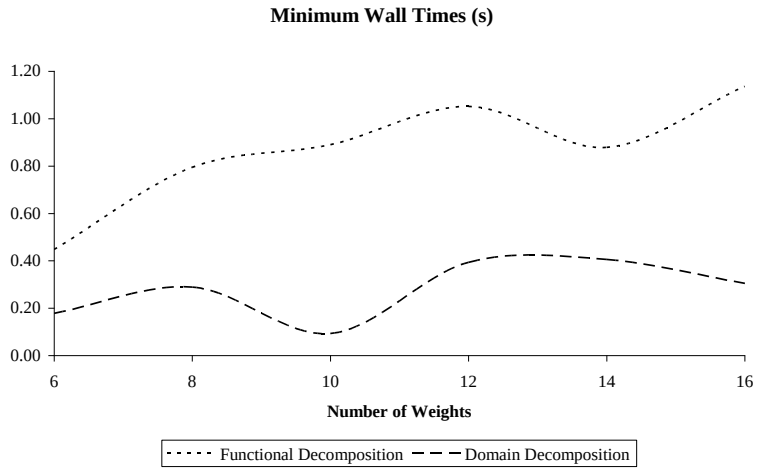

Figure 4.1 Minimum Wall Times for Stehfest's Method

[35] Zakian and Littlewood (1973) show that when $x(t) = e^{-t}$, the truncation error in the SLP method is zero. Hence highly accurate inverse Laplace transform values can be expected in this case.

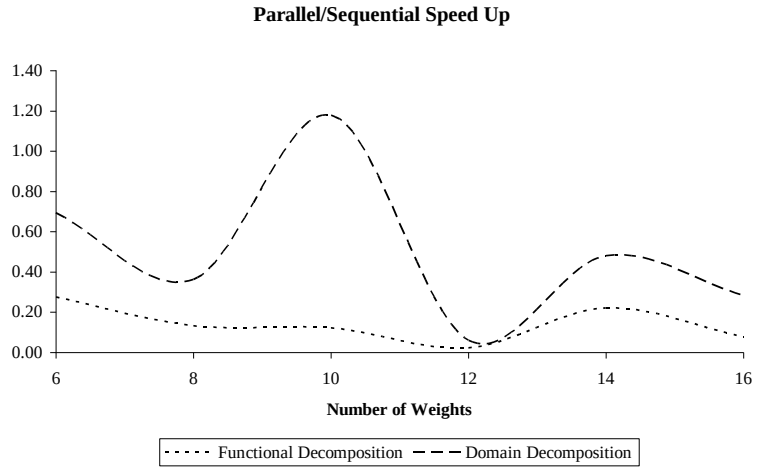

Figure 4.2 Parallel/Sequential Speed Up for Stehfest's Method

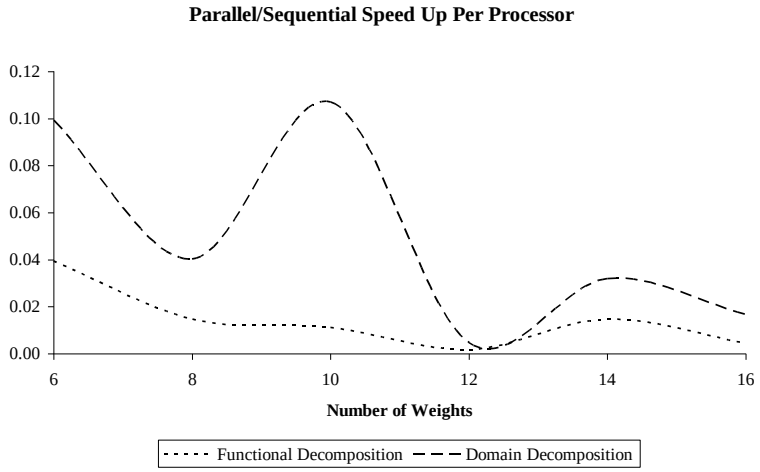

Figure 4.3 Parallel/Sequential Speed Up Per Processor for Stehfest's Method

The SLP Method

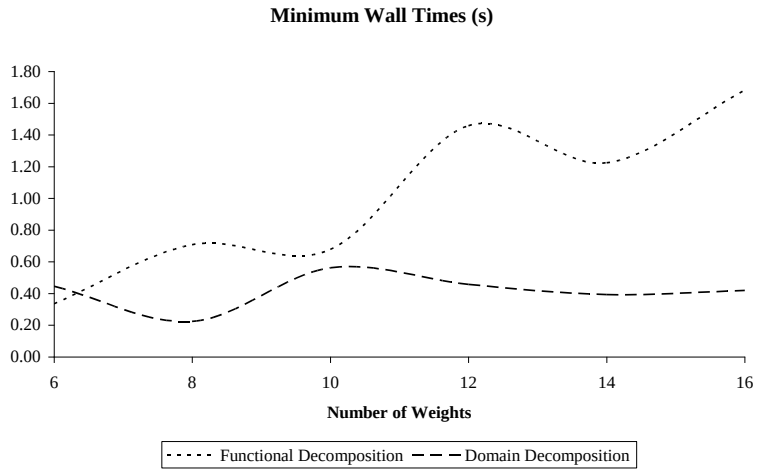

Figure 4.4 Minimum Wall Times for the SLP Method

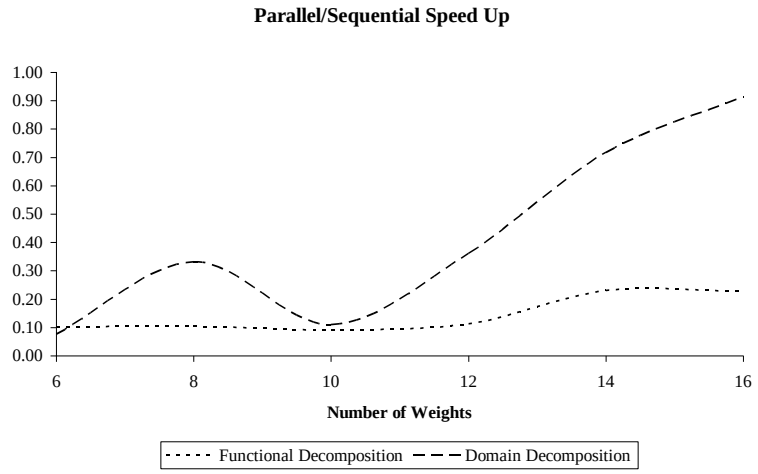

Figure 4.5 Parallel/Sequential Speed Up for the SLP Method

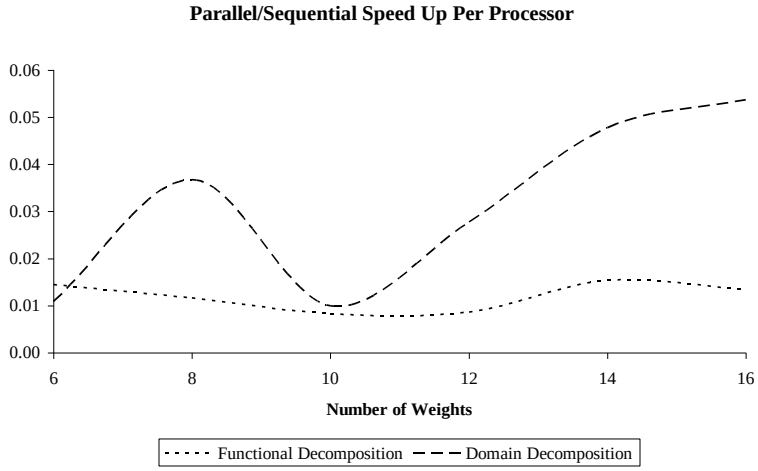

Figure 4.6 Parallel/Sequential Speed Up Per Processor for the SLP Method

The Jacobi Polynomial Method

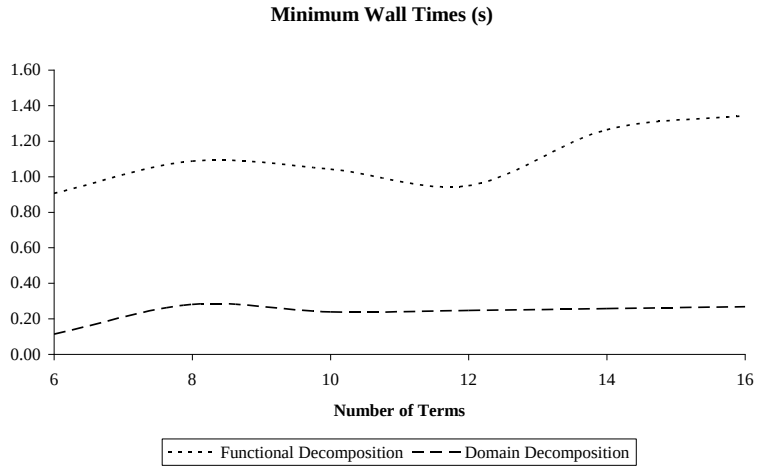

Figure 4.7 Minimum Wall Times for the Jacobi Polynomial Method

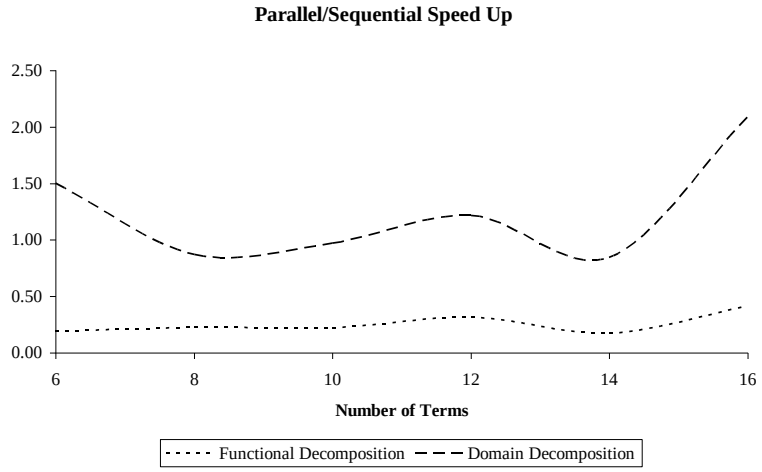

Figure 4.8 Parallel/Sequential Speed Up for the Jacobi Polynomial Method

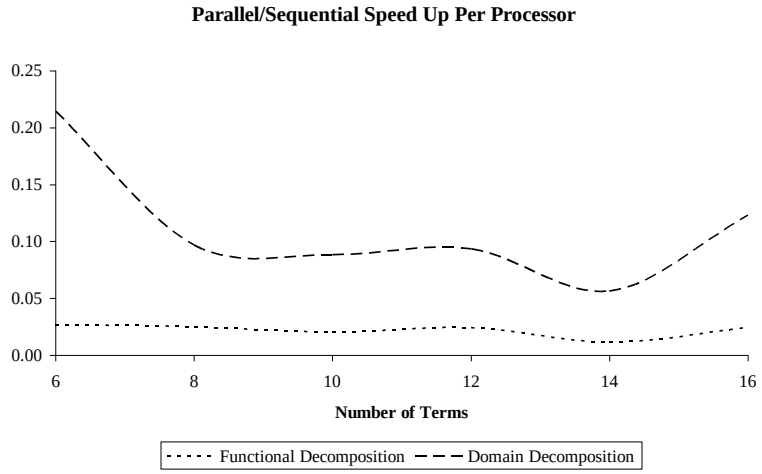

Figure 4.9 Parallel/Sequential Speed Up Per Processor for the Jacobi Polynomial Method

The Laguerre Polynomial Method

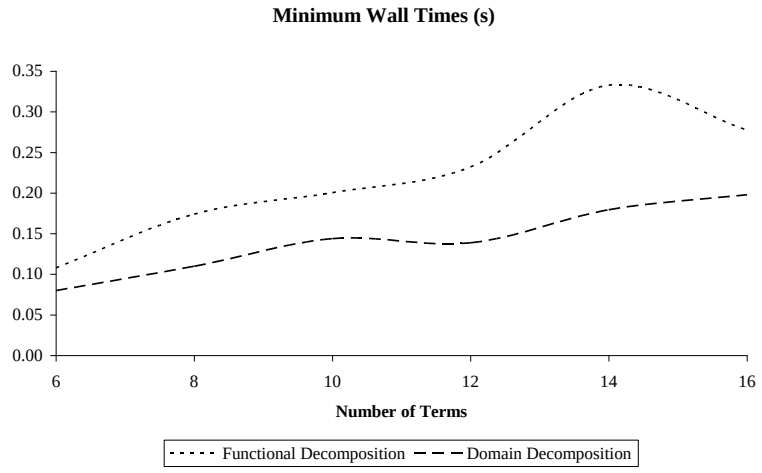

Figure 4.10 Minimum Wall Times for the Laguerre Polynomial Method

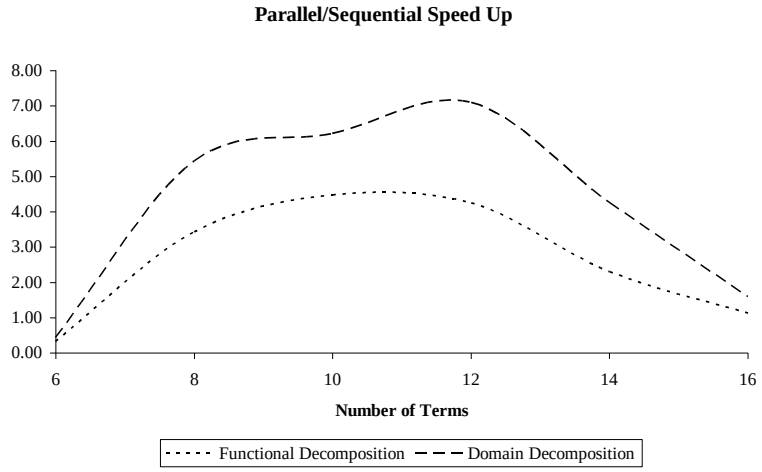

Figure 4.11 Parallel/Sequential Speed Up for the Laguerre Polynomial Method

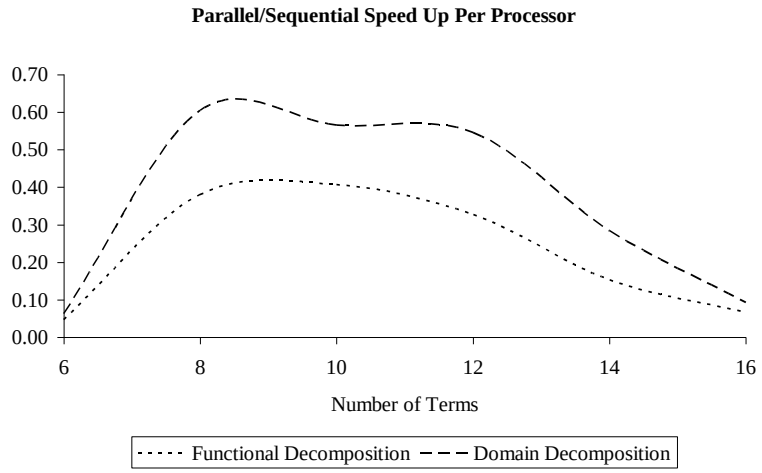

Figure 4.12 Parallel/Sequential Speed Up Per Processor for the Laguerre Polynomial Method

It can be seen from these graphs that in all cases, domain decomposition gives the fastest execution speeds, the largest parallel/sequential speed up and the largest parallel/sequential speed up per processor.

4.4.2.3 Conclusions

When numerically inverting Laplace transforms in a distributed computing environment domain decomposition is the preferred method to use. Domain decomposition was found to :

- minimise inter-processor communication. The only inter-processor communication required was the initial transmission from the master processor to the slave processors of the weight/term/interval data and the final transmission from the slave processors to the master processor of the parameters used in the calculation of the normalised root mean square deviation

- give the fastest execution speeds. Using domain decomposition much more work was performed in parallel than with the functional decomposition approach and hence a greater parallel/sequential speed up and parallel/sequential speed up per processor was obtained
- facilitate load balancing across the cluster. Barney (2010) defines load balancing as *the practice of distributing work among tasks so that all tasks are kept busy all of the time*. Load balancing is an important issue in parallel program design. If the workload is not spread evenly over the cluster then the fastest processors will have to wait *i.e.* idle, until the slowest one has completed. This means that the slowest processor will determine the overall performance of the program. Furthermore, this idling time is a waste of system resources. It is time that could be used for doing work, that is, time that could be used for reducing the program execution time. In the parallel programs developed in this research programme load balancing is achieved by allocating each slave processor an equal size part of the solution domain. Load balancing can be achieved using a functional decomposition. However, it is sometimes more difficult in this case.

For these reasons all future parallel algorithms developed and described in this dissertation will use a domain decomposition.

4.5 Parameter Values Used in the Numerical Inversion Algorithms

The parameter values used in the numerical inversion algorithms are within the ranges recommended by the authors cited earlier, and in the case of the Laguerre polynomial method, they are the optimal values calculated.

Algorithm	Parameter Values	Reference
Stehfest's Method	-	-
The SLP Method	$\kappa = 1$	Crann (2005)
The Jacobi Polynomial Method	$\beta = 5,\ \delta = 1$	Miller and Guy (1966)
The Laguerre Polynomial Method	$\alpha = 0,\ c = 2.4,\ N = 100,\ T = 0.1$	-

Table 4.5 Parameter Values Used in the Numerical Inversion Algorithms

4.6 Number of Weights/Terms and Processors Used

When solving one-dimensional, linear and nonlinear Black-Scholes equations :

- the number of weights/terms used in the numerical inversion algorithms is varied in the range 6(2)26 in both sequential and parallel programs. While the optimal number of weights/terms is being determined each parallel program uses 21 processors *i.e.* one master processor and twenty slave processors
- the number of processors used in the parallel programs is varied in the ranges 3(1)8 and 8(16)152[36]. While the optimal number of processors is being determined, the number of weights/terms used is fixed at the optimal number determined in the investigations above. This enables the optimal combination of weights/terms and processors for each numerical inversion algorithm to be determined. This is an aim of this research programme.

  An alternative approach is to fix the number of weights/terms used in each numerical inversion algorithm to the same value *e.g.* 6. In theory this should enable accurate speed comparisons to be made. However, an investigation conducted using the codes developed in Chapter 6 showed that although this approach produced minor differences in the data, the relative speeds of the numerical inversion algorithms remained the same. The likely reasons for this behaviour are explained in 10.1.1.

When solving two-dimensional, linear and nonlinear Black-Scholes equations the number of weights/terms used in the numerical inversion algorithms is again varied in the range 6(2)26 in both sequential and parallel programs. However, due to the nature of the two-dimensional algorithm, the number of processors used in the parallel version is always the number of values of the transform variable plus one (*i.e.* for the master processor).

For both the one-dimensional and the two-dimensional Black-Scholes equations, the ranges of weights/terms and processors given above capture the optimal performance data.

4.7 Chapter Summary

The methodology used in the investigations that follow has been described. In the next chapter, initial investigations will be conducted to determine whether Laplace transform based algorithms are effective when they are used to solve the one-dimensional, linear Black-Scholes equation.

4.8 Contribution to Knowledge

This chapter has established the most efficient decomposition method to use when numerically inverting Laplace transforms in a distributed computing environment.

[36] Multiples of eight are used because each node on the cluster contains eight processors.

Chapter 5

Laplace Transform Solutions - Initial Investigations

5.0 Introduction

In these initial investigations Laplace transform based algorithms are developed for solving the one-dimensional, linear Black-Scholes equation, firstly by using the Laplace transform of its analytical solution and secondly by solving its ODE BVP form.

Since this equation has an analytical solution *i.e.* expression (14), it is not necessary to use Laplace transform based algorithms to find its solution. However, algorithms of this type are developed here for three reasons. Firstly, to demonstrate the potential of Laplace transform based algorithms for solving problems of this type, secondly, to develop and evaluate parallel implementations of the numerical inversion algorithms described in Chapter 3 *i.e.* Stehfest's method, the shifted Legendre polynomial method, the Jacobi polynomial method and the Laguerre polynomial method and thirdly, to develop and evaluate numerical procedures that will be used within the more advanced algorithms described later in this dissertation.

5.1 The Solution Domain

By looking at the Black-Scholes equation (10) it can be seen that the value of an option V depends upon the current value of the underlying asset S and the time to expiry τ. To ensure that each computer program performs a significant amount of work, the equation is solved for a variety of S and τ values, that is, over an S-τ domain. In the parallel programs, each slave processor performs the calculations associated with a range of τ values.

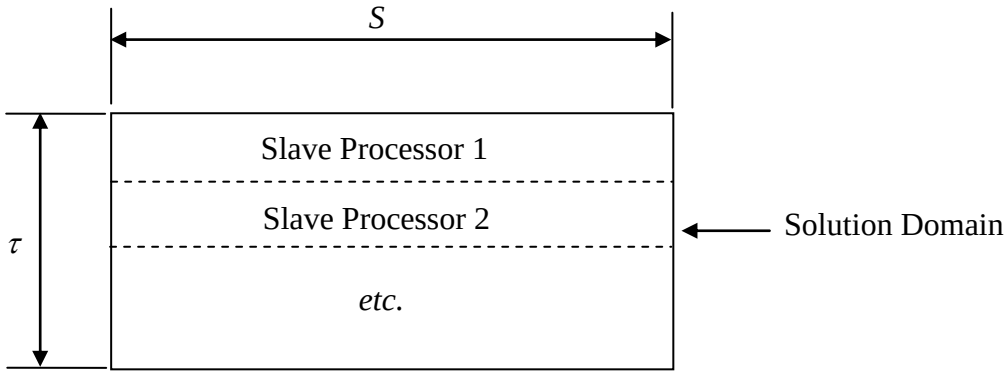

Figure 5.1 Domain Decomposition

5.2 Parameter Values

The parameter values used in the Black-Scholes equation (10) in this chapter are :

Parameter	Values
S	0.0(0.01)40.0
E	10
r	0.05
σ	0.25
τ	0.0(0.1)100.0[37]

Table 5.1 Parameter Values Used in the One-Dimensional, Linear Black-Scholes Equation

5.3 Investigation 1 : The Laplace Transform of the Analytical Solution

5.3.1 Aim

The aim of this investigation is to collect speed and accuracy data for sequential and parallel implementations of the numerical inversion algorithms when they are used to invert the Laplace transform of the analytical solution of the one-dimensional, linear Black-Scholes equation (10) and use it to compare their relative performances..

5.3.2 The Laplace Transform Formula

In Chapter 2 it was shown that the Black-Scholes equation (6) can be written as :

$$-\frac{\partial V}{\partial \tau}+\frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}+rS\frac{\partial V}{\partial S}-rV=0 \quad \text{---- (10)}$$

with initial condition :

$$V(S,0)=\max(S-E,0) \quad \text{---- (11)}$$

and boundary conditions :

$$V(0,\tau)=0 \quad \text{---- (12)} \quad \text{and} \quad V(S,\tau) \sim S \text{ as } S\to\infty \quad \text{---- (13)}$$

Taking Laplace transforms with respect to τ , the Black-Scholes equation (10) can be written as :

$$\frac{1}{2}\sigma^2 S^2\frac{d^2\bar{V}}{dS^2}+rS\frac{d\bar{V}}{dS}-(r+\lambda)\bar{V}=-V(S,0)$$

Substituting the initial condition $V(S,0)$ *i.e.* (11) :

[37] Since τ is measured in years, large values are unrealistic. However, large values are used because they produce execution times that allow accurate timing comparisons to be made. Furthermore, if a numerical algorithm is fast and accurate for a large value of τ , it will also be fast and accurate for a small value of τ .

$$\frac{1}{2}\sigma^2 S^2 \frac{d^2\bar{V}}{dS^2} + rS\frac{d\bar{V}}{dS} - (r+\lambda)\bar{V} = \begin{cases} -(S-E) & S > E \\ 0 & S \le E \end{cases} \quad \text{---- (37)}$$

Taking Laplace transforms with respect to τ, the boundary conditions (12) and (13) become :

$$\bar{V}(0;\lambda) = 0 \quad \text{---- (38)} \quad \bar{V}(S;\lambda) \to \frac{S}{\lambda} \text{ as } S \to \infty \quad \text{---- (39)}$$

In these equations $\bar{V}$ is the Laplace transform of V and λ is the transform variable. Equation (37) is the ODE BVP form of the one-dimensional, linear Black-Scholes equation.

Equation (37) can be solved using the analytical method, (Edwards and Penney 2008). The complementary function (CF) can be found using the trial solution $\bar{V} = CS^n$, $C, n \in \mathbb{R}$. Substituting this solution into the homogeneous form of equation (37) :

$$CS^n\left(\frac{1}{2}\sigma^2 n^2 + \left(r - \frac{1}{2}\sigma^2\right)n - (r+\lambda)\right) = 0$$

Dividing through by $\frac{1}{2}\sigma^2$, substituting $k = \frac{2r}{\sigma^2}$ and solving for n :

$$n = \frac{1-k}{2} \pm \sqrt{\frac{k+1}{2} + \frac{2\lambda}{\sigma^2}}$$

Hence :

$$\text{CF} = \begin{cases} C_1 S^{n_1} + C_2 S^{n_2} & S > E \\ C_3 S^{n_1} + C_4 S^{n_2} & S < E \end{cases}$$

where :

$$C_1, C_2, C_3, C_4 \in \mathbb{R}, \quad n_1 = \frac{1-k}{2} + \sqrt{\frac{k+1}{2} + \frac{2\lambda}{\sigma^2}}, \quad n_2 = \frac{1-k}{2} - \sqrt{\frac{k+1}{2} + \frac{2\lambda}{\sigma^2}} \quad \text{and} \quad k = \frac{2r}{\sigma^2}$$

When $S > E$ the particular integral (PI) has the general form :

$$\bar{V} = D_1 + D_2 S \text{ where } D_1, D_2 \in \mathbb{R}$$

Substituting this into equation (37) and equating coefficients :

$$\text{PI} = -\frac{E}{r+\lambda} + \frac{S}{\lambda}$$

When $S < E$, $\text{PI} = 0$.

Combining the complementary functions and particular integrals :

$$\bar{V} = \begin{cases} C_1 S^{n_i} + C_2 S^{n_2} - \dfrac{E}{r+\lambda} + \dfrac{S}{\lambda} & S > E \\ C_3 S^{n_1} + C_4 S^{n_2} & S < E \end{cases}$$

To satisfy boundary condition (39), the complementary function $C_1 S^{n_i} + C_2 S^{n_2}$ must tend to $\dfrac{E}{r+\lambda}$ as $S \to \infty$. Since $n_1 > 0$ and $n_2 < 0$, this will only happen if $C_1 = 0$.

To satisfy boundary condition (38), the complementary function $C_3 S^{n_1} + C_4 S^{n_2}$ must tend to zero as $S \to 0$. Since $n_1 > 0$ and $n_2 < 0$, this will only happen if $C_4 = 0$. Substituting these values :

$$\bar{V} = \begin{cases} C_2 S^{n_2} - \dfrac{E}{r+\lambda} + \dfrac{S}{\lambda} & S > E \\ C_3 S^{n_1} & S < E \end{cases} \quad \text{---- (40)}$$

The values of C_2 and C_3 can be found by considering the behaviour of $\bar{V}$ at the transition point *i.e.* the point where $S = E$. Here, the expressions in (40) must give the same value of $\bar{V}$ *i.e.*

$$C_2 S^{n_2} - \frac{E}{r+\lambda} + \frac{S}{\lambda} = C_3 S^{n_1}$$

Substituting $S = E$:

$$C_2 E^{n_2} - \frac{E}{r+\lambda} + \frac{E}{\lambda} = C_3 E^{n_1} \quad \text{---- (41)}$$

Since $\bar{V}$ varies continuously, the derivatives of the expressions in (40) with respect to *S* must also be equal *i.e.*

$$n_2 C_2 S^{n_2 - 1} + \frac{1}{\lambda} = n_1 C_3 S^{n_1 - 1}$$

Substituting $S = E$:

$$n_2 C_2 E^{n_2 - 1} + \frac{1}{\lambda} = n_1 C_3 E^{n_1 - 1} \quad \text{---- (42)}$$

Solving (41) and (42) simultaneously :

$$\left.\begin{aligned} C_2 &= \left[\frac{n_1}{r+\lambda} - \frac{n_1 - 1}{\lambda}\right] \frac{E^{1-n_2}}{(n_1 - n_2)} \\ C_3 &= \left[\frac{n_2}{r+\lambda} + \frac{1-n_2}{\lambda}\right] \frac{E^{1-n_1}}{(n_1 - n_2)} \end{aligned}\right\} \quad \text{---- (43)}$$

Substituting the expressions for C_2 and C_3 in (43) into (40), the Laplace transform of the analytical solution of the Black-Scholes equation (10) becomes :

$$\bar{V} = \begin{cases} \left[\frac{n_1}{r+\lambda} - \frac{n_1 - 1}{\lambda}\right] \frac{E^{1-n_2}}{(n_1 - n_2)} S^{n_2} - \frac{E}{r+\lambda} + \frac{S}{\lambda} & S > E \\ \left[\frac{n_2}{r+\lambda} + \frac{1-n_2}{\lambda}\right] \frac{E^{1-n_1}}{(n_1 - n_2)} S^{n_1} & S < E \end{cases} \quad \text{---- (44)}$$

where $n_1 = \frac{1-k}{2} + \sqrt{\frac{k+1}{2} + \frac{2\lambda}{\sigma^2}}$, $n_2 = \frac{1-k}{2} - \sqrt{\frac{k+1}{2} + \frac{2\lambda}{\sigma^2}}$ and $k = \frac{2r}{\sigma^2}$

By looking at the initial condition (11) it can be seen that when $S = E$, $V = 0$ and hence $\bar{V} = 0$.

5.3.3 Performance Data

The graphs and tables below provide a summary of the data collected during this investigation. The graphs show visually, the relative performances of the numerical inversion algorithms. As before, only the graphs showing the minimum wall times are included. Detailed results are given in Appendix B.

5.3.3.1 Optimal Sequential Programs Data

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Value
NRMSD :	Stehfest	6	0.00760937300
Minimum Wall Time (s) :	Stehfest	6	7.40825700800

Table 5.2 Optimal Sequential Programs Data (Analytical LT)

5.3.3.2 Part 1 - Varying the Number of Weights/Terms Used

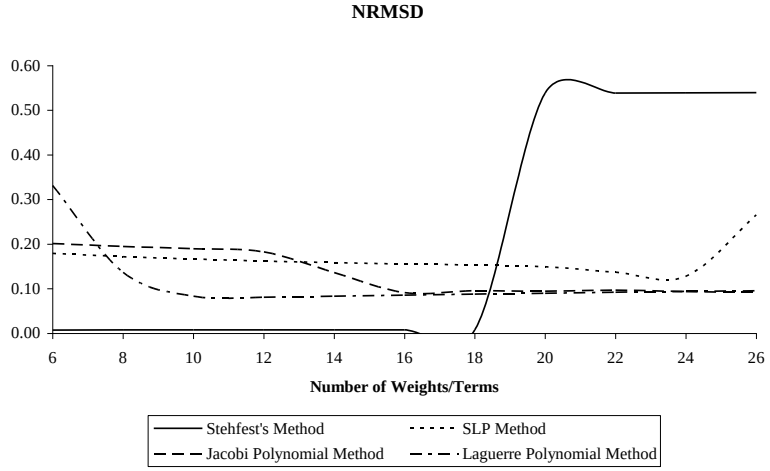

Figure 5.2 Normalised Root Mean Square Deviation, Parallel Programs (Analytical LT)

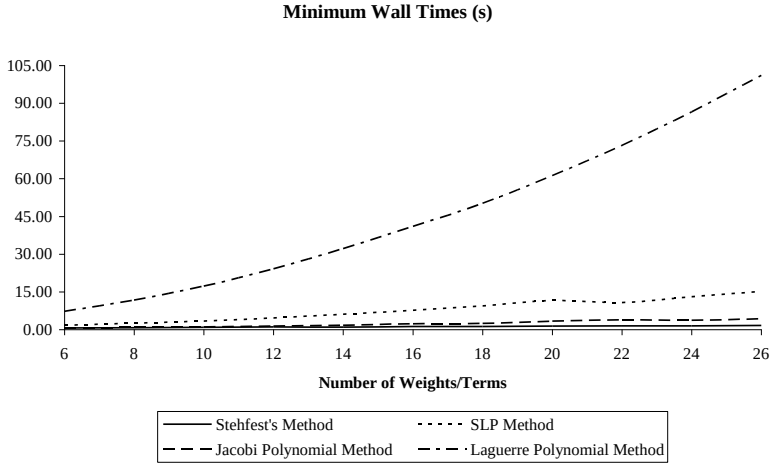

Figure 5.3 Minimum Wall Times, Parallel Programs (Analytical LT)[38]

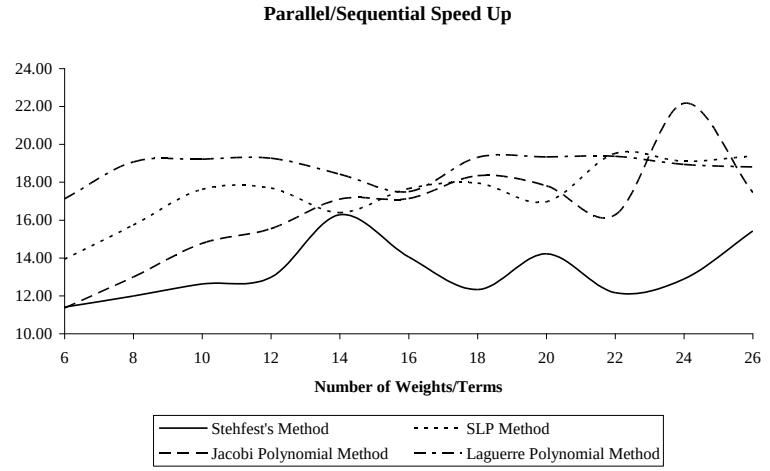

Figure 5.4 Parallel/Sequential Speed Up (Analytical LT)

[38] The minimum wall times for Stehfest's method and the Jacobi polynomial method follow slight upward trends.

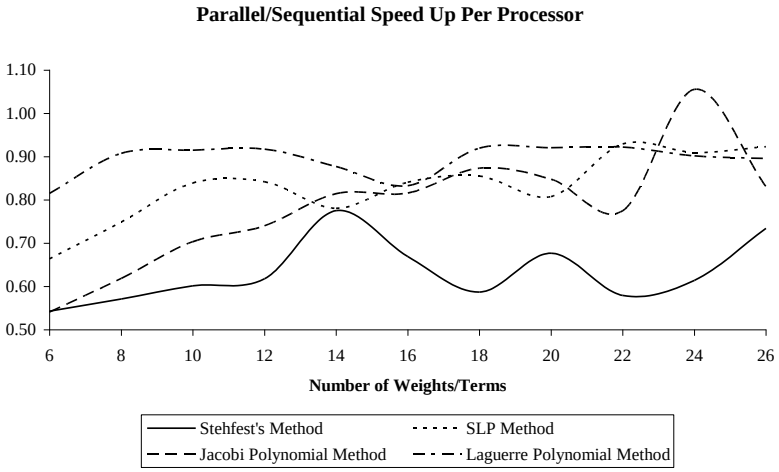

Figure 5.5 Parallel/Sequential Speed Up Per Processor (Analytical LT)

5.3.3.3 Part 2 - Varying the Number of Processors Used

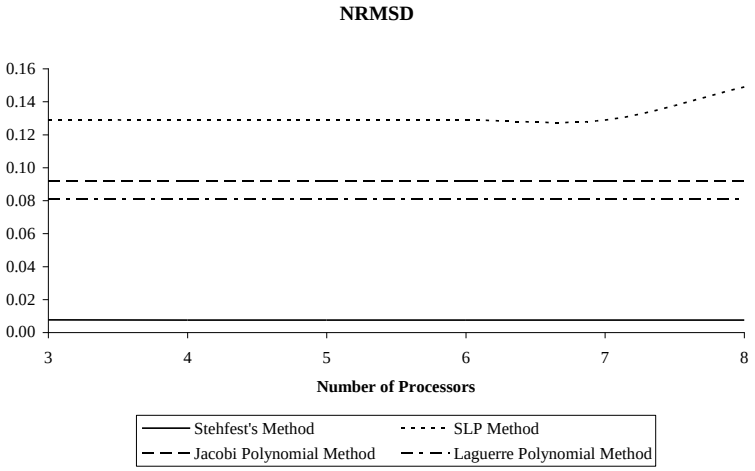

Figure 5.6 Normalised Root Mean Square Deviation (Analytical LT) (3-8 Processors)[39]

[39] The NRMSD values for Stehfest's method, the Jacobi polynomial method and the Laguerre polynomial method follow slight oscillating trends.

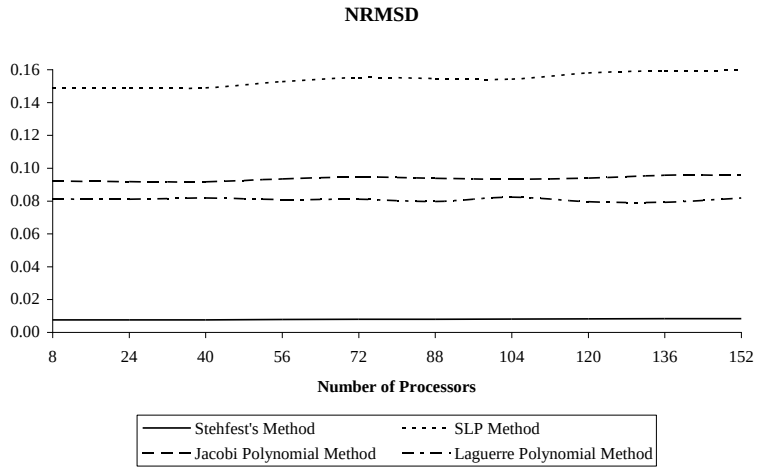

Figure 5.7 Normalised Root Mean Square Deviation (Analytical LT) (8-152 Processors)[40]

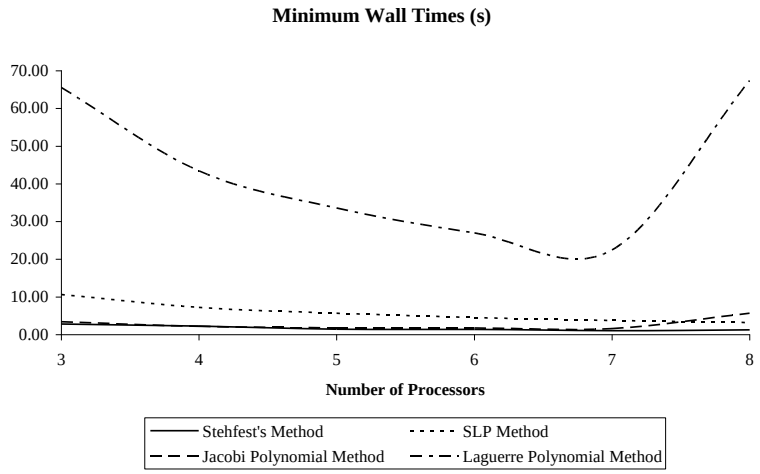

Figure 5.8 Minimum Wall Times (Analytical LT) (3-8 Processors)[41]

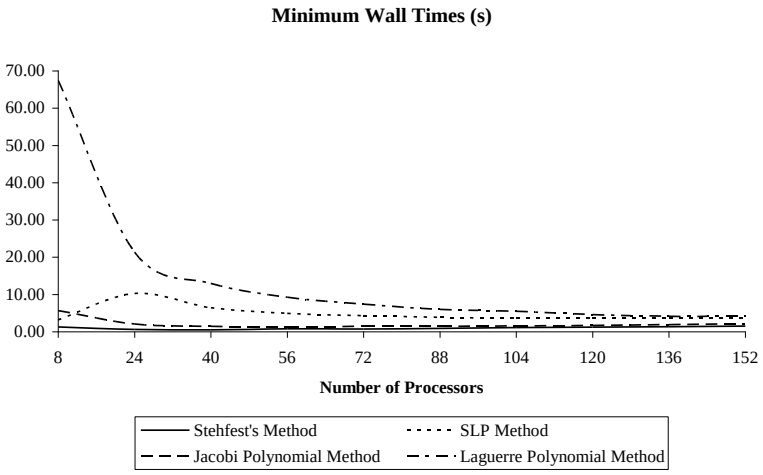

Figure 5.9 Minimum Wall Times (Analytical LT) (8-152 Processors)[42]

40 The NRMSD values for Stehfest's method follow a slight oscillating trend.

41 The minimum wall times for Stehfest's method and the Jacobi polynomial method follow oscillating trends.

42 The minimum wall times for Stehfest's method follow an oscillating trend.

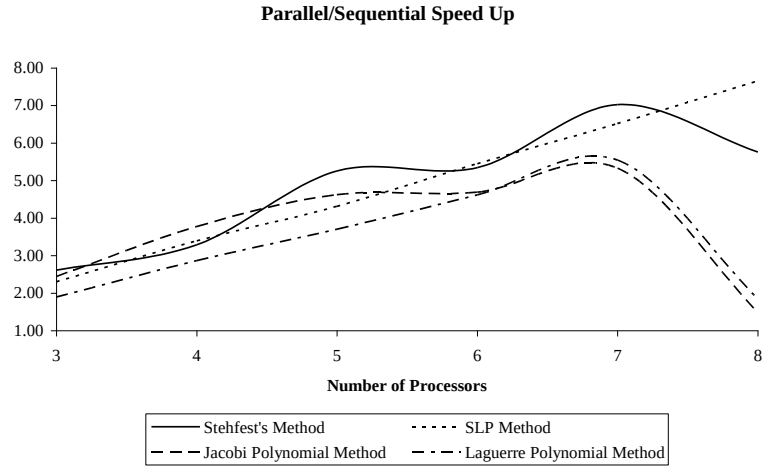

Figure 5.10 Parallel/Sequential Speed Up (Analytical LT) (3-8 Processors)

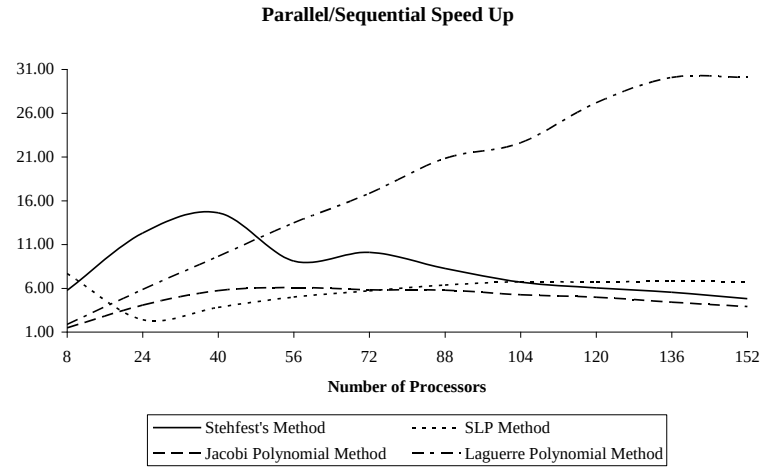

Figure 5.11 Parallel/Sequential Speed Up (Analytical LT) (8-152 Processors)

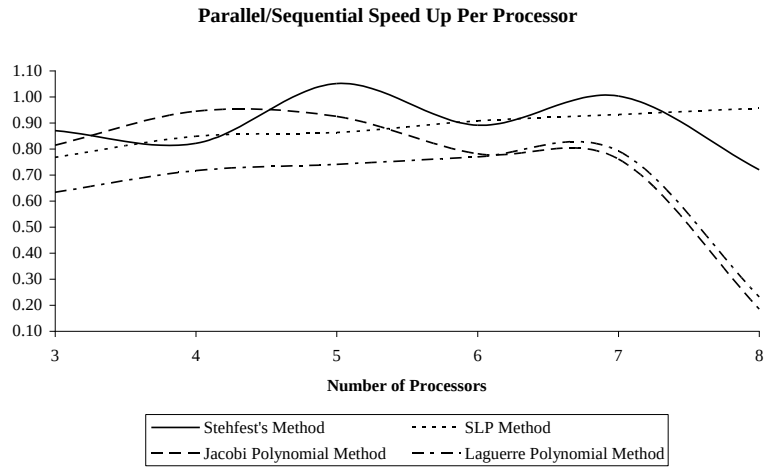

Figure 5.12 Parallel/Sequential Speed Up Per Processor (Analytical LT) (3-8 Processors)

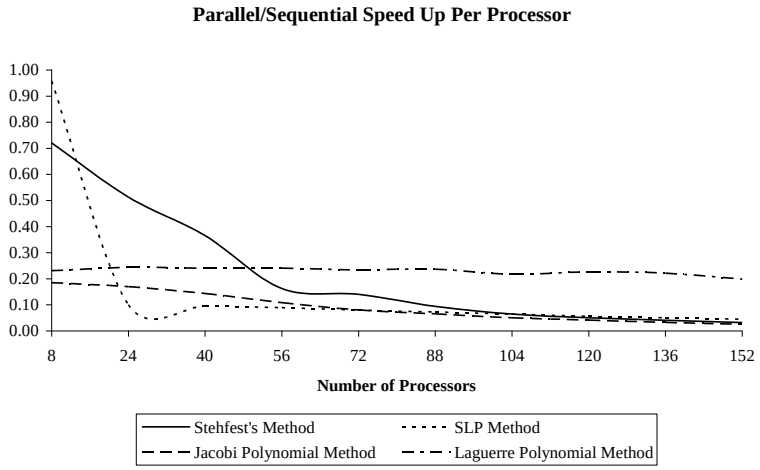

Figure 5.13 Parallel/Sequential Speed Up Per Processor (Analytical LT) (8-152 Processors)[43]

5.3.3.4 Optimal Parallel Programs Data

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Processors	Optimal Value
NRMSD :	Stehfest	6	21	0.00753181310
Minimum Wall Time (s) :	Stehfest	6	40	0.50650691880
Parallel/Sequential Speed Up :	Laguerre	6	136	30.08960574915
Parallel/Sequential Speed Up/Processor :	Stehfest	6	5	1.05105894946

Table 5.3 Optimal Parallel Programs Data (Analytical LT)

5.4 Investigation 2 : The Laplace Transforms Arising in the ODE BVP Form

5.4.1 Aim

The aim of this investigation is to collect speed and accuracy data for sequential and parallel implementations of the numerical inversion algorithms when they are used to invert the Laplace transforms arising in the finite difference solution of the ODE BVP form of the one-dimensional, linear Black-Scholes equation (10) and use it to compare their relative performances.

5.4.2 The Finite Difference Solution of the ODE BVP Form

In 5.3.2 it was shown that the Black-Scholes equation (10) can be written as :

$$\frac{1}{2}\sigma^2 S^2 \frac{d^2\bar{V}}{dS^2} + rS\frac{d\bar{V}}{dS} - (r+\lambda)\bar{V} = \begin{cases} -(S-E) & S > E \\ 0 & S \le E \end{cases} \quad \text{---- (37)}$$

with boundary conditions :

$$\bar{V}(0;\lambda) = 0 \quad \text{---- (38)} \quad \bar{V}(S;\lambda) \sim \frac{S}{\lambda} \text{ as } S \to \infty \quad \text{---- (39)}$$

[43] The parallel/sequential speed up per processor values for the Laguerre polynomial method follow a slight oscillating trend.

An alternative way to solve the Black-Scholes equation (10) is to solve the ODE BVP form (37) using an appropriate numerical method and then to invert the $\bar{V}$ values obtained using a numerical inversion algorithm. Crann *et al.* (1998) used this approach and solved the ODE BVP form using the finite volume method. In this investigation the ODE BVP form is solved using the finite difference method, that is, by replacing the derivatives with central difference approximations. Smith (2004) shows that if *h* is the step length in the *x*-direction then :

$$\frac{dy}{dx} \approx \frac{y_{i+1} - y_{i-1}}{2h} \quad \text{and} \quad \frac{d^2y}{dx^2} \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

Applying these results to equation (37) :

$$\frac{1}{2}\sigma^2 S_i^{\,2}\left(\frac{\bar{V}_{i+1} - 2\bar{V}_i + \bar{V}_{i-1}}{h^2}\right) + rS_i\left(\frac{\bar{V}_{i+1} - \bar{V}_{i-1}}{2h}\right) - (r+\lambda)\bar{V}_i = \begin{cases} -(S_i - E) & S > E \\ 0 & S \le E \end{cases}$$

$$\left(\sigma^2 S_i^2 - hrS_i\right)\bar{V}_{i-1} - 2\left(\sigma^2 S_i^2 + h^2(r+\lambda)\right)\bar{V}_i + \left(\sigma^2 S_i^2 + hrS_i\right)\bar{V}_{i+1} = \begin{cases} -2h^2(S_i - E) & S > E \\ 0 & S \le E \end{cases}$$

where *h* is now the step length in the *S*-direction. This expression produces a tridiagonal system of linear equations for each value of λ. To solve the Black-Scholes equation (10) using this method the λ values used in the numerical inversion algorithm are calculated and the corresponding systems of linear equation are formed and solved. The numerical inversion algorithm is then applied to the solutions. Each system is stored within a compact storage scheme. Only the leading diagonal, the principal sub-diagonal, the principal super-diagonal and the right-hand side vector are stored. Following (Chapra and Canale 2010) we solve the tridiagonal system using the Thomas algorithm, which is a computationally efficient method for solving the tridiagonal systems arising from diffusion equations.[44]

5.4.3 Performance Data

The graphs and tables below provide a summary of the data collected during this investigation. Once again, the graphs show visually, the relative performances of the numerical inversion algorithms. As before, the minimum wall time is considered to be the most accurate measure of absolute speed. Detailed results are given in Appendix B.

[44] An alternative, more general approach would be to use the routine DGTSV from LAPACK (http://www.netlib.org/lapack/)

5.4.3.1 Optimal Sequential Programs Data

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Value
NRMSD :	Stehfest	6	0.04177589251
Minimum Wall Time (s) :	Stehfest	6	2.13941407200

Table 5.4 Optimal Sequential Programs Data (BVP LT)

5.4.3.2 Part 1 - Varying the Number of Weights/Terms Used

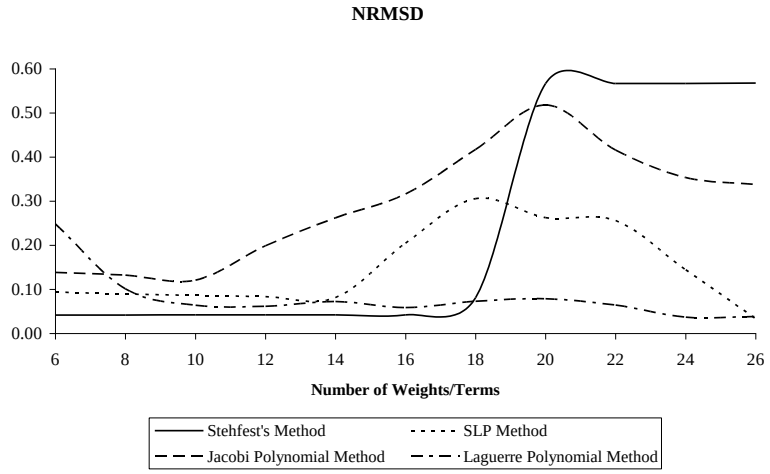

Figure 5.14 Normalised Root Mean Square Deviation, Parallel Programs (BVP LT)

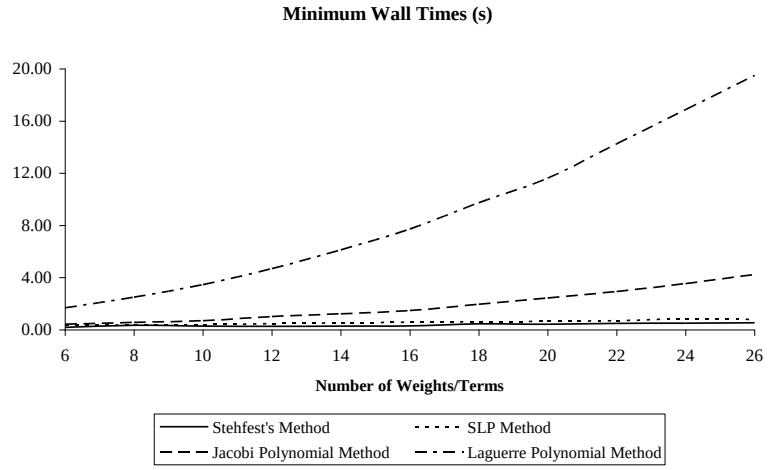

Figure 5.15 Minimum Wall Times, Parallel Programs (BVP LT)[45]

[45] The minimum wall times for Stehfest's method and the SLP method follow increasing trends.

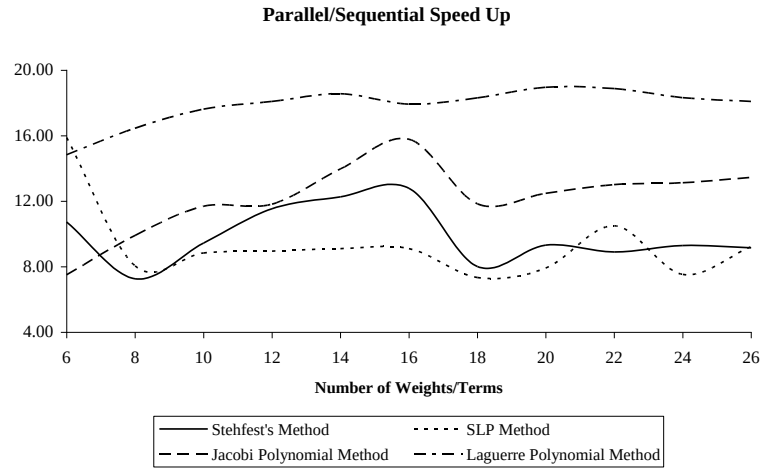

Figure 5.16 Parallel/Sequential Speed Up (BVP LT)

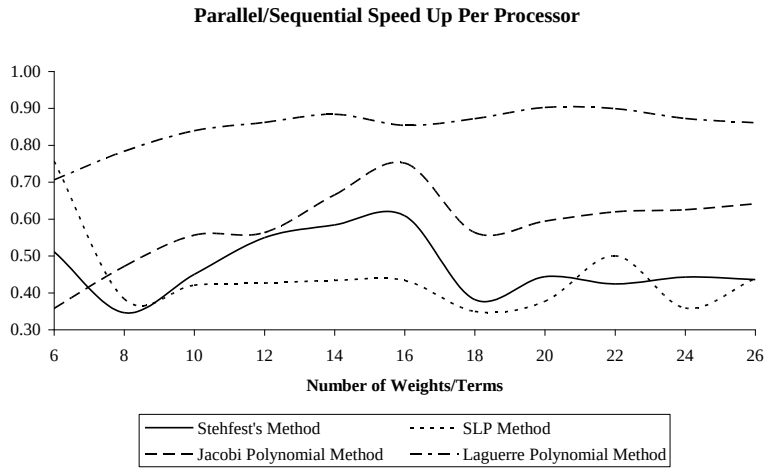

Figure 5.17 Parallel/Sequential Speed Up Per Processor (BVP LT)

5.4.3.3 Part 2 - Varying the Number of Processors Used

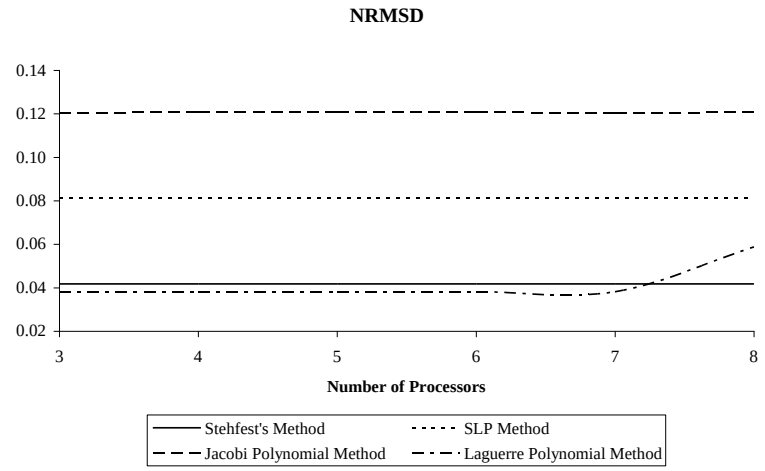

Figure 5.18 Normalised Root Mean Square Deviation (BVP LT) (3-8 Processors)[46]

[46] The NRMSD values for Stehfest's method, the SLP method and the Jacobi polynomial method follow slight oscillating trends.

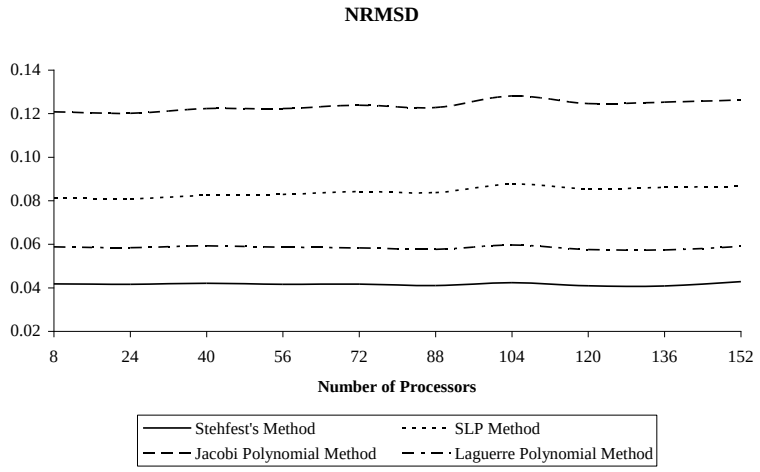

Figure 5.19 Normalised Root Mean Square Deviation (BVP LT) (8-152 Processors)

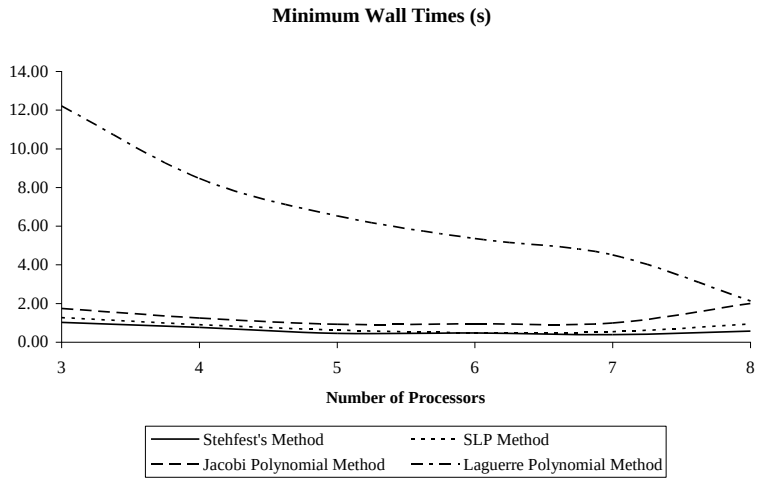

Figure 5.20 Minimum Wall Times (BVP LT) (3-8 Processors)

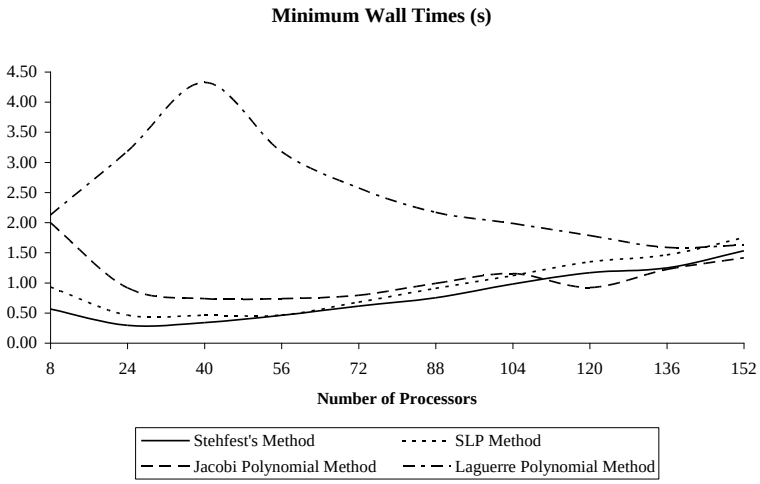

Figure 5.21 Minimum Wall Times (BVP LT) (8-152 Processors)

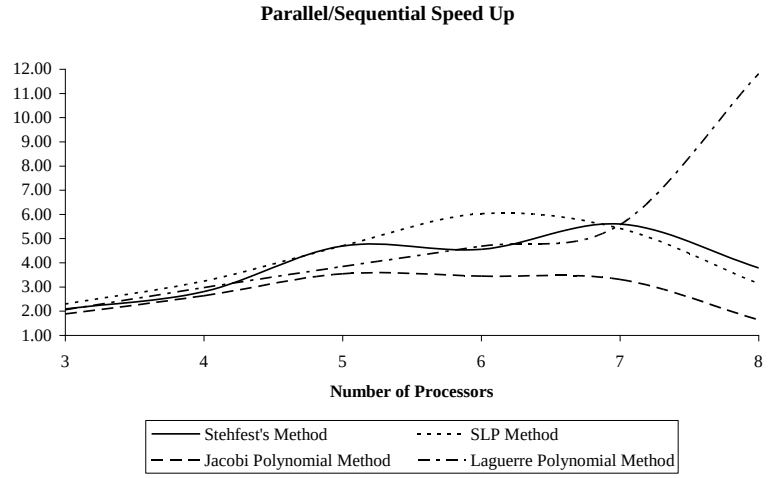

Figure 5.22 Parallel/Sequential Speed Up (BVP LT) (3-8 Processors)

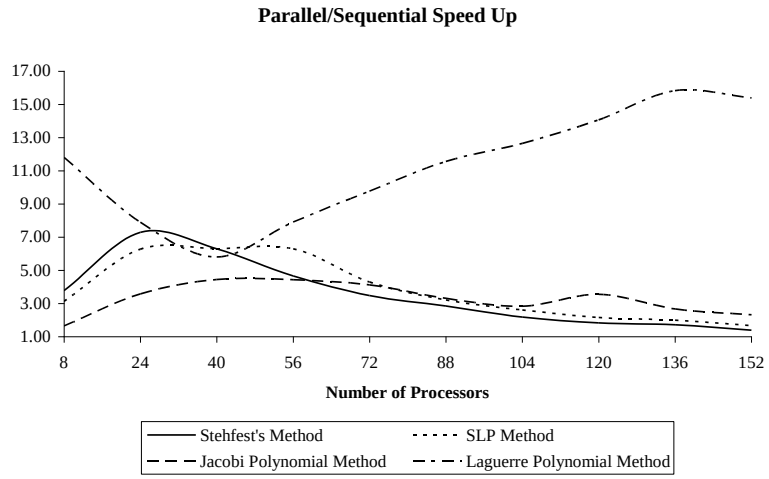

Figure 5.23 Parallel/Sequential Speed Up (BVP LT) (8-152 Processors)

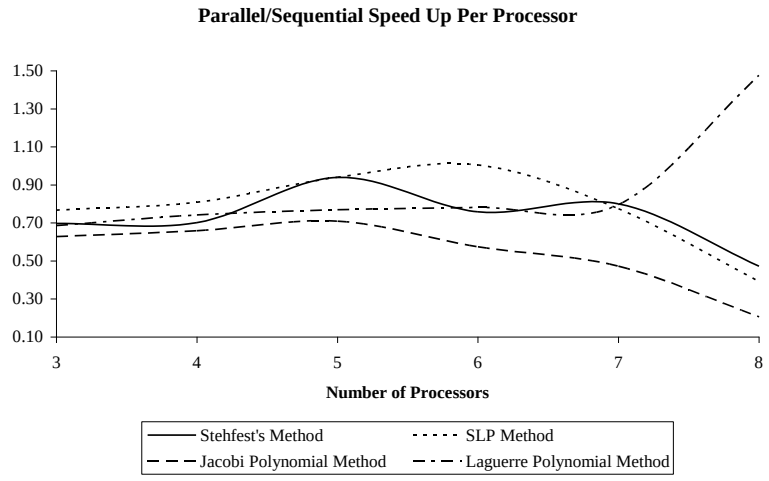

Figure 5.24 Parallel/Sequential Speed Up Per Processor (BVP LT) (3-8 Processors)

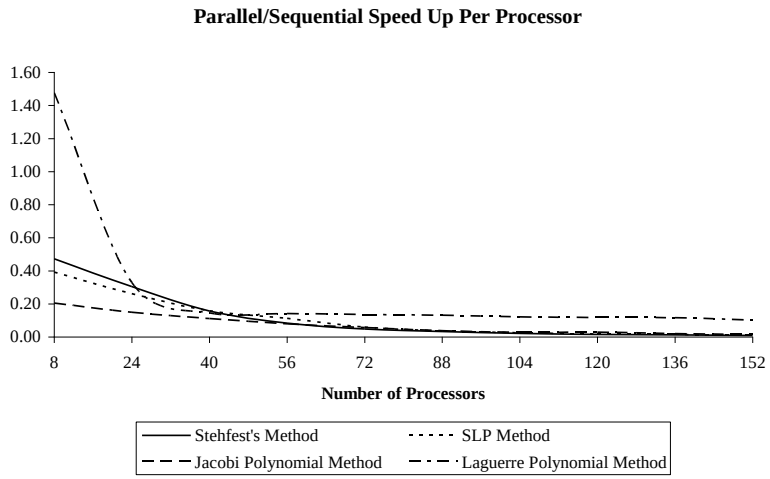

Figure 5.25 Parallel/Sequential Speed Up Per Processor (BVP LT) (8-152 Processors)[47]

5.4.3.4 Optimal Parallel Programs Data

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Processors	Optimal Value
NRMSD :	SLP	26	21	0.03302565905
Minimum Wall Time (s) :	Stehfest	6	21	0.19930100440
Parallel/Sequential Speed Up :	Laguerre	20	21	18.95694137668
Parallel/Sequential Speed Up/Processor :	Laguerre	6	8	1.47589113525

Table 5.5 Optimal Parallel Programs Data (BVP LT)

5.5 Conclusions

Tables 5.2, 5.3, 5.4 and 5.5 above give the optimal number of weights/terms and processors to use in/with sequential and parallel programs for solving the one-dimensional, linear Black-Scholes equation (10) using each of the Laplace transform based approaches considered. In these initial investigations :

- the parallel programs are faster and more accurate than the corresponding sequential programs. However, the differences in accuracy are negligible
- the most accurate way to solve the Black-Scholes equation (10) using the Laplace transform method is to use the Laplace transform of the analytical solution approach
- the fastest way to solve the Black-Scholes equation (10) using the Laplace transform method is to use the ODE BVP approach
- the *best* all-round numerical inversion algorithm is Stehfest's method. The Laguerre polynomial method only features in the parallel/sequential speed up categories because it

[47] The parallel/sequential speed up per processor values for all numerical inversion algorithms follow decreasing trends.

is significantly slower than the other numerical inversion algorithms tested.

Speed and accuracy are equally important when solving mathematical problems using numerical methods. However, since the ODE BVP approach can be used for solving both linear and nonlinear equations, this will be the method used in the parallel, Laplace transform based algorithms described in Chapter 6 and Chapter 7.

5.6 Chapter Summary

These initial investigations have shown that despite the disadvantages of Laplace transform based algorithms described in Chapter 3, methods of this type can be used for solving the one-dimensional, linear Black-Scholes equation (10). The next chapter will develop and evaluate a parallel algorithm for solving this equation that combines Laplace transform methods with finite difference techniques.

5.7 Contribution to Knowledge

This chapter has :

- shown that Laplace transform based algorithms can produce fast and accurate solutions of the one-dimensional, linear Black-Scholes equation (10)
- evaluated parallel implementations of Stehfest's method, the shifted Legendre polynomial method, the Jacobi polynomial method and the Laguerre polynomial method when these algorithms are used to invert the Laplace transforms arising in the solution of the one-dimensional, linear Black-Scholes equation (10)
- determined the optimal number of weights/terms and processors to use with the numerical inversion algorithms in this financial context. The optimal values are the ones that give the most accurate solution (*i.e.* the smallest NRMSD value) and the fastest solution (*i.e.* the minimum program wall time, the largest parallel/sequential speed up and the largest parallel/sequential speed up per processor).

Chapter 6

The One-Dimensional, Laplace Transform-Finite Difference Algorithm

6.0 Introduction

One-dimensional, linear Black-Scholes equations are often solved using finite difference methods. The solutions along the first time row are given by the initial condition of the equation. The solutions along the remaining time rows are then calculated using a time-marching algorithm. The following investigation will show how Laplace transform methods can be incorporated into this procedure to produce a parallel algorithm that will find the solutions as accurately as its sequential counterpart but much more quickly.

Since one-dimensional, linear Black-Scholes equations can be solved using the analytical solution (14), it is not necessary to solve these equations using numerical methods[48]. A parallel, Laplace transform-finite difference (LTFD) algorithm is developed and evaluated here :

- to demonstrate the potential of the Laplace transform based approach
- as the first stage in the development of an algorithm that can be used for solving one-dimensional, nonlinear Black-Scholes equations *i.e.* Black-Scholes equations for which very few analytical solutions exist
- because some nonlinear Black-Scholes equations become linear under the financial assumptions described in 2.3.2 and 2.3.3
- to create a fast and robust[49] algorithm for those who need to use numerical methods to solve diffusion equations in other application areas[50]. This will the case when no analytical solutions of these equations exist.

6.1 The Algorithm

Using the Laplace transform method the time domain of a one-dimensional, parabolic PDE can decomposed so that the problem can be solved using parallel computing methods,

---

[48] Numerical methods are used for simplicity. The analytical solution (14) contains the complementary error function. Some computing environments do not provide a built-in procedure for evaluating this function.

[49] A numerical algorithm is said to be robust if it produces accurate solutions for a variety of parameter values and these solutions are reproducible.

[50] See Chapter 1 for possible application areas.

(Crann *et al.* 2007). The solutions along the first row of the first sub-domain (*i.e.* the first time row) can be found using the initial condition of the equation. The solutions along the first row of the second and all following sub-domains can be found using the Laplace transform method. The solutions within each sub-domain can be found using a finite difference method *i.e.*

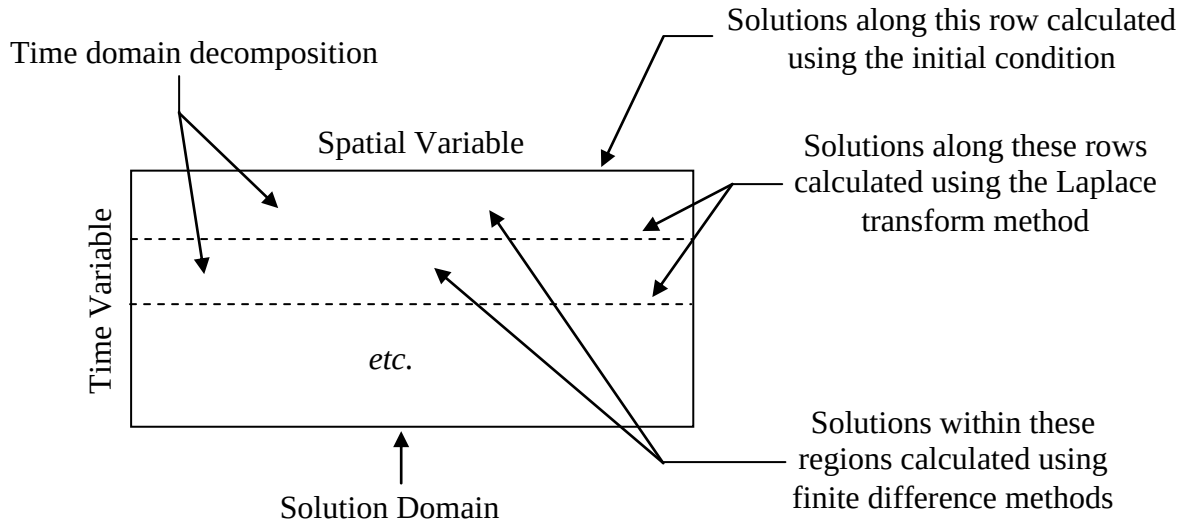

Figure 6.1 The Laplace Transform-Finite Difference Algorithm

Domain decomposition based algorithms of this type have been developed previously. For example, Tagliani and Milev (2012), Wang *et al.* (2009), Natkunam (2009), Lai *et al*. (2005) and Chen and Lin (1991) have developed similar Laplace transform and finite difference based algorithms[51]. Crann *et al.* (2007) and Davies *et al.* (2007) have developed algorithms that use Laplace transform methods for finding the initial values required and then the finite volume method or the boundary element method for finding the solutions within each sub-domain. Another well known domain decomposition based algorithm is the predictor-corrector version of the Parareal[52] algorithm. This was developed by Bal and Maday and described in their 2002 paper. This algorithm uses an iterative finite difference procedure for finding the initial values required[53] and then uses a time-marching finite difference algorithm

[51] The main difference between these algorithms and the LTFD algorithm presented here is that the authors have not applied finite difference methods to the dimensionless, forward diffusion equation form of the PDE being solved. Hence, they have not used a computational procedure like the one described in 6.2.2.

[52] Parallel in time.

[53] In outline, the procedure is as follows. Firstly, the initial values are *predicted* by solving the PDE over the entire solution domain using a finite difference method with a relatively large time step *i.e.* using a coarse grid. These calculations must be performed sequentially. Secondly, the initial values are *corrected* by solving the PDE over part of each sub-domain using a finite difference method with a relatively small time step *i.e*. using a fine grid. The predicted values are used as initial conditions. These calculations are performed in parallel. If necessary, the *corrector* stage can be repeated iteratively until the initial values are sufficiently accurate.

for finding the solutions within each sub-domain.

This research programme extends previous work in this area by using the Laplace transform-finite difference approach to solve a variety of one-dimensional, linear and nonlinear Black-Scholes equations.

6.2 The Diffusion Equation Form

When solving the one-dimensional, linear Black-Scholes equation (10) using finite difference methods it is usual to apply the numerical algorithms to the diffusion equation form. Wilmot *et al.* (1999) state that this is the simplest way to solve these equations.

In Chapter 2 it was shown that the one-dimensional, backward Black-Scholes equation is :

$$\frac{\partial V}{\partial t}+\frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}+rS\frac{\partial V}{\partial S}-rV=0 \quad \text{---- (6)}$$

with the final condition :

$$V(S,T)=\max(S-E,0) \quad \text{---- (7)}$$

and the boundary conditions :

$$V(0,t)=0 \quad \text{---- (8)} \qquad V(S,t)\approx S \text{ as } S\to\infty \quad \text{---- (9)}$$

Wilmot *et al.* (1999) show that by using the changes of variable :

$$S=Ee^{x} \quad \text{---- (45)} \qquad t=T-\frac{2\tau}{\sigma^2} \quad \text{---- (46)} \qquad V=Ee^{\alpha x+\beta\tau}u(x,\tau) \quad \text{---- (47)}$$

this problem can be written as the dimensionless[54], forward diffusion equation :

$$\frac{\partial^2 u}{\partial x^2}=\frac{\partial u}{\partial \tau} \qquad -\infty<x<\infty,\ \ \tau>0 \quad \text{---- (48)}$$

with the initial condition :

$$u(x,0)=\max\left(e^{(1-\alpha)x}-e^{-\alpha x},0\right) \quad \text{---- (49)}$$

and the boundary conditions :

$$u(x,\tau)\to 0 \text{ as } x\to-\infty \quad \text{---- (50)} \qquad u(x,\tau)\to e^{(1-\alpha)x-\beta\tau} \text{ as } x\to\infty \quad \text{---- (51)}$$

Here, *S*, *E*, *t*, *T*, *r* and σ are as before, τ is now the non-dimensional time to expiry,

---

[54] Non-dimensionalisation is the partial or full removal of the units from an equation involving physical quantities using suitable changes of variable. The technique is used to simplify and parameterise problems. Once the simplified problem has been solved, the values of the original variables can be recovered using the changes of variable used.

$\alpha = -\frac{1}{2}(k-1),\ \beta = -\frac{1}{4}(k+1)^2$ and $k = \frac{2r}{\sigma^2}$.

6.2.1 The Computational Procedure

Using the diffusion equation form, the solution of the forward Black-Scholes equation (10) *i.e.* the value of $V(S,t)$ for particular values of *S* and *t* can be found by :

- calculating the corresponding τ -value using (46) *i.e.* $\tau = \frac{\sigma^2}{2}(t-T)$
- solving equation (48) for $u(x,\tau)$. This can be done by choosing step lengths δx and $\delta\tau$, choosing large values of N^- and N^+ to simulate the infinite spatial domain and then solving equation (48) over the region $\left[N^-\delta x, N^+\delta x\right] \mathrm{x} [0,\tau]$. The *u*-values in the first row of this region can be calculated using the initial condition (49). The *u*-values at the ends of this region can be calculated using the boundary conditions (50) and (51). The *u*-values in the remaining part of this region can be calculated using a finite difference method
- converting the *x*-values into *S*-values using (45) *i.e.* $S = Ee^x$
- converting the $u(x,\tau)$ values into *V*-values using (47) *i.e.* $V = Ee^{\alpha x+\beta\tau}u(x,\tau)$
- calculating $V(S,t)$ by interpolation. The most accurate way to do this is to fit a cubic spline to the *S* and *V*-values and to interpolate the required value using this.

6.2.2 An Improved Procedure

A practical difficulty with this algorithm is choosing appropriate values for N^- and N^+. These values must be sufficiently large to ensure that the required *x*-value (*i.e. S*-value) is within the solution domain. However, choosing large values for N^- and N^+ increases the amount of calculations that must be performed. During this investigation it was discovered that a better algorithm for calculating $V(S,t)$ is to :

- calculate the corresponding τ -value using (46) *i.e.* $\tau = \frac{\sigma^2}{2}(t-T)$
- calculate the corresponding *x*-value using (45) *i.e.* $x = \ln\left(\frac{S}{E}\right)$
- solve equation (48) over a region $\left[N^-\delta x, N^+\delta x\right] \mathrm{x} [0,\tau]$ as before. The difference here is that the x-value is placed in the centre of this region

- calculate $V(S,t)$ using (47) *i.e.* $V(S,t) = Ee^{\alpha x+\beta\tau}u(x,\tau)$.

The advantages of this modified procedure are that since the x-value is guaranteed to be within the solution domain, smaller values of N^- and N^+ can be used. This increases the computation speed and reduces the storage requirements. Furthermore, since the exact position of the *x*-value (*i.e. S*-value) is known, there is no need to calculate $V(S,t)$ using interpolation. This modified procedure is the one that will be used.

6.3 Finite Difference Methods

Significant effort has been put into developing sophisticated finite difference schemes that can be used for solving diffusion equations. Interested readers should consult references such as Düring *et al.* (2012), Hirsa (2012), Wang *et al.* (2011), Jeong *et al.* (2009), Liao *et al.* (2001), Buetow and Sochacki (2000), Wood (1990) and Hull and White (1990). However, the question arises, *do these methods offer significant advantages over simple finite difference schemes* ?

To support our thesis that good results may be obtained using Laplace transforms we choose the simplest scheme available to us, the explicit method.

In the explicit method the spatial derivative is replaced with a central difference approximation and the time derivative is replaced with a forward difference approximation. Hence, the diffusion equation (48) is approximated at the point (i, j) by :

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\delta x)^2} = \frac{u_{i,j+1} - u_{i,j}}{\delta\tau}$$

This can be transposed to give :

$$u_{i,j+1} = Ru_{i-1,j} + (1-2R)u_{i,j} + Ru_{i+1,j}$$

where $R = \frac{\delta\tau}{(\delta x)^2}$ is the mesh ratio. Smith (2004) shows that the explicit method is stable[55] only for values of *R* in the range $0 < R \leq 0.5$.

For very high accuracy a small spatial step may be needed and this may impose a very severe restriction on the time step. If this were prohibitive it would then be appropriate to reconsider the use of implicit methods.

6.4 Solving One-Dimensional, Linear Black-Scholes Equations

[55] A numerical method is said to be stable if a small change in the data produces a small change in the solution.

6.4.1 Parameter Values

The parameter values used in the Black-Scholes equation (10) are :

Parameter	Values
S	0.0(0.1)40.0
E	10
r	0.05
σ	0.25
τ	0.0(0.1)100.0

Table 6.1 Parameter Values Used in the One-Dimensional, Linear Black-Scholes Equation

6.4.2 Aim

The aim of this investigation is to collect speed and accuracy data for sequential and parallel algorithms for solving the one-dimensional, linear Black-Scholes equation (10) and use it to compare their relative performances.

6.4.3 Preliminary Notes

When using the LTFD approach to solve the Black-Scholes equation (10) :

- the solutions along the first row of the first sub-domain (*i.e.* the first time row) are found using the initial condition of the equation

$$V(S,0) = \max(S-E,0)$$

- the solutions along the first row of the second and all following sub-domains are found using the Laplace transform method *i.e.* by solving the ODE BVP form :

$$\frac{1}{2}\sigma^2 S^2 \frac{d^2\bar{V}}{dS^2} + rS\frac{d\bar{V}}{dS} - (r+\lambda)\bar{V} = -V(S,0) \quad \text{---- (37)}$$

$$\bar{V}(0;\lambda) = 0 \quad \text{---- (38)} \qquad \bar{V}(S;\lambda) \approx \frac{S}{\lambda} \text{ as } S \to \infty \quad \text{---- (39)}$$

  using the procedure described in 5.4.2

  In each case the initial values of V are converted into the u-values required by the finite difference method using the change of variable :

$$u(x,\tau) = \frac{V}{E} e^{-(\alpha x + \beta \tau)}$$

- the remaining solutions within each sub-domain are calculated by solving the diffusion equation (48), using the procedure described in 6.2.2

- by looking at the ODE BVP form (37) it can be seen that the term on the right-hand side of the equation is always (the negative of) the solution in the first time row. This means that the solutions along the first rows of the second and all following sub-domains can be calculated in parallel *i.e.* that each slave processor can calculate its own initial condition and then proceed to calculate the remaining solutions within its sub-domain.

6.4.4 Performance Data

The graphs and tables below provide a summary of the data collected. Once again, the graphs show visually, the relative performances of the numerical inversion algorithms. As before, only the graphs showing the minimum wall times are included. Detailed results are given in Appendix C.

6.4.4.1 Sequential Program Data

Table 6.2 below gives the data collected using the sequential, finite difference program.

	Value
NRMSD :	0.03543374850
Minimum Wall Time (s) :	6.81629490900

Table 6.2 Sequential, Finite Difference Program Data

6.4.4.2 Part 1 - Varying the Number of Weights/Terms Used

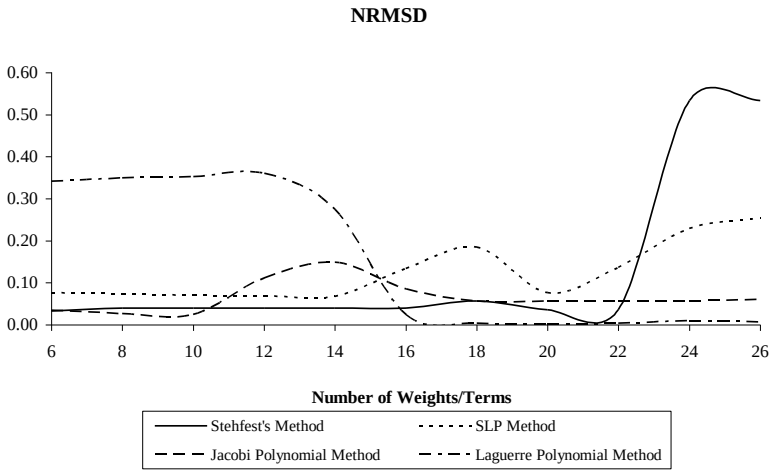

Figure 6.2 Normalised Root Mean Square Deviation, Parallel Programs

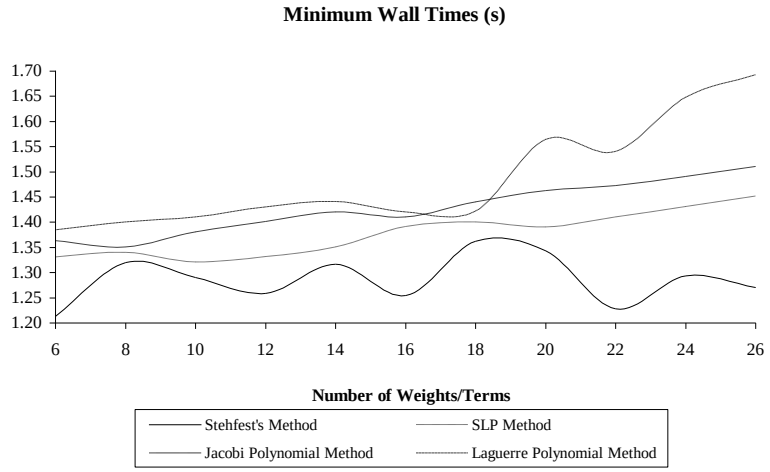

Figure 6.3 Minimum Wall Times, Parallel Programs

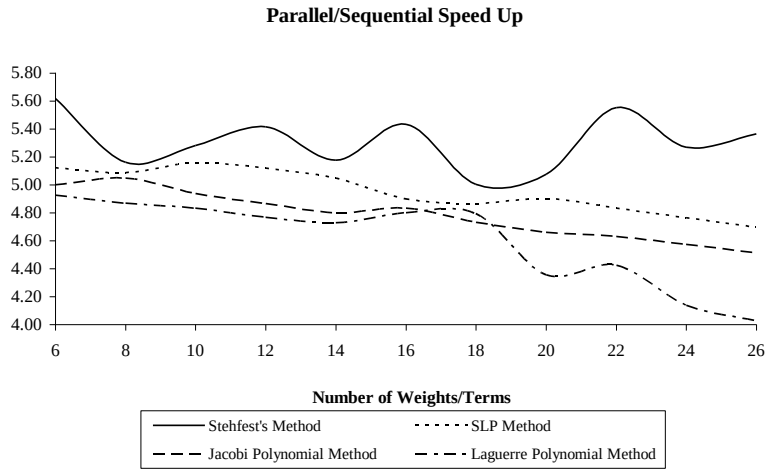

Figure 6.4 Parallel/Sequential Speed Up

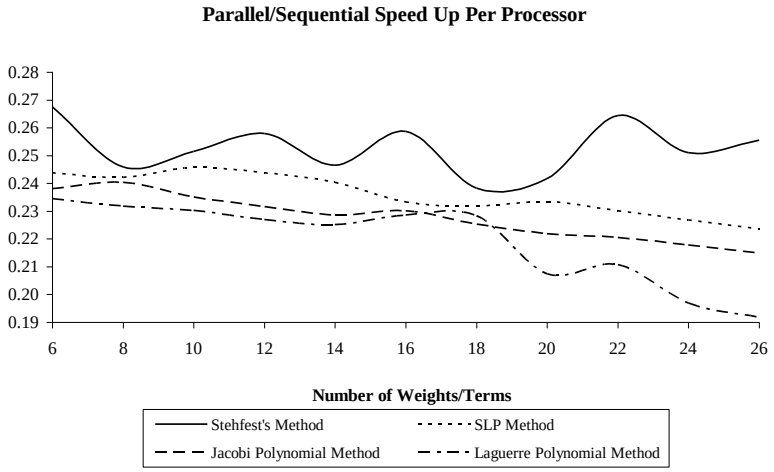

Figure 6.5 Parallel/Sequential Speed Up Per Processor

6.4.4.3 Part 2 - Varying the Number of Processors Used

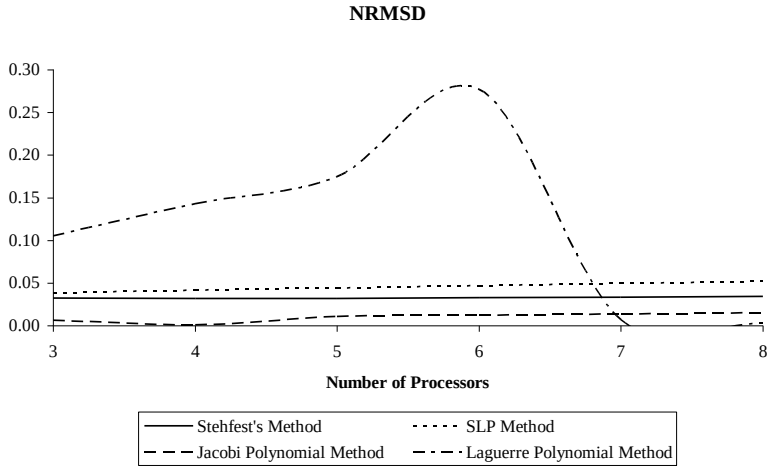

Figure 6.6 Normalised Root Mean Square Deviation (3-8 Processors)[56]

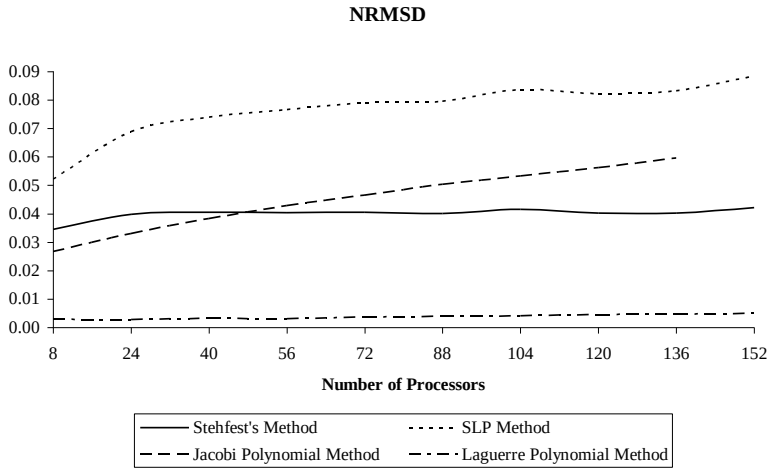

Figure 6.7 Normalised Root Mean Square Deviation (8-152 Processors)[57]

[56] The NRMSD values for Stehfest's method, the SLP method and the Jacobi Polynomial method follow slight oscillating tends.

[57] The NRMSD values for the Laguerre polynomial method follow a slight oscillating tend.

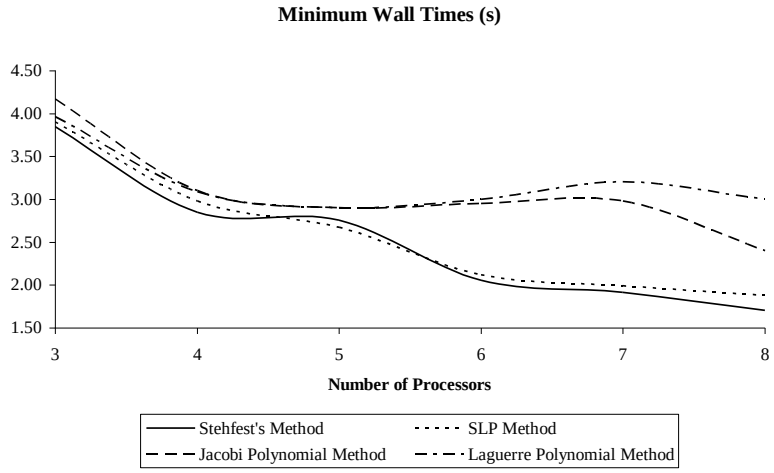

Figure 6.8 Minimum Wall Times (3-8 Processors)

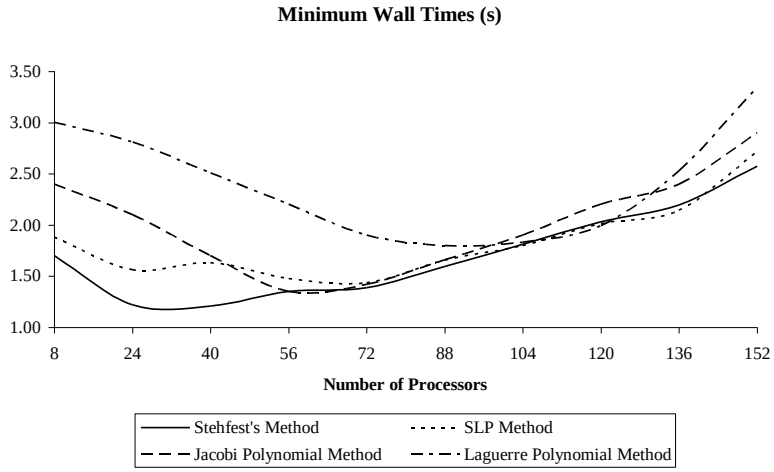

Figure 6.9 Minimum Wall Times (8-152 Processors)

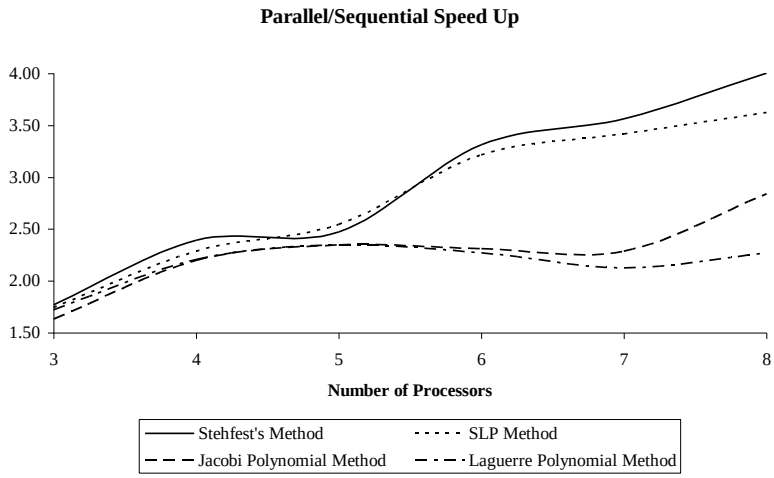

Figure 6.10 Parallel/Sequential Speed Up (3-8 Processors)

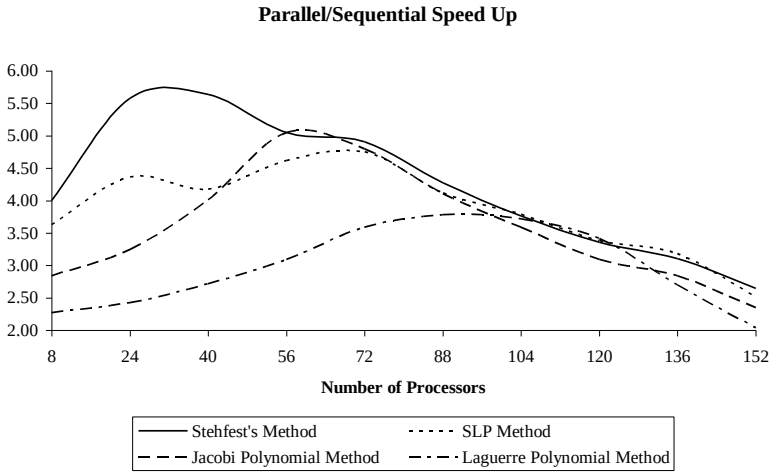

Figure 6.11 Parallel/Sequential Speed Up (8-152 Processors)

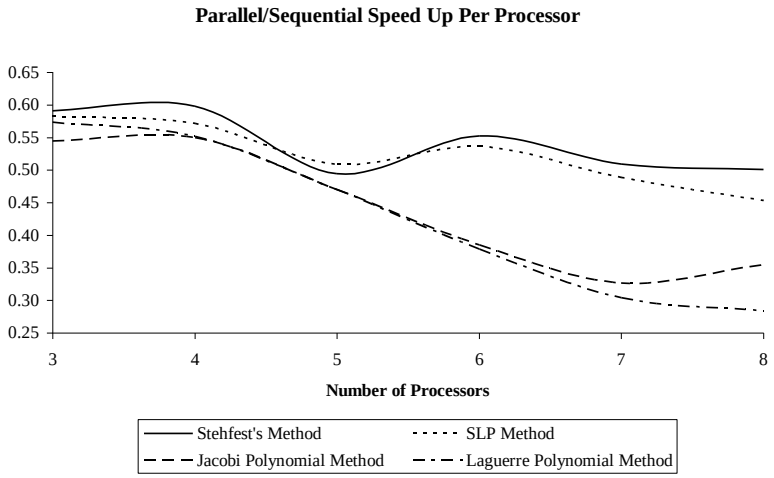

Figure 6.12 Parallel/Sequential Speed Up Per Processor (3-8 Processors)

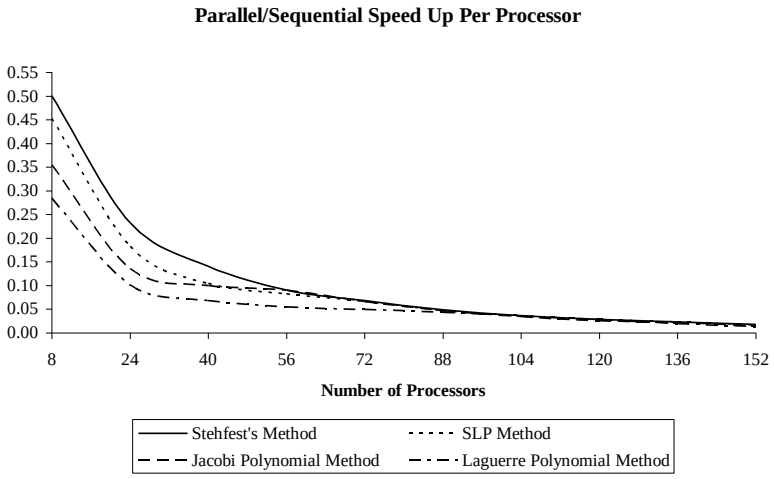

Figure 6.13 Parallel/Sequential Speed Up Per Processor (8-152 Processors)

6.4.4.4 Optimal Parallel Programs Data

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Processors	Optimal Value
NRMSD :	Jacobi	10	4	0.00088366262
Minimum Wall Time (s) :	Stehfest	6	40	1.20931506200
Parallel/Sequential Speed Up :	Stehfest	6	40	5.63649219561
Parallel/Sequential Speed Up/Processor :	Stehfest	6	4	0.59781910392

Table 6.3 Optimal Parallel Programs Data

6.4.5 Conclusions

Table 6.3 above gives the optimal number of weights/terms and processors to use in/with the LTFD algorithm when it is used to solve the Black-Scholes equation (10).

It can be seen from the data collected that the LTFD program is faster and more accurate than the sequential finite difference program. The reason for the improved accuracy is that the Laplace transform solution at the beginning of the second and all following sub-domains is more accurate than the corresponding time-marched result in the sequential finite difference solution. The Laplace transform approach appears to *pull* the finite difference solution back towards the analytical solution at the beginning of each sub-domain. The reason that the accuracy of the LTFD solution generally decreases as the number of processors increases is explained in Chapter 10.

Further information about the behaviour observed in this investigation is given in 7.4.5.

Once again, the *best* all-round numerical inversion algorithm is Stehfest's method.

6.5 Chapter Summary

The one-dimensional, Laplace transform-finite difference algorithm can accurately and quickly solve one-dimensional, linear Black-Scholes equations. Chapter 7 will determine whether this algorithm will be equally successful when it is used to solve nonlinear Black-Scholes equations of this type.

6.6 Contribution to Knowledge

This chapter has :

- introduced an improved computational procedure for solving the diffusion equation form of the one-dimensional, linear Black-Scholes equation
- developed and evaluated a parallel, Laplace transform-finite difference algorithm for solving the Black-Scholes equation (10) and shown that this algorithm produces fast and accurate solutions

- evaluated Stehfest's method, the shifted Legendre polynomial method, the Jacobi polynomial method and the Laguerre polynomial method when these methods are used to invert the Laplace transforms arising in the LTFD algorithm
- determined the optimal number of weights/terms and processors to use with each of the numerical inversion algorithms when they are used in the LTFD algorithm
- demonstrated the potential of the LTFD approach and established the advantages of using this algorithm for solving the one-dimensional linear Black-Scholes equation.

Chapter 7

Solving One-Dimensional, Nonlinear Black-Scholes Equations

7.0 Introduction

In 2.7 it was explained that when transaction costs are taken into account, Black-Scholes equations become nonlinear. Very few equations of this type have an analytical solution. Here, it is shown how the basic LTFD algorithm described in Chapter 6 can be modified to deal with the nonlinear terms and how, in the absence of analytical solutions, accurate reference solutions can be calculated.

7.1 Parameter Values

The parameter values used in the modified volatility models described in Chapter 2 are given in Table 7.1 below.

Modified Volatility Model	Parameter Values
Simulated Modified Volatility	-
Leland	$\delta t = 1/252$ [58], $k = 0.0072$ [59] *i.e.* $Le \approx 5.7907$
Boyle and Vorst	$\delta t = 1/6048$ [60], $k = 0.0072$
Barles and Soner	$\gamma = 0.25$ [61], $N = 1$, $a = 0.01$ [62]
Risk Adjusted Pricing Methodology	$C = 0.01$, $\delta t = 1/252$, $k = 0.0072$

Table 7.1 Parameter Values Used in the Modified Volatility Models

7.2 Practical Difficulties When Solving the Nonlinear Form

The one-dimensional, nonlinear Black-Scholes equation (32) is a much more complicated problem to solve than the linear equation (10). Before a corresponding LTFD algorithm can be developed for solving this equation, methods must be found for dealing with the nonlinear volatility term $\tilde{\sigma}^2$ and for calculating accurate reference solutions.

---

58 1 day

59 Estimated from the BP share price data on 18/9/2012

60 1 hour

61 Estimated from the Chicago Board Options Exchange (CBOE) Volatility Index on 31/12/2012. This index shows the expected volatility for a 30-day period.

62 1%

7.2.1 Linearisation Techniques

The Laplace transform is a linear operator and cannot be applied to a nonlinear PDE directly, (Crann 2005). This problem can sometimes be overcome by linearising the nonlinear terms in the equation before this operator is applied. The most commonly used linearisation techniques are direct iteration (Crann 2005), semi-direct iteration (Zhu 1999) and Taylor series iteration, (Zhu 1999). These methods will be illustrated using the second-order ODE BVP :

$$\frac{d^2x}{dt^2} - \left(1 - \frac{t}{5}\right)x^2 = t \qquad x(1) = 2, \qquad x(3) = -1 \qquad \text{---- (52)}$$

To solve this equation using the finite difference method, incorporating a linearisation technique :

- divide the interval $[1,3]$ into n+1 equally spaced points $x_0, x_1, \ldots, x_n$ where $x_{i+1} - x_i = h$
- linearise the x^2 term using one of the techniques mentioned above
- replace the first and second derivatives with finite difference approximations *e.g.*

$$\frac{dx}{dt} \approx \frac{x_{i+1} - x_{i-1}}{2h} \quad \text{and} \quad \frac{d^2x}{dt^2} \approx \frac{x_{i+1} - 2x_i + x_{i-1}}{h^2}$$

- approximate the boundary-value problem at each of the internal points $x_1, x_2, \ldots, x_{n-1}$. This operation produces a tridiagonal system of linear equations
- solve this system of linear equations iteratively to obtain the values of $x_1, x_2, \ldots, x_{n-1}$. The values of x_0 and x_n are obtained from the boundary conditions. The initial values of $x_1, x_2, \ldots, x_{n-1}$ are obtained by interpolation *i.e.* by fitting a straight line to the boundary conditions and then estimating their values from this. For equation (52) this straight line is $x = 3.5 - 1.5t$.

7.2.1.1 Direct Iteration

In direct iteration the nonlinear term is converted into a constant by using its value at the previous iteration, (Crann 2005). For example, at the *r*th iteration the nonlinear term x_i^2 would be written as $\left(x_i^{(r-1)}\right)^2$. Hence, using the procedure described in 7.2.1 with direct iteration, equation (52) becomes :

$$\frac{x_{i+1}^{(r)} - 2x_i^{(r)} + x_{i-1}^{(r)}}{h^2} - \left(1 - \frac{t_i}{5}\right)\left(x_i^{(r-1)}\right)^2 = t_i$$

and the expression used to approximate the boundary-value problem at each of the internal points $x_1, x_2, \ldots, x_{n-1}$ is :

$$x_{i-1}^{(r)} - 2x_i^{(r)} + x_{i+1}^{(r)} = h^2\left(t_i + \left(1 - \frac{t_i}{5}\right)\left(x_i^{(r-1)}\right)^2\right)$$

7.2.1.2 Semi-Direct Iteration

Semi-direct iteration can be used for linearising algebraic terms. Zhu (1999) shows that at the *r*th iteration a nonlinear term in the form x_i^n can be linearised by writing it as $\left(x_i^{(r-1)}\right)^{(n-1)} x_i^{(r)}$. Hence, using semi-direct iteration with the procedure given in 7.2.1, equation (52) becomes :

$$\frac{x_{i+1}^{(r)} - 2x_i^{(r)} + x_{i-1}^{(r)}}{h^2} - \left(1 - \frac{t_i}{5}\right)x_i^{(r-1)}x_i^{(r)} = t_i$$

and the expression used to approximate the boundary-value problem at each of the internal points $x_1, x_2, \ldots, x_{n-1}$ is :

$$x_{i-1}^{(r)} - \left(2 + h^2\left(1 - \frac{t_i}{5}\right)x_i^{(r-1)}\right)x_i^{(r)} + x_{i+1}^{(r)} = h^2 t_i$$

7.2.1.3 Taylor Series Iteration

Zhu (1999) also suggests that at the *r*th iteration a nonlinear term in the form $f\left(x_i^{(r)}\right)$ can be linearised by replacing it in the equation with a first-order Taylor series in the form :

$$f\left(x_i^{(r)}\right) = f\left(x_i^{(r-1)}\right) + f'\left(x_i^{(r-1)}\right)\left(x_i^{(r)} - x_i^{(r-1)}\right)$$

Hence, incorporating Taylor series iteration into the procedure described in 7.2.1, equation (52) becomes :

$$\frac{x_{i+1}^{(r)} - 2x_i^{(r)} + x_{i-1}^{(r)}}{h^2} - \left(1 - \frac{t_i}{5}\right)\left(\left(x_i^{(r-1)}\right)^2 + 2x_i^{(r-1)}\left(x_i^{(r)} - x_i^{(r-1)}\right)\right) = t_i$$

and the expression used to approximate the boundary-value problem at each of the internal points $x_1, x_2, \ldots, x_{n-1}$ is :

$$x_{i-1}^{(r)} - 2\left(1 + h^2\left(1 - \frac{t_i}{5}\right)x_i^{(r-1)}\right)x_i^{(r)} + x_{i+1}^{(r)} = h^2\left(t_i - \left(1 - \frac{t_i}{5}\right)\left(x_i^{(r-1)}\right)^2\right)$$

7.2.1.4 Termination

All three linearisation techniques are terminated when an accuracy criterion is satisfied. A commonly used condition is :

$$\left|\frac{\max\left(\text{abs}\left(x_i^{(r-1)} - x_i^{(r)}\right)\right)}{\max\left(\text{abs}\left(x_i^{(r-1)} + x_i^{(r)}\right)\right)}\right| < \varepsilon \quad \text{---- (53)}$$

where $x_i^{(r)}$ denotes the value of x_i at the *r*th iteration and ε is a small positive number.

7.2.1.5 Comparison of Methods

Crann (2005) evaluates these techniques by using them to solve a nonlinear Poisson equation[63] whose analytical solution is known. She found that all three methods were able to produce solutions that were close to those calculated using the analytical solution and that they had similar rates of convergence. Using $\varepsilon = 0.001$ in termination condition (53), the average number of iterations required for convergence was five. The limitations of these methods are that they require a good initial approximation to the solution, they are not guaranteed to converge and that they do not always linearise the equation being solved. For example, consider the second-order ODE BVP :

$$\frac{d^2x}{dt^2} - \left(1 - \frac{t}{5}\right)x^2\frac{dx}{dt} = t \qquad x(1) = 2\,, \quad x(3) = -1 \quad \text{---- (54)}$$

Applying direct iteration to this equation, together with the procedure given in 7.2.1 gives :

$$\frac{x_{i+1}^{(r)} - 2x_i^{(r)} + x_{i-1}^{(r)}}{h^2} - \left(1 - \frac{t_i}{5}\right)\left(x_i^{(r-1)}\right)^2\left(\frac{x_{i+1}^{(r)} - x_{i-1}^{(r)}}{2h}\right) = t_i$$

and hence :

$$\left(1 + \frac{h}{2}\left(1 - \frac{t_i}{5}\right)\left(x_i^{(r-1)}\right)^2\right)x_{i-1}^{(r)} - 2x_i^{(r)} + \left(1 - \frac{h}{2}\left(1 - \frac{t_i}{5}\right)\left(x_i^{(r-1)}\right)^2\right)x_{i+1}^{(r)} = h^2 t_i$$

In this case equation (54) has been linearised as required. However, using the same procedure with semi-direct iteration produces :

$$\frac{x_{i+1}^{(r)} - 2x_i^{(r)} + x_{i-1}^{(r)}}{h^2} - \left(1 - \frac{t_i}{5}\right)x_i^{(r-1)}x_i^{(r)}\left(\frac{x_{i+1}^{(r)} - x_{i-1}^{(r)}}{2h}\right) = t_i$$

and hence :

---

[63] A Poisson equation is a type of elliptic PDE. Equations of this type arise frequently in engineering and physics.

$$\left(1+\frac{h}{2}\left(1-\frac{t_i}{5}\right)x_i^{(r-1)}x_i^{(r)}\right)x_{i-1}^{(r)}-2x_i^{(r)}+\left(1-\frac{h}{2}\left(1-\frac{t_i}{5}\right)x_i^{(r-1)}x_i^{(r)}\right)x_{i+1}^{(r)}=h^2t_i \quad \text{---- (60)}$$

Using the same procedure with Taylor series iteration produces :

$$\frac{x_{i+1}^{(r)}-2x_i^{(r)}+x_{i-1}^{(r)}}{h^2}-\left(1-\frac{t_i}{5}\right)\left(\left(x_i^{(r-1)}\right)^2+2x_i^{(r-1)}\left(x_i^{(r)}-x_i^{(r-1)}\right)\right)\left(\frac{x_{i+1}^{(r)}-x_{i-1}^{(r)}}{2h}\right)=t_i$$

and hence :

$$\left(1+\frac{h}{2}\left(1-\frac{t_i}{5}\right)\left(\left(x_i^{(r-1)}\right)^2+2x_i^{(r-1)}\left(x_i^{(r)}-x_i^{(r-1)}\right)\right)\right)x_{i-1}^{(r)}-2x_i^{(r)}+$$
$$\left(1-\frac{h}{2}\left(1-\frac{t_i}{5}\right)\left(\left(x_i^{(r-1)}\right)^2+2x_i^{(r-1)}\left(x_i^{(r)}-x_i^{(r-1)}\right)\right)\right)x_{i+1}^{(r)}=h^2t_i \quad \text{---- (61)}$$

The presence of the $x_i^{(r)}x_{i-1}^{(r)}$ and $x_i^{(r)}x_{i+1}^{(r)}$ terms in expressions (60) and (61) indicates that these expressions produce systems of nonlinear equations. Hence, semi-direct iteration and Taylor series iteration have failed to linearise equation (54). These methods also fail to linearise the Black-Scholes equations (32) and (33) when they contain the modified volatility models proposed by Lai *et al*., Barles and Soner and Kratka (*i.e.* the Risk Adjusted Pricing Methodology model). Since direct iteration is a simple method can be used in all of these cases it will be the linearisation technique used in all future sequential and parallel nonlinear algorithms. Other linearisation techniques are available. It is left for future work to investigate and evaluate alternative methods of this kind. See 11.2.

7.2.1.6 Accuracy in the Nonlinear Modified Volatility Models

The modified volatility models proposed by Lai *et al*., Barles and Soner and Kratka (*i.e.* the Risk Adjusted Pricing Model) are functions of the value of the option *V*, either directly or indirectly via the *gamma* of the underlying asset. These models forecast the volatility used in the Black-Scholes equations.

By looking at the algorithms used for solving the one-dimensional, nonlinear Black-Scholes equation (32) shown in Figures 7.5 and 7.6 it can be seen that at the start of each iterative step, the modified volatility is calculated using the option values from the previous iteration. Hence, throughout the iterative procedure there is a close correspondence between the accuracy of the option values and the accuracy of the volatility forecast. Since the option values calculated using these forecasts are measured in monetary units, predicted volatilities accurate to two decimal places are sufficient.

7.2.2 Monte Carlo Algorithms

When solving the linear Black-Scholes equation (10) the accuracy of the numerical solution can be assessed by comparing it with a solution produced using the analytical solution (14). However, for nonlinear Black-Scholes equations very few analytical solutions exist. To generate accurate reference solutions in these cases numerical methods must be used. Ideally, the methods used should be independent *i.e.* not involve Laplace transforms or finite differences. This not only allows the accuracy of the solutions to be assessed but also ensures that the LTFD algorithm is not being compared with itself. A common method used for solving PDEs arising in mathematical finance is the Monte Carlo method.

7.2.2.1 History and Background

Monte Carlo methods were developed by the Polish mathematician Stanislaw Ulam at the Los Alamos laboratory in New Mexico during World War Two for evaluating the high-dimension integrals that arose during the development of the first atomic bomb[64] . The methods were first used for option pricing by Phelim Boyle in the 1970's, (Boyle 1977). In general, these methods obtain solutions by calculating and then averaging estimates for the quantity being calculated using random values drawn from appropriate statistical distributions. They have the advantage that they can be used to solve problems that are impossible to solve using other methods. However, they have the disadvantage that millions of random values are sometimes required to produce accurate results and hence these methods can be slow. A range of methods is available for speeding up the rate of convergence of Monte Carlo algorithms such as antithetic variables, control variate techniques, leapfrog methods, variance reduction techniques, overrelaxation, simulated annealing and Hamiltonian methods. Consideration of these method is beyond the scope of this research programme and interested readers should consult a reference such as Glasserman (2003) or Mackay (2004).

[64] For further information on the history and development of these methods see Ulam (1983).

7.2.2.2 The Monte Carlo Algorithm for Solving One-Dimensional, Linear Black-Scholes Equations

In the forward, linear Black-Scholes model, the future value of the underlying asset is given by the stochastic differential equation :

$$dS_\tau = rSd\tau + \sigma SdW(\tau) \quad \text{---- (62)}$$

where *r* is the risk-free interest rate, *S* is the current value of the underlying asset, τ is the time to expiry, σ is the volatility and $W(\tau)$ is a Brownian motion. The solution of equation (62) is :

$$S_\tau = S\exp\left(\left[r - \frac{1}{2}\sigma^2\right]\tau + \sigma W(\tau)\right)$$

(Wilmot *et al.* 1999). Glasserman (2003) shows that the Brownian motion $W(\tau)$ is normally distributed with a mean of zero and a variance of $d\tau$. The term $\sqrt{\tau}Z$, where Z is a standard normal random variable[65], has the same distribution. Hence, the future value of the underlying asset can be written as :

$$S_\tau = S\exp\left(\left[r - \frac{1}{2}\sigma^2\right]\tau + \sigma\sqrt{\tau}Z\right) \quad \text{---- (63)}$$

The payoff of a European call option at the expiry time is :

$$\max(S_\tau - E, 0) \quad \text{---- (64)}$$

where *E* is the exercise price. The seller of the option will wish to know the current value of the option. This can be calculated by pre-multiplying expression (64) by the discount factor $e^{-r\tau}$. Hence, the current value of the option *V* is :

$$V = e^{-r\tau}\max(S_\tau - E, 0) \quad \text{---- (65)}$$

Expressions (63) and (65) form the basis of the Monte Carlo algorithm. Given a mechanism for generating standard normal random variables *Z*, expressions (63) and (65) can be used to produce sample values of *V* that can be averaged to give an estimate of the current value of the option. More formally, the algorithm can be written as :

[65] A normally distributed random variable with a mean of 0 and a variance of 1.

`

loop i = 1 to n

generate Z_i

$$S_\tau = S\exp\left(\left[r - \frac{1}{2}\sigma^2\right]\tau + \sigma\sqrt{\tau}Z_i\right)$$

$$V_i = e^{-r\tau}\max\left(S_\tau - E, 0\right)$$

end loop

$$\hat{V} = \frac{\left(V_1 + V_2 + V_3 + \ldots + V_n\right)}{n} = \frac{1}{n}\sum_{i=1}^{n} V_i$$

Figure 7.1 The Monte Carlo Algorithm for Solving One-Dimensional, Linear Black-Scholes Equations[66]

Glasserman (2003) shows that by the strong law of large numbers[67], the estimator $\hat{V}$ is unbiased *i.e.* $\mathrm{E}\left(\hat{V}\right) = V$ and strongly consistent *i.e.* as $n \to \infty$, $\hat{V} \to V$ with a probability of one.

7.2.2.3 Random Number Generation

Most Monte Carlo algorithms are based upon a sequence of pseudo-random numbers u_i[68].

The u_i values must be :

- uniformly distributed between 0 and 1
- mutually independent *i.e.* u_i must not be predictable from u_{i-1}.

A variety of algorithms is available for generating values of this type. Glasserman (2003) states that the attributes of a good pseudo-random number generator are :

- a long period length. The algorithm must generate a long sequence of random values before that sequence repeats
- reproducability. The algorithm must be able to reproduce a particular sequence of random values in case it is required to re-run a simulation with the same inputs
- speed. Since millions of random values may be required the algorithm must be fast

---

66 This algorithm is due to Wilmot *et al.* (1999) and Glasserman (2003).

67 The average of the results obtained from a large number of trials will become closer to the expected value as the number of trials increases.

68 A sequence of random numbers generated by an algorithm.

- portability. The algorithm must produce the same sequence of random values on all computing platforms
- randomness. The algorithm must produce random values u_i with the properties listed above.

An algorithm that possesses all of these properties is the *multiple recursive generator* (MRG). This has the general form :

$$x_i = \left(a_1 x_{i-1} + a_2 x_{i-2} + \ldots + a_k x_{i-k}\right) \bmod m$$

$$u_i = \frac{x_i}{m}$$

where $a_i \in \mathbb{Z} \ \forall i$ and $m \in \mathbb{Z}$. The initial (*i.e.* seed) values of $x_{i-1}, x_{i-2}, \ldots, x_{i-k}$ are usually assigned using a simple algorithm such as the intrinsic random number generator available in a programming language. Research has shown that the period length of the sequence u_i can be increased significantly if two or more MRGs are combined. The combination is usually performed by summing the x_i values and then dividing this sum by the largest value of *m*. The pseudo-random number generator used in this investigation is the combined MRG shown in Figure 7.2. This is due to L'Ecuyer (1999). He combines two MRGs to produce an algorithm that has a period of 2^{319}. Details of the parameter values used in this algorithm can be found in Glasserman (2003).

$$x_{1,i} = \left(a_1 x_{1,i-1} + a_2 x_{1,i-2} + \ldots + a_k x_{1,i-k}\right) \bmod m_1$$

$$x_{2,i} = \left(b_1 x_{2,i-1} + b_2 x_{2,i-2} + \ldots + b_k x_{2,i-k}\right) \bmod m_2$$

$$m = \max\left(m_1, m_2\right)$$

$$y_i = \sum_{j=1}^{2} (-1)^{j-1} x_{j,i} \bmod m$$

$$u_i = \begin{cases} \dfrac{y_i}{m}, & y_i > 0 \\ \dfrac{(m-1)}{m} & y_i = 0 \end{cases}$$

Figure 7.2 The L'Ecuyer Combined Multiple Recursive Generator

7.2.2.4 Obtaining Standard Normal Values

To implement the Monte Carlo algorithm described in Figure 7.1 the u_i values must be converted into standard normal values. One of the most commonly used algorithms for doing this is the Box-Muller method, (Box and Muller 1958). This algorithm is based upon the following property. Let z_1 and z_2 be univariate[69] standard normal values. Then, z_1 and z_2 are the coordinates of a point on the circumference of a circle, centred at the origin with radius $\sqrt{R}$, where *R* is a random value from an exponential distribution with a mean of 2. Let u_1 and u_2 be uniformly distributed random numbers between 0 and 1. Then, the Box-Muller algorithm is :

- generate a random exponential value *R* $\boxed{R = -2\ln(u_1)}$
- generate a random angle *V* in the interval $[0, 2\pi]$ $\boxed{V = 2\pi u_2}$
- map the angle *V* to a point (z_1, z_2) on the circumference of a circle, centred at the origin with radius $\sqrt{R}$ $\boxed{z_1 = \sqrt{R}\cos V,\ z_2 = \sqrt{R}\sin V}$

This procedure is summarised in Figure 7.3.

Generate u_1 *and* u_2

$R = -2\ln(u_1)$

$V = 2\pi u_2$

Return $z_1 = \sqrt{R}\cos V,\ z_2 = \sqrt{R}\sin V$

Figure 7.3 The Box-Muller Algorithm

7.2.2.5 A Monte Carlo Algorithm for Solving One-Dimensional, Nonlinear Black-Scholes Equations

Jäckel (2002) states that to solve nonlinear Black-Scholes equations using Monte Carlo methods least squares techniques must be used. A more straightforward approach is to incorporate the direct iteration technique from 7.2.1.1 into the basic Monte Carlo algorithm given in Figure 7.1. The proposed algorithm is given in Figure 7.4.

---

[69] A univariate distribution is a distribution of a single variable. A distribution of a combination of two variables *e.g.* $z_1 + z_2$ is called a bivariate distribution.

initialise $\tilde{\sigma}$

repeat

loop i = 1 to n

generate Z_i

$$S_\tau = S\exp\left(\left[r - \frac{1}{2}\tilde{\sigma}^2\right]\tau + \tilde{\sigma}\sqrt{\tau}Z_i\right)$$

$$V_i = e^{-r\tau}\max\left(S_\tau - E, 0\right)$$

end loop

$$\hat{V} = \frac{\left(V_1 + V_2 + V_3 + \ldots + V_n\right)}{n} = \frac{1}{n}\sum_{i=1}^{n} V_i$$

recalculate $\tilde{\sigma}$

until $$\left|\frac{\max\left(\text{abs}\left(\hat{V}^{(r-1)} - \hat{V}^{(r)}\right)\right)}{\max\left(\text{abs}\left(\hat{V}^{(r-1)} + \hat{V}^{(r)}\right)\right)}\right| < \varepsilon$$

Figure 7.4 A Monte Carlo Algorithm for Solving One-Dimensional, Nonlinear Black-Scholes Equations[70]

Here $\tilde{\sigma}$ is the modified volatility, $\hat{V}^{(r)}$ denotes the value of $\hat{V}$ at the *r*th iteration and ε is a small positive number. This algorithm is simpler than the procedure given by Jäckel and converges for the modified volatility models considered in this research programme.

The Monte Carlo reference solutions are stored in data files. The relevant file is then read into a two-dimensional array at the beginning of the Laplace transform-finite difference program so that accurate comparison data is available for calculating the normalised root mean square deviation values.

7.3 The Laplace Transform-Finite Difference Algorithm

The Laplace-transform-finite difference algorithm for solving one-dimensional, nonlinear Black-Scholes equations is a more complicated and computationally expensive procedure than the algorithm for solving the corresponding linear equations. The main reason for this is the need to incorporate a linearisation technique into the basic algorithm described in Chapter 6.

---

[70] This algorithm was invented by the author *i.e.* is a contribution to knowledge. It is based upon the algorithm for solving one-dimensional, linear Black-Scholes equations developed by Wilmot *et al.* (1999) and Glasserman (2003).

As before, the solutions along the first row of the first sub-domain are found using the initial condition of the equation (11). The solutions along the first row of the second and all following sub-domains are found using the Laplace transform method *i.e.* by solving the ODE BVP form of the Black-Scholes equation (32) :

$$\frac{1}{2}\tilde{\sigma}^2 S^2 \frac{d^2\bar{V}}{dS^2} + rS\frac{d\bar{V}}{dS} - (r+\lambda)\bar{V} = -V(S,\tau_{i-1}) \quad \text{---- (66)}$$

subject to the boundary conditions :

$$\bar{V}(0;\lambda) = 0 \qquad \bar{V}(S;\lambda) \sim \frac{S}{\lambda} \text{ as } S \to \infty$$

In equation (66) the function on the right-hand side is the solution of the equation in the previous time row. This gives a good initial approximation to the required solution and helps to ensure that the linearisation technique converges. However, the consequence of this is that in the nonlinear case, the initial conditions cannot be calculated in parallel. Instead, they must be calculated one at a time at the beginning of the program and then passed to the slave processors as parameters. Equation (66) is solved using the finite difference method for BVPs incorporating the direct iteration linearisation technique described in 7.2.1.1. This is a computationally expensive procedure since the inverse Laplace transform of the $\bar{V}$ values must be found at each iterative step in order to update the modified volatility $\tilde{\sigma}$. The algorithm used to calculate the solutions along the first row of the second and all following sub-domains is summarised in Figure 7.5.

initialise $\underline{V}$ *using initial condition (11)*

repeat

solve equation (66) (using $\underline{V}$ *to calculate the* $\tilde{\sigma}^2$ *values) to obtain* $\underline{\bar{V}}$

invert $\underline{\bar{V}}$ *obtain* $\underline{V}$

until $\left| \dfrac{\max\left(\text{abs}\left(V^{(r-1)} - V^{(r)}\right)\right)}{\max\left(\text{abs}\left(V^{(r-1)} + V^{(r)}\right)\right)} \right| < \varepsilon$

Figure 7.5 The Algorithm for Calculating the Subsequent Initial Conditions

Here, $V^{(r)}$ denotes a value in $\underline{V}$ at the *r*th iteration and ε is a small positive number. The remaining solutions within each sub-domain are calculated using the explicit finite difference method. This procedure does not involve tridiagonal matrices. Hence, the

computational advantages of applying this technique to a diffusion equation form of the equation disappear. In this algorithm the explicit method is therefore applied to the nonlinear Black-Scholes equation (32) directly. This produces the iterative formula :

$$V_{i,j+1} = \frac{1}{1+kr}\left[\frac{k}{2h^2}\tilde{\sigma}^2 S^2 V_{i-1,j} + \left(1 - \frac{krS}{h} - \frac{k}{h^2}\tilde{\sigma}^2 S^2\right)V_{i,j} + \frac{k}{h}\left(\frac{1}{2h}\tilde{\sigma}^2 S^2 + rS\right)V_{i+1,j}\right] \quad \text{---- (67)}$$

where *k* is the τ -step and *h* is the *S*-step. The rows within each sub-domain are calculated in sequence. The algorithm used to calculate each row $\underline{V}$ is summarised in Figure 7.6.

initialise $\underline{V}$ *using the previous row in the sub-domain*

repeat

calculate the $\tilde{\sigma}^2$ *values using* $\underline{V}$

use expression (67) to calculate $\underline{V}$

until $\left|\dfrac{\max\left(\text{abs}\left(V^{(r-1)} - V^{(r)}\right)\right)}{\max\left(\text{abs}\left(V^{(r-1)} + V^{(r)}\right)\right)}\right| < \varepsilon$

Figure 7.6 The Algorithm for Calculating the Rows of Each Sub-Domain

Here, $V^{(r)}$ and ε are as defined before.

7.4 Solving One-Dimensional, Nonlinear Black-Scholes Equations

7.4.1 Parameter Values

The parameter values used in the Black-Scholes equation (32) are :

Parameter	Values
S	0.0(0.1)5.0[71]
E	10
r	0.05
σ	0.25
τ	0.0(0.1)100.0

Table 7.2 Parameter Values Used in the One-Dimensional, Nonlinear Black-Scholes Equation

[71] The *S*-range is reduced for the nonlinear modified volatility models because of the slow speed of the Monte Carlo algorithm for calculating the reference solutions. Using the range used with the linear equation the Monte Carlo algorithm failed to produce sufficiently accurate solutions after 150 hours of processing time.

For a European call option the modified volatilities proposed by Leland and Boyle and Vorst become constants. The data for these functions is therefore collected using the LTFD program for the one-dimensional, linear Black-Scholes equation (10) and the parameter values given in Table 6.1.

7.4.2 Aim

The aim of this investigation is to collect speed and accuracy data for sequential and parallel algorithms for solving one-dimensional, nonlinear Black-Scholes equations and use it to compare their relative performances.

7.4.3 Preliminary Notes

- Düring *et al.* (2003) recommend estimating the gamma term in the Barles and Soner and Risk Adjusted Pricing Methodology modified volatility models using the finite difference approximation :

$$\Gamma = \frac{V_{i+2} - 2V_i + V_{i-2}}{4h^2}$$

  where h is the spatial step. They claim that this expression gives a better approximation to the second spatial derivative than the usual central difference approximation[72]

- the reference solutions used to determine the NRMSD values for the nonlinear Black-Scholes equations are calculated using the Monte Carlo algorithm given in Figure 7.4

- in 4.3 it was explained that the load on the cluster should not have a significant effect upon the program wall times. However, this does not mean that it will not have any effect at all. While all the codes for a particular nonlinear equation were run at the same time *i.e.* under the same load conditions, the codes for other equations were run on different days. Hence, when comparing the performances of the numerical inversion algorithms it is important to do so only for the same modified volatility model.

7.4.4 Performance Data

The graphs and tables below provide a summary of the data collected. Once again, the graphs show visually, the relative performances of the numerical inversion algorithms. As before, only the graphs showing the minimum wall times are included. Detailed results are given in Appendix D.

---

[72] Their experiments showed that if the usual central difference approximation is used, the resulting finite difference scheme becomes unstable for small values of the spatial step unless small values of the time step are also used.

7.4.4.1 Modified Volatility Model : Simulated Modified Volatility[73]

7.4.4.1.1 Sequential Program Data

Table 7.3 below gives the data collected using the sequential, finite difference program.

	Value
NRMSD :	0.08588035887
Minimum Wall Time (s) :	0.53442788120

Table 7.3 Sequential, Finite Difference Program Data (Simulated Modified Volatility)

7.4.4.1.2 Part 1 - Varying the Number of Weights/Terms Used

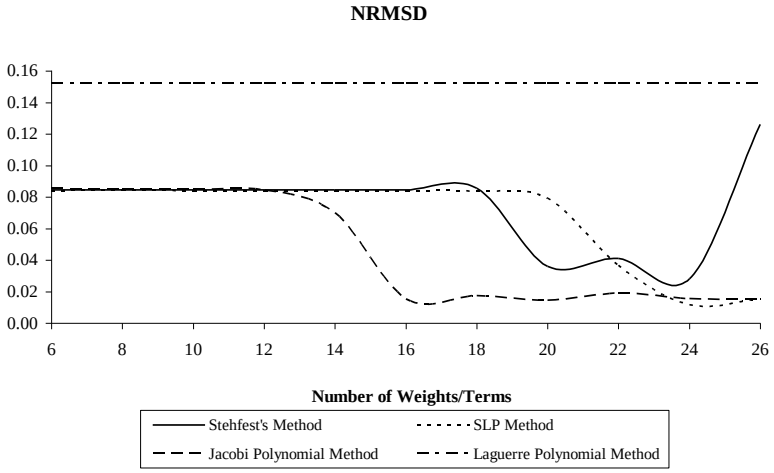

Figure 7.7 Normalised Root Mean Square Deviation, Parallel Programs (Simulated Modified Volatility)[74]

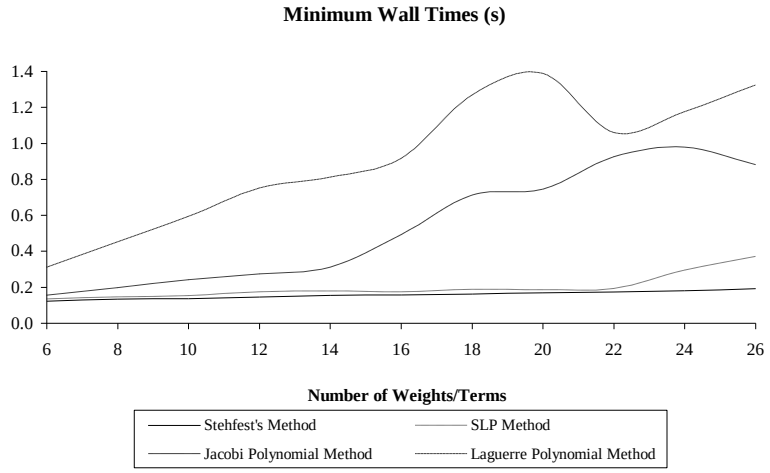

Figure 7.8 Minimum Wall Times, Parallel Programs (Simulated Modified Volatility)[75]

---

73 See Lai *et al.* (2005).

74 The NRMSD values for all numerical inversion algorithms follow oscillating trends.

75 The minimum wall times for Stehfest's method follow a slight upward trend.

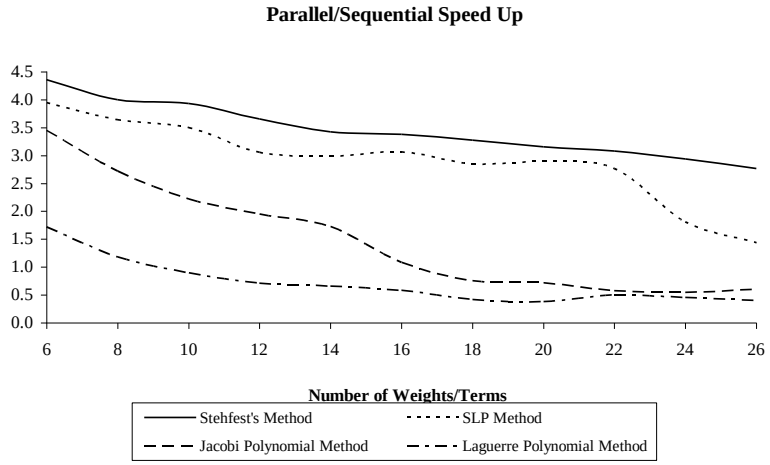

Figure 7.9 Parallel/Sequential Speed Up (Simulated Modified Volatility)

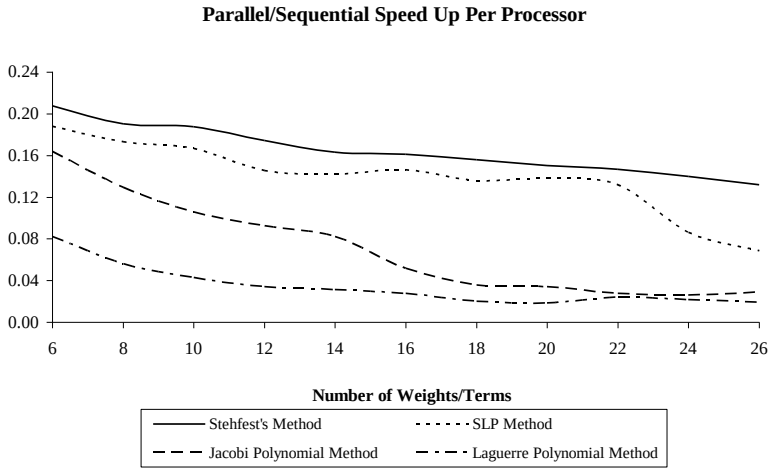

Figure 7.10 Parallel/Sequential Speed Up Per Processor (Simulated Modified Volatility)

7.4.4.1.3 Part 2 - Varying the Number of Processors Used

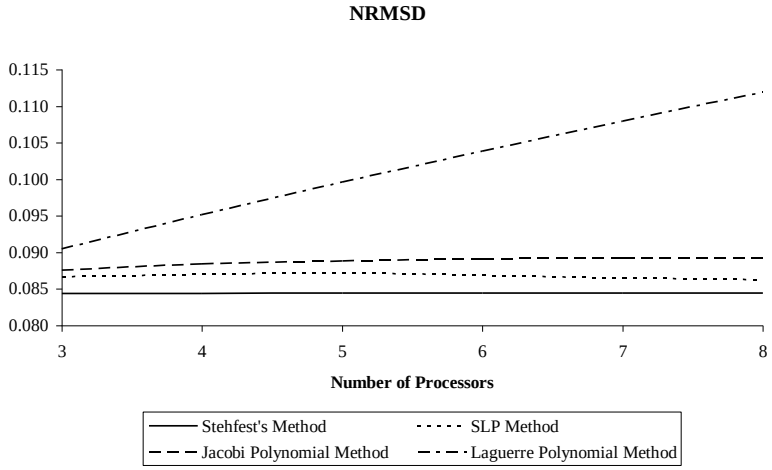

Figure 7.11 Normalised Root Mean Square Deviation (Simulated Modified Volatility) (3-8 Processors)[76]

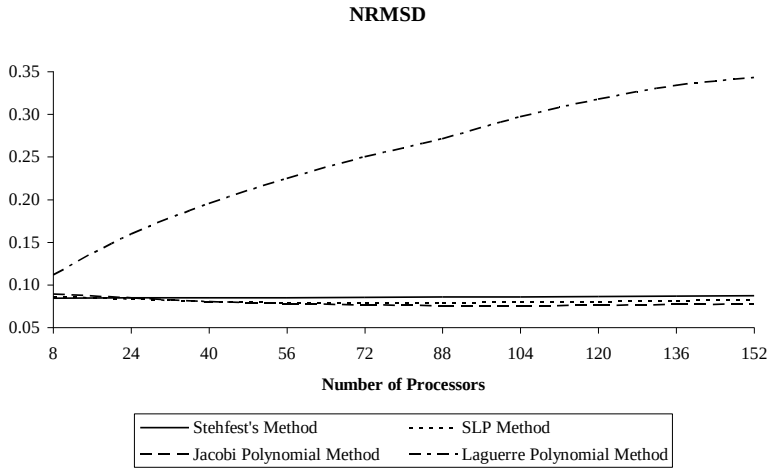

Figure 7.12 Normalised Root Mean Square Deviation (Simulated Modified Volatility) (8-152 Processors)[77]

[76] The NRMSD values for Stehfest's method follow a slight upward trend. For the SLP method the NRMSD values oscillate slightly.

[77] The NRMSD values for Stehfest's method follow a slight upward trend. For the SLP method and the Jacobi polynomial method the NRMSD values oscillate slightly.

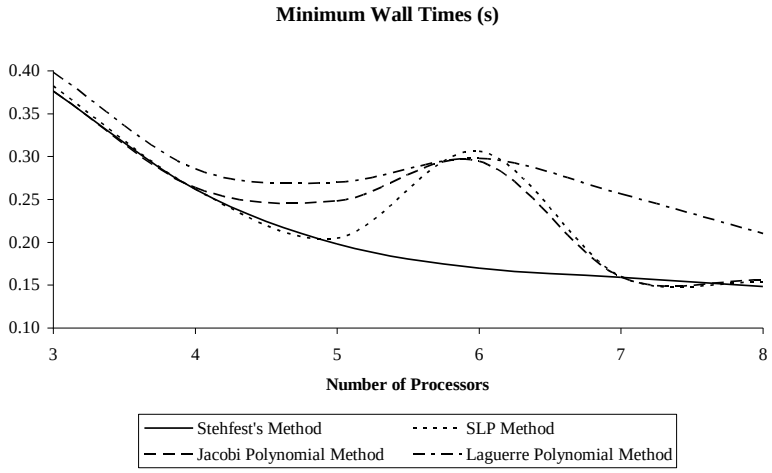

Figure 7.13 Minimum Wall Times (Simulated Modified Volatility) (3-8 Processors)

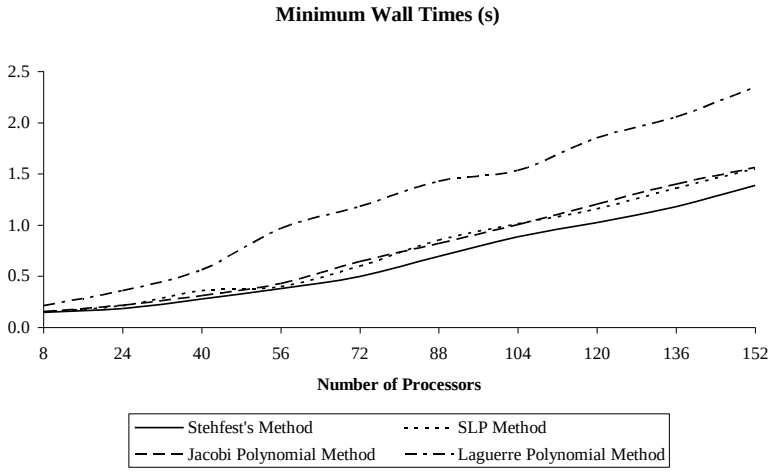

Figure 7.14 Minimum Wall Times (Simulated Modified Volatility) (8-152 Processors)

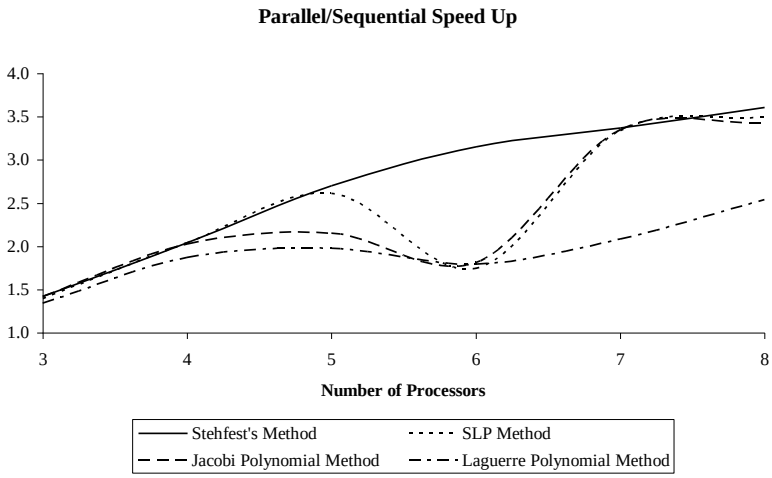

Figure 7.15 Parallel/Sequential Speed Up (Simulated Modified Volatility) (3-8 Processors)

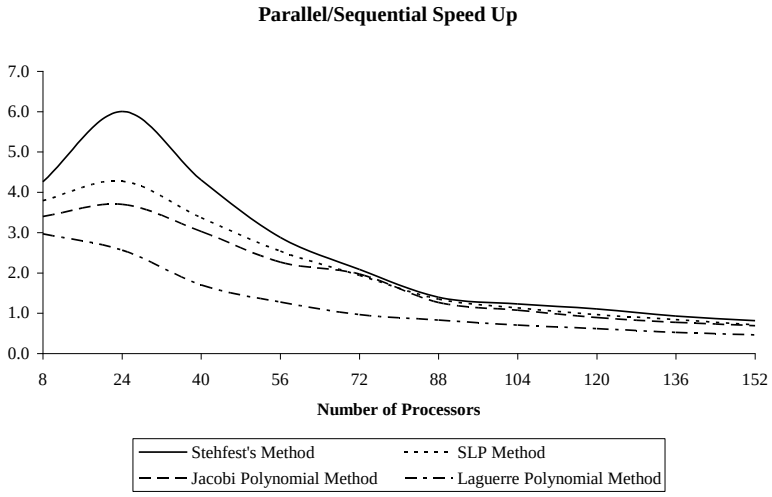

Figure 7.16 Parallel/Sequential Speed Up (Simulated Modified Volatility) (8-152 Processors)

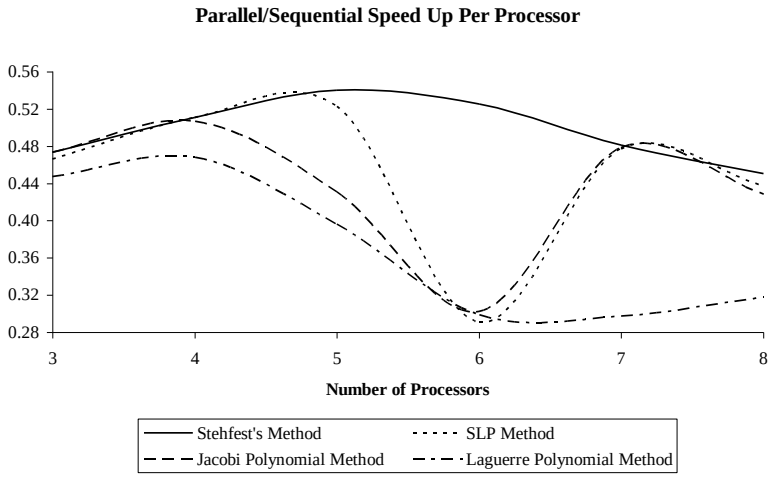

Figure 7.17 Parallel/Sequential Speed Up Per Processor (Simulated Modified Volatility) (3-8 Processors)

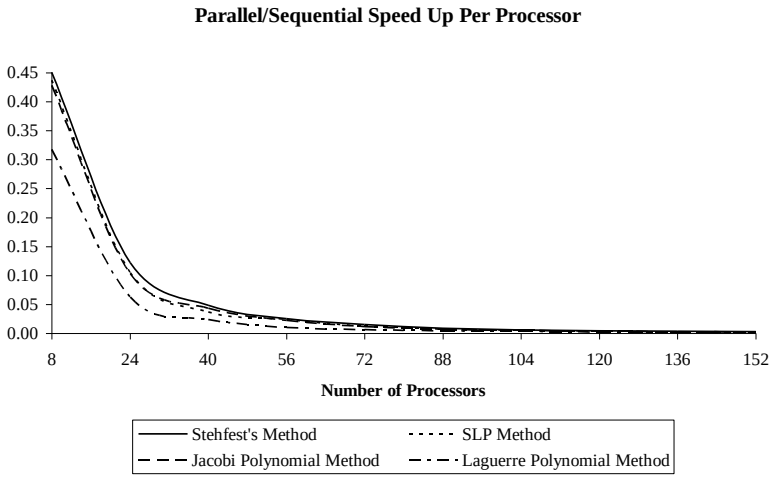

Figure 7.18 Parallel/Sequential Speed Up Per Processor (Simulated Modified Volatility) (8-152 Processors)

7.4.4.1.4 Optimal Parallel Programs Data

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Processors	Optimal Value
NRMSD :	SLP	24	21	0.01175761424
Minimum Wall Time (s) :	Stehfest	6	21	0.12257361320
Parallel/Sequential Speed Up :	Stehfest	6	21	4.36005651827
Parallel/Sequential Speed Up/Processor :	Stehfest	6	5	0.54030901625

Table 7.4 Optimal Parallel Programs Data (Simulated Modified Volatility)

7.4.4.2 Modified Volatility Model : Leland

7.4.4.2.1 Sequential Program Data

Table 7.5 below gives the data collected using the sequential, finite difference program.

	Value
NRMSD :	0.01402741597
Minimum Wall Time (s) :	8.32147940000

Table 7.5 Sequential, Finite Difference Program Data (Leland)

7.4.4.2.2 Part 1 - Varying the Number of Weights/Terms Used

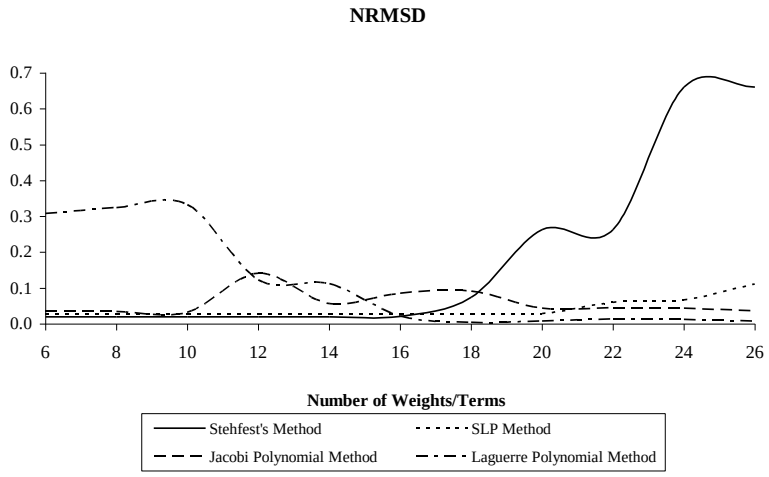

Figure 7.19 Normalised Root Mean Square Deviation, Parallel Programs (Leland)[78]

[78] The NRMSD values for Stefhest's method and the SLP method follow slight oscillating trends.

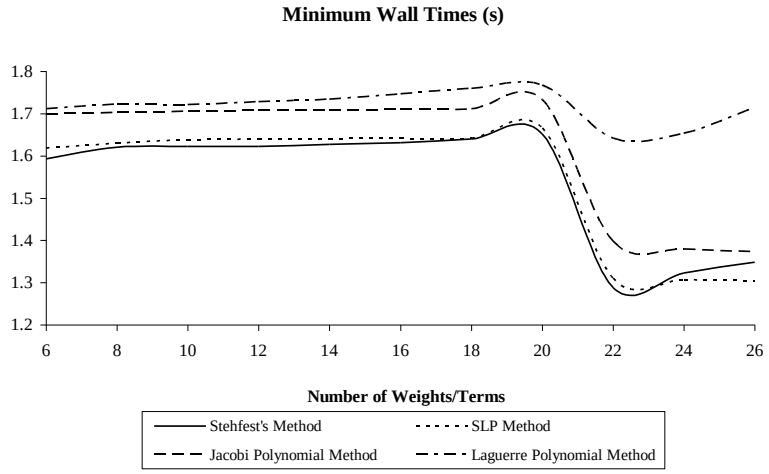

Figure 7.20 Minimum Wall Times, Parallel Programs (Leland)

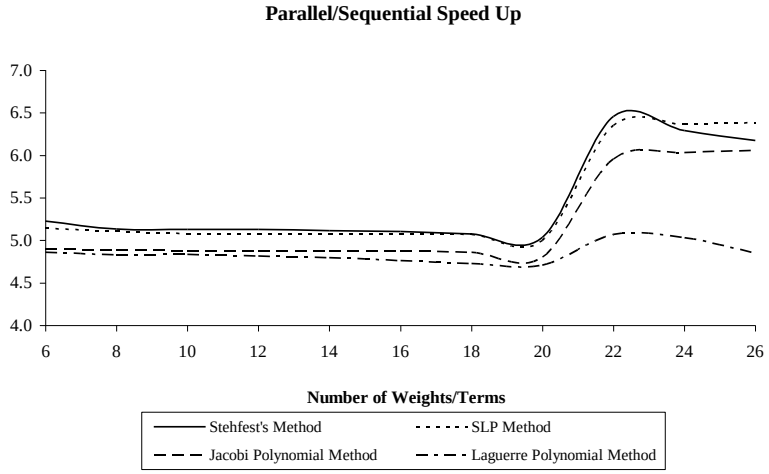

Figure 7.21 Parallel/Sequential Speed Up (Leland)

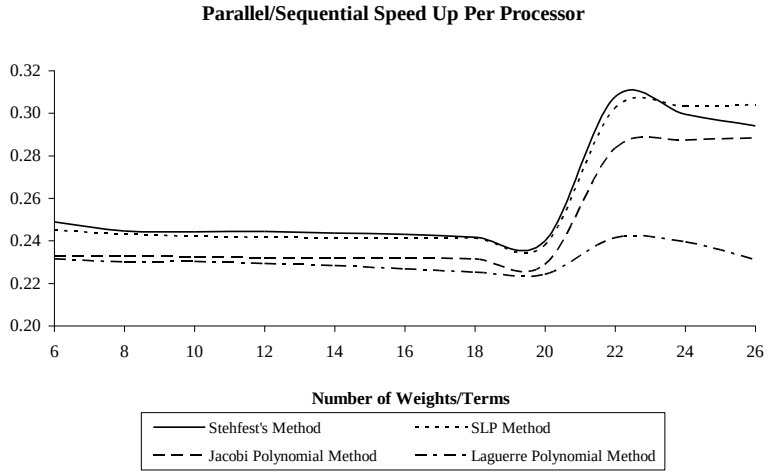

Figure 7.22 Parallel/Sequential Speed Up Per Processor (Leland)

7.4.4.2.3 Part 2 - Varying the Number of Processors Used

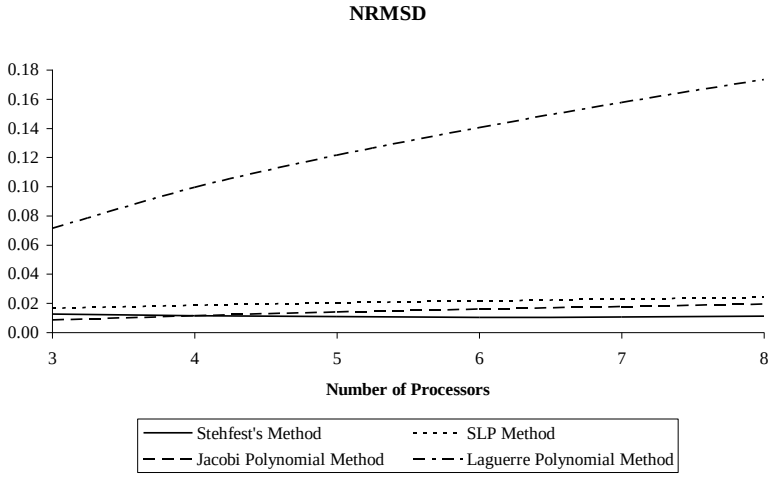

Figure 7.23 Normalised Root Mean Square Deviation (Leland) (3-8 Processors)[79]

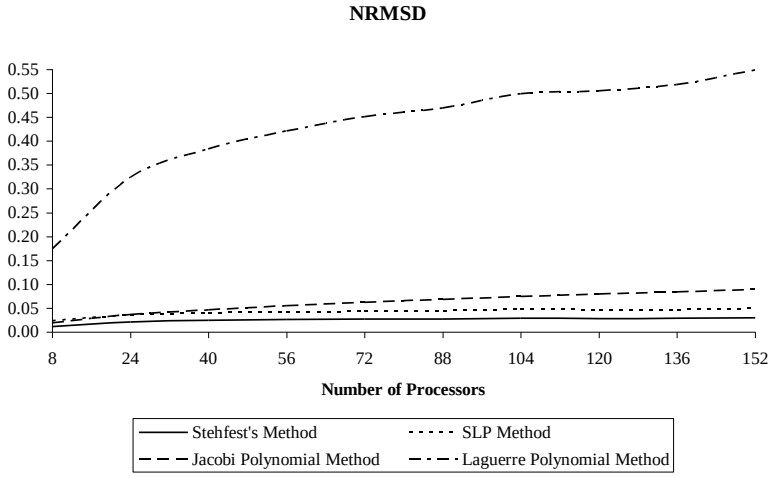

Figure 7.24 Normalised Root Mean Square Deviation (Leland) (8-152 Processors)

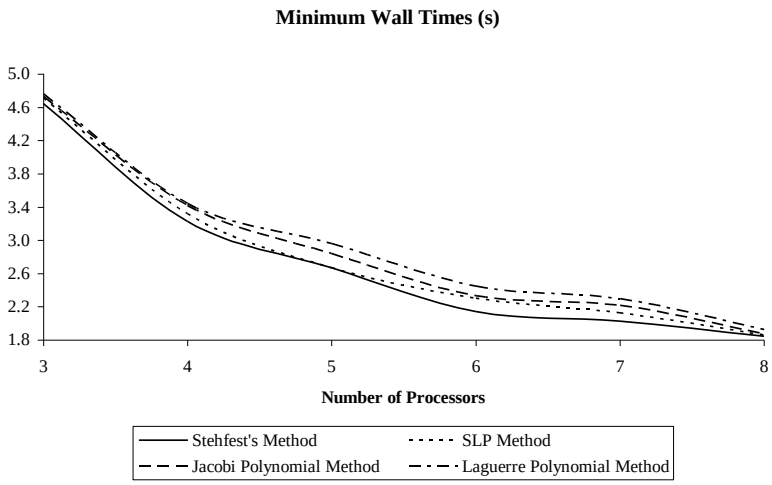

Figure 7.25 Minimum Wall Times (Leland) (3-8 Processors)

[79] The NRMSD values for Stehfest's method follow a slight oscillating trend. For the SLP method and the Jacobi polynomial method the NRMSD values follow a slight upward trend.

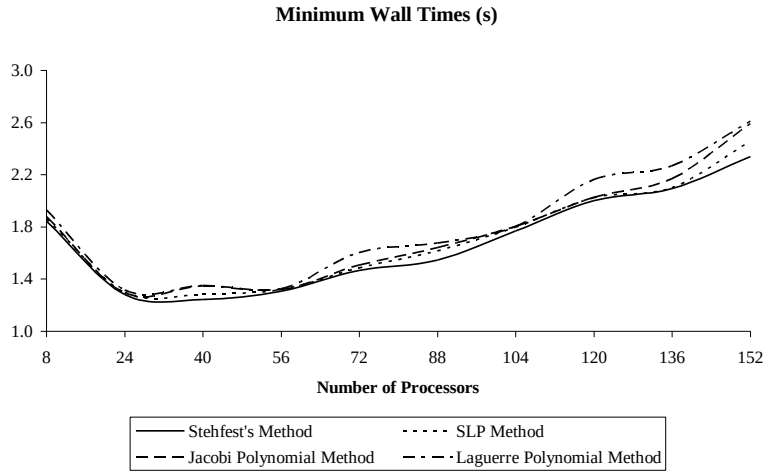

Figure 7.26 Minimum Wall Times (Leland) (8-152 Processors)

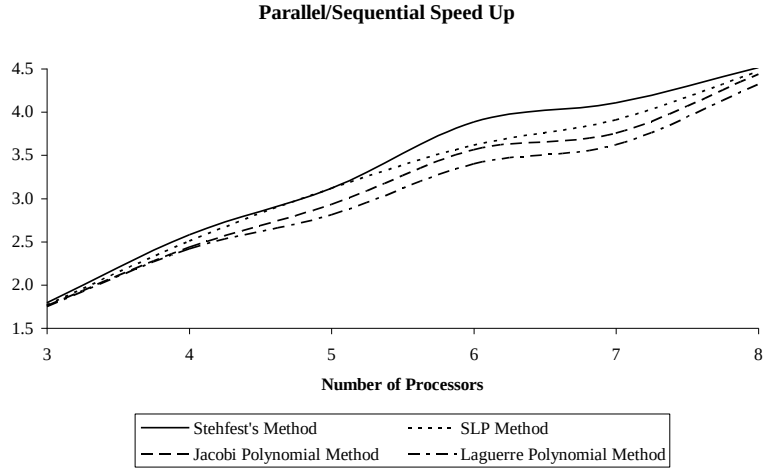

Figure 7.27 Parallel/Sequential Speed Up (Leland) (3-8 Processors)

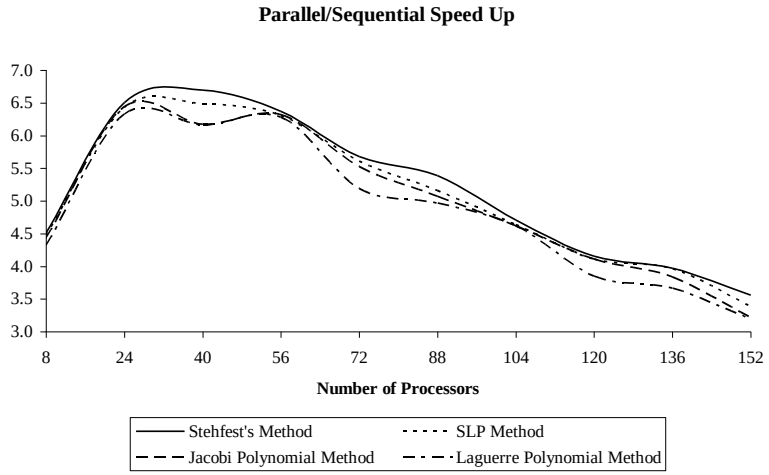

Figure 7.28 Parallel/Sequential Speed Up (Leland) (8-152 Processors)

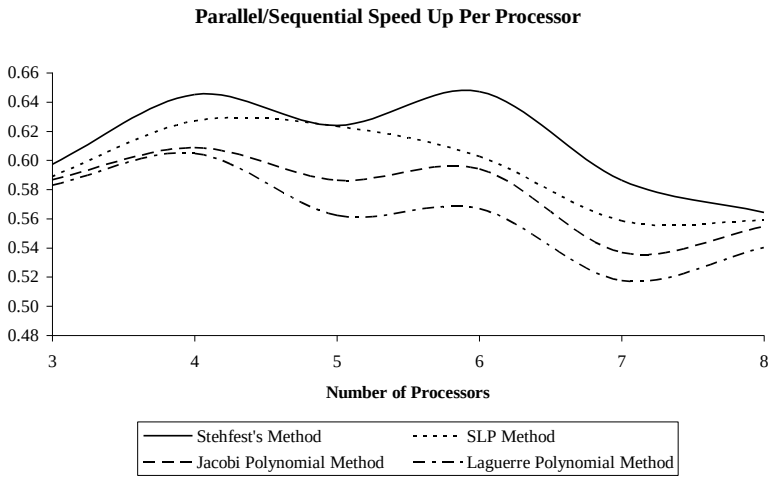

Figure 7.29 Parallel/Sequential Speed Up Per Processor (Leland) (3-8 Processors)

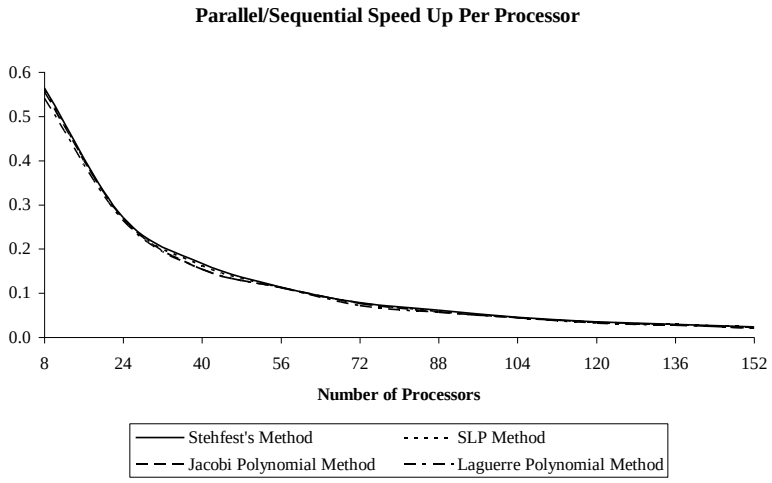

Figure 7.30 Parallel/Sequential Speed Up Per Processor (Leland) (8-152 Processors)

7.4.4.2.4 Optimal Parallel Programs Data

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Processors	Optimal Value
NRMSD :	Laguerre	20	21	0.00769955575
Minimum Wall Time (s) :	Stehfest	6	40	1.24268794100
Parallel/Sequential Speed Up :	Stehfest	6	40	6.45417539463
Parallel/Sequential Speed Up/Processor :	Stehfest	6	6	0.62382862880

Table 7.6 Optimal Parallel Programs Data (Leland)

7.4.4.3 Modified Volatility Model : Boyle and Vorst

7.4.4.3.1 Sequential Program Data

Table 7.7 below gives the data collected using the sequential, finite difference program.

	Value
NRMSD :	0.04648613774
Minimum Wall Time (s) :	7.50991988200

Table 7.7 Sequential, Finite Difference Program Data (Boyle and Vorst)

7.4.4.3.2 Part 1 - Varying the Number of Weights/Terms Used

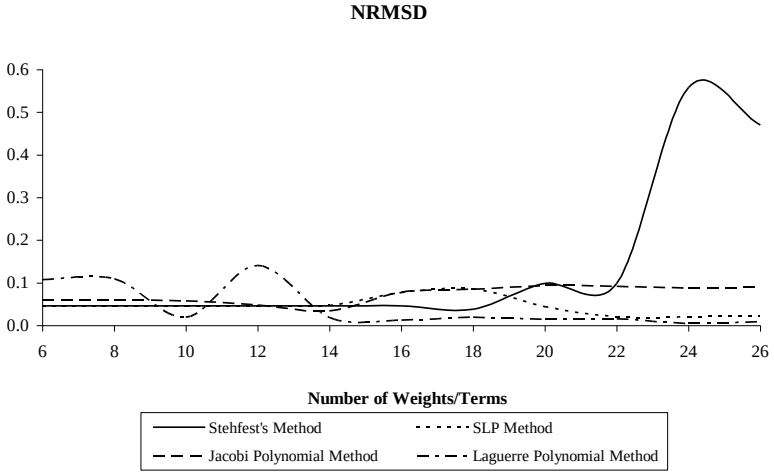

Figure 7.31 Normalised Root Mean Square Deviation, Parallel Programs (Boyle and Vorst)[80]

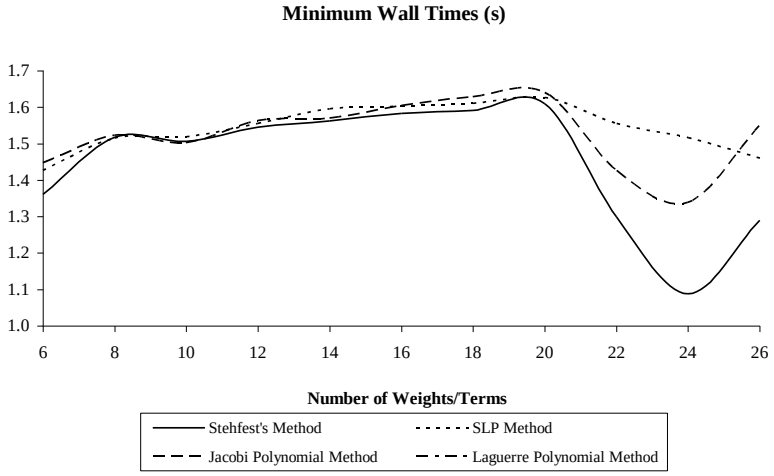

Figure 7.32 Minimum Wall Times, Parallel Programs (Boyle and Vorst)

[80] The NRMSD values for Stehfest's method follow a slight oscillating trend.

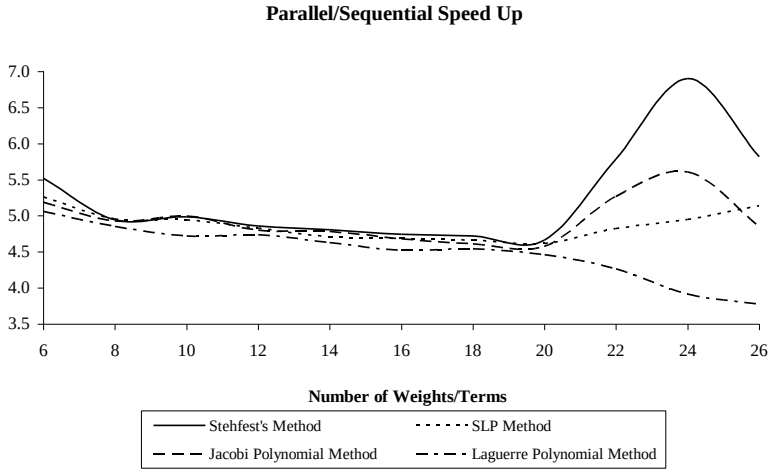

Figure 7.33 Parallel/Sequential Speed Up (Boyle and Vorst)

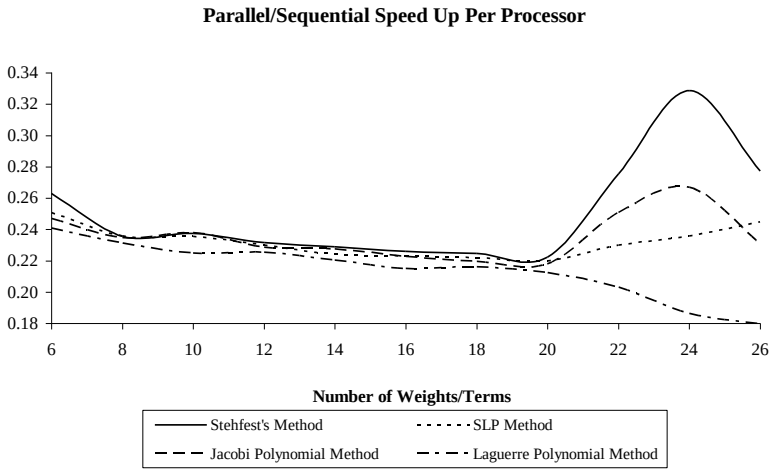

Figure 7.34 Parallel/Sequential Speed Up Per Processor (Boyle and Vorst)

7.4.4.3.3 Part 2 - Varying the Number of Processors Used

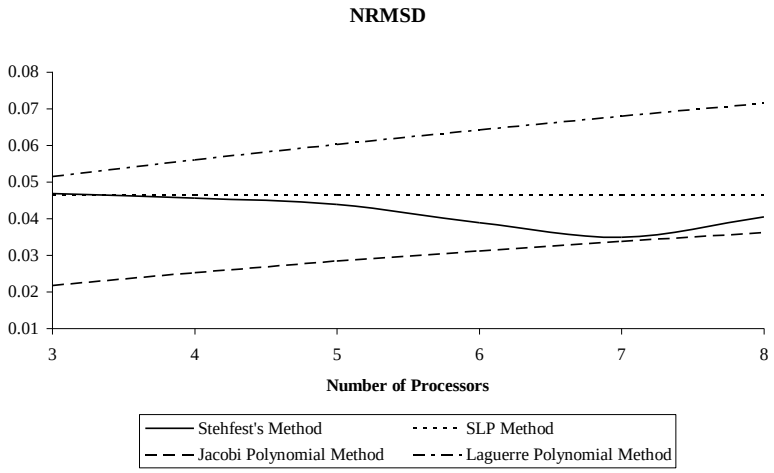

Figure 7.35 Normalised Root Mean Square Deviation (Boyle and Vorst) (3-8 Processors)[81]

[81] The NRMSD values for Stehfest's method and the SLP method follow slight decreasing trends.

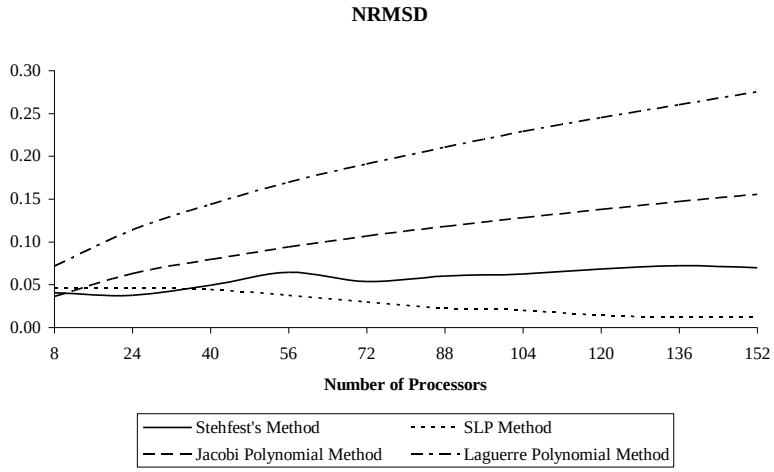

Figure 7.36 Normalised Root Mean Square Deviation (Boyle and Vorst) (8-152 Processors)[82]

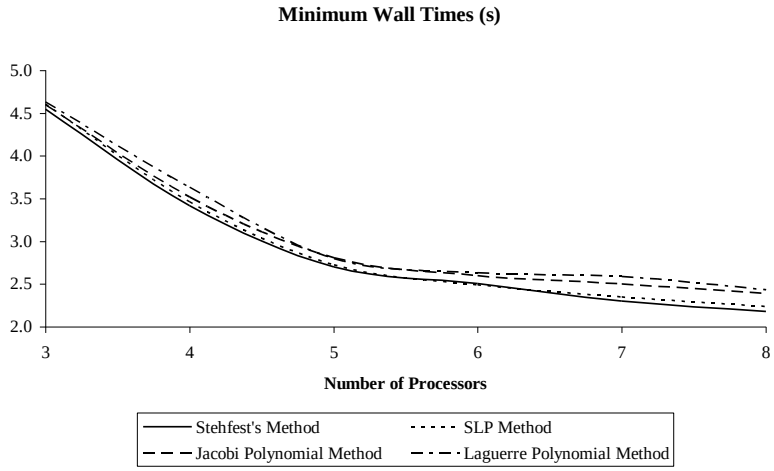

Figure 7.37 Minimum Wall Times (Boyle and Vorst) (3-8 Processors)

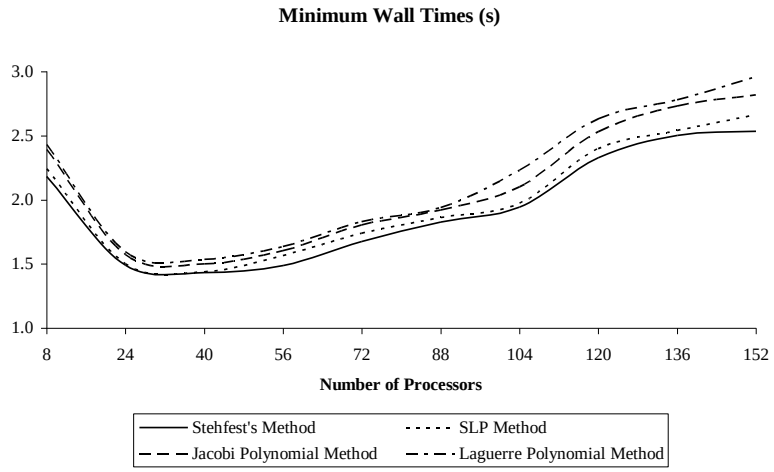

Figure 7.38 Minimum Wall Times (Boyle and Vorst) (8-152 Processors)

[82] The NRMSD values for Stehfest's method follow a slight decreasing trend.

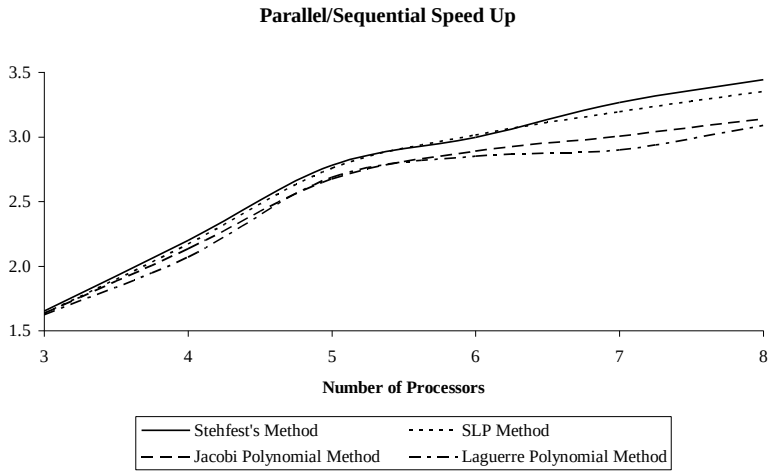

Figure 7.39 Parallel/Sequential Speed Up (Boyle and Vorst) (3-8 Processors)

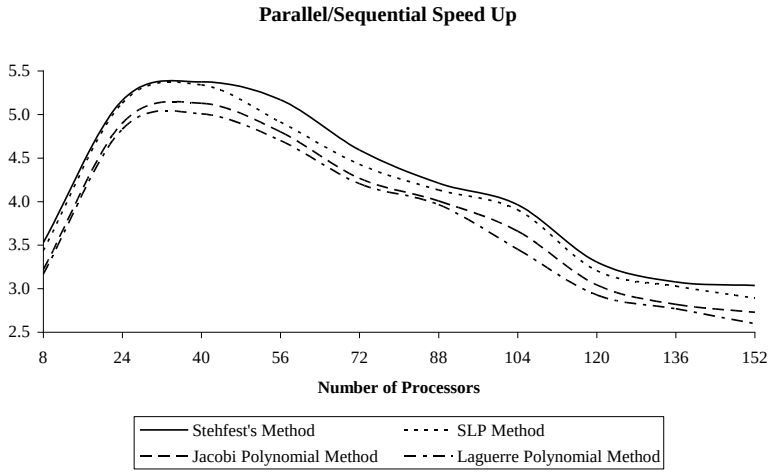

Figure 7.40 Parallel/Sequential Speed Up (Boyle and Vorst) (8-152 Processors)

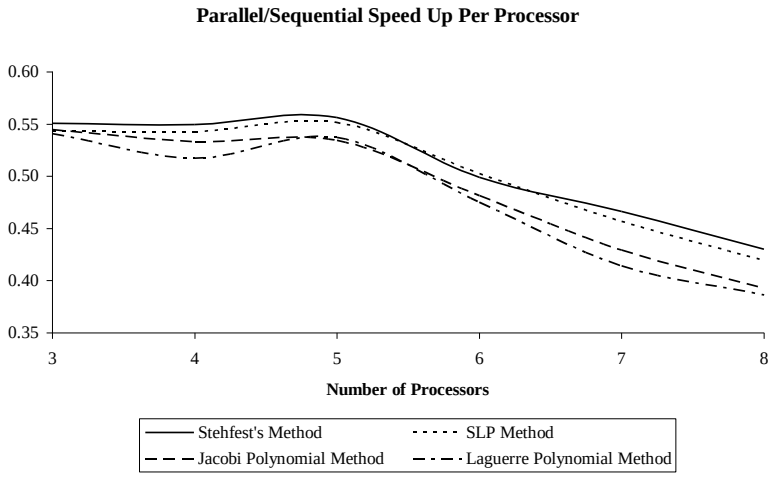

Figure 7.41 Parallel/Sequential Speed Up Per Processor (Boyle and Vorst) (3-8 Processors)

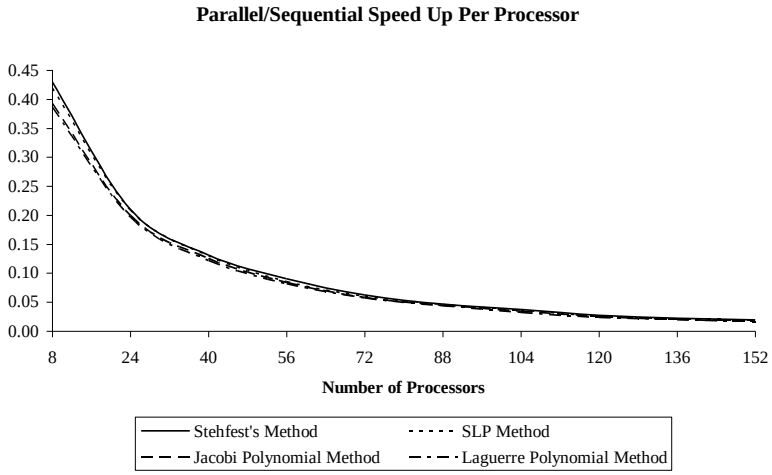

Figure 7.42 Parallel/Sequential Speed Up Per Processor (Boyle and Vorst) (8-152 Processors)

7.4.4.3.4 Optimal Parallel Programs Data

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Processors	Optimal Value
NRMSD :	SLP	20	136	0.01167072834
Minimum Wall Time (s) :	Stehfest	6	21	1.36069107100
Parallel/Sequential Speed Up :	Stehfest	6	21	5.65549822220
Parallel/Sequential Speed Up/Processor :	Stehfest	6	5	0.56982627545

Table 7.8 Optimal Parallel Programs Data (Boyle and Vorst)

7.4.4.4 Modified Volatility Model : Barles and Soner

7.4.4.4.1 Sequential Program Data

Table 7.9 below gives the data collected using the sequential, finite difference program.

	Value
NRMSD :	0.08974615390
Minimum Wall Time (s) :	3.91849494000

Table 7.9 Sequential, Finite Difference Program Data (Barles and Soner)

7.4.4.4.2 Part 1 - Varying the Number of Weights/Terms Used

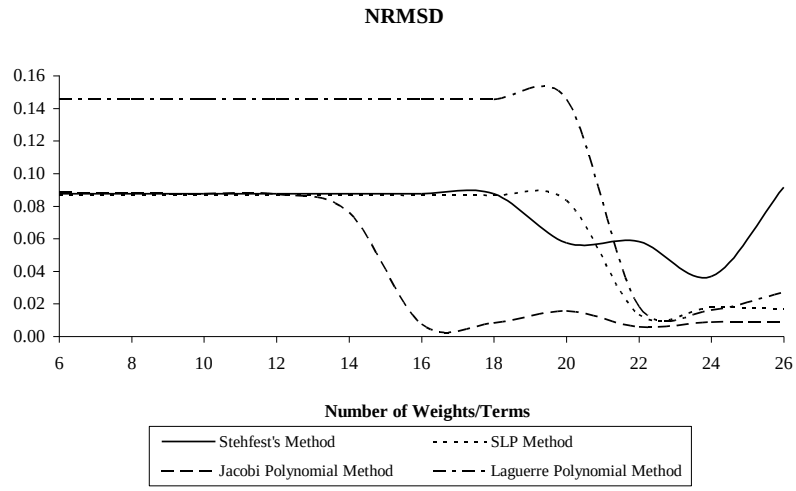

Figure 7.43 Normalised Root Mean Square Deviation, Parallel Programs (Barles and Soner)[83]

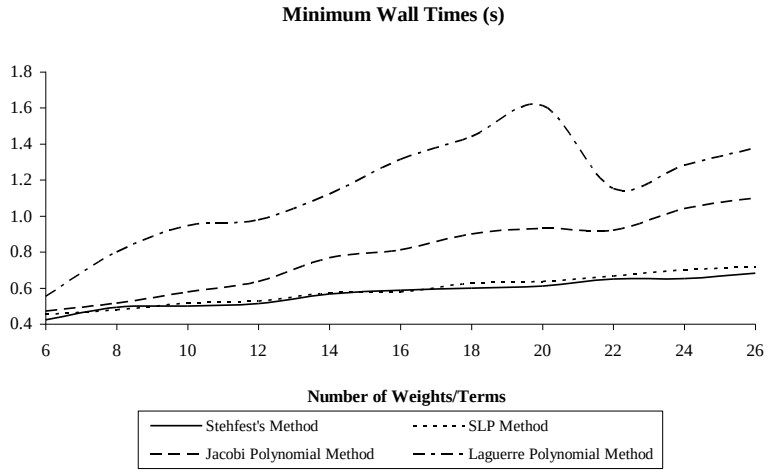

Figure 7.44 Minimum Wall Times, Parallel Programs (Barles and Soner)

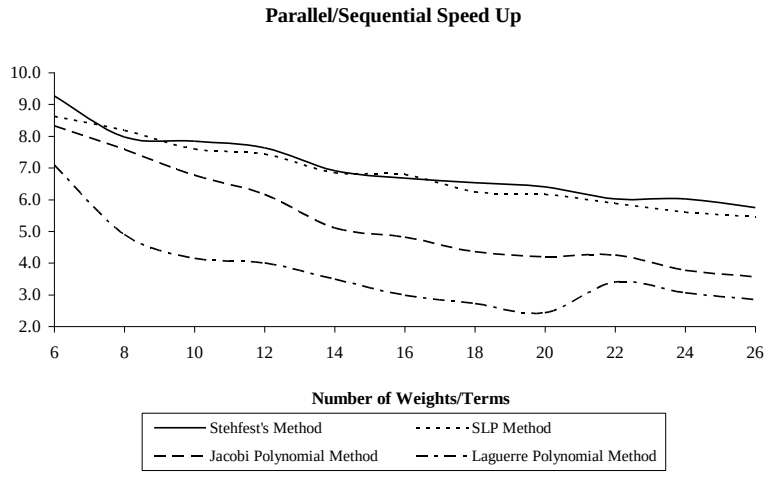

Figure 7.45 Parallel/Sequential Speed Up (Barles and Soner)

[83] The NRMSD values for all numerical inversion algorithms follow oscillating trends.

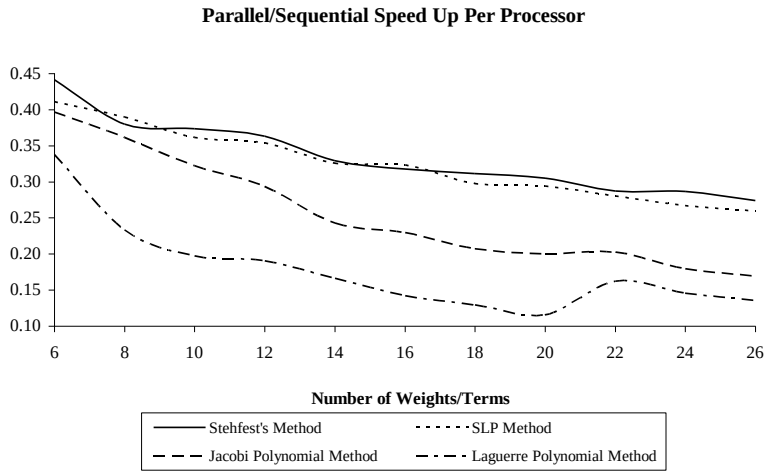

Figure 7.46 Parallel/Sequential Speed Up Per Processor (Barles and Soner)

7.4.4.4.3 Part 2 - Varying the Number of Processors Used

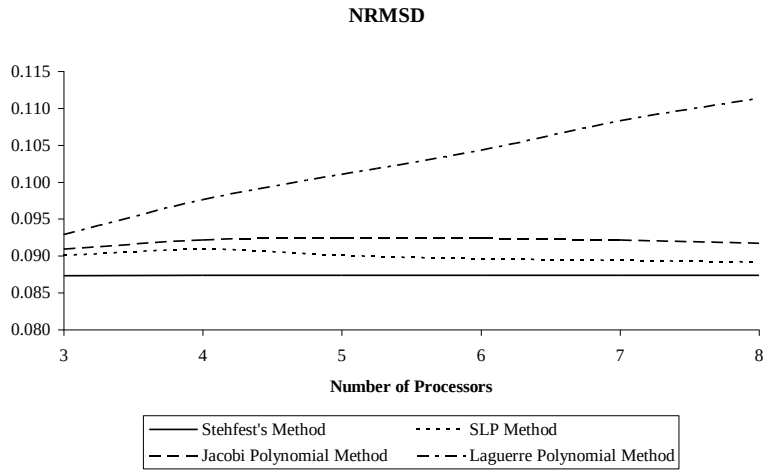

Figure 7.47 Normalised Root Mean Square Deviation (Barles and Soner) (3-8 Processors)[84]

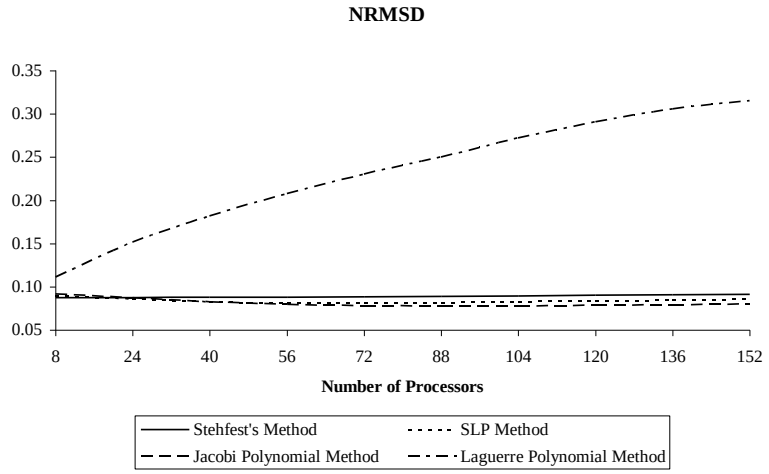

Figure 7.48 Normalised Root Mean Square Deviation (Barles and Soner) (8-152 Processors)[85]

[84] The NRMSD values for Stehfest's method follow a slight upward trend. For the SLP method and the Jacobi polynomial method the NRMSD values oscillate slightly.

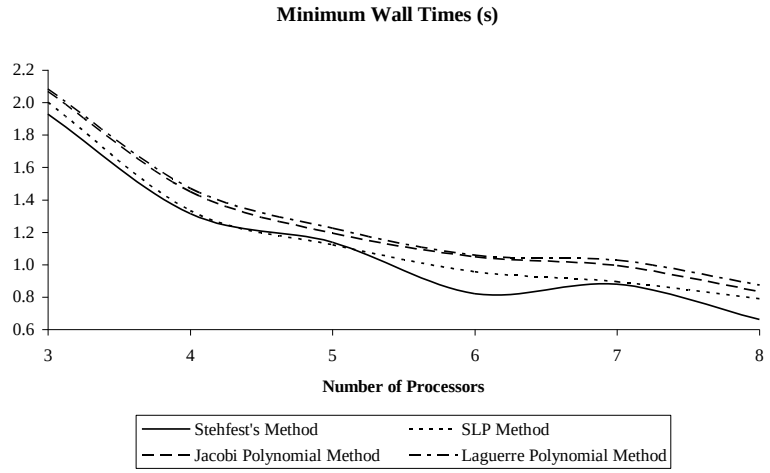

Figure 7.49 Minimum Wall Times (Barles and Soner) (3-8 Processors)

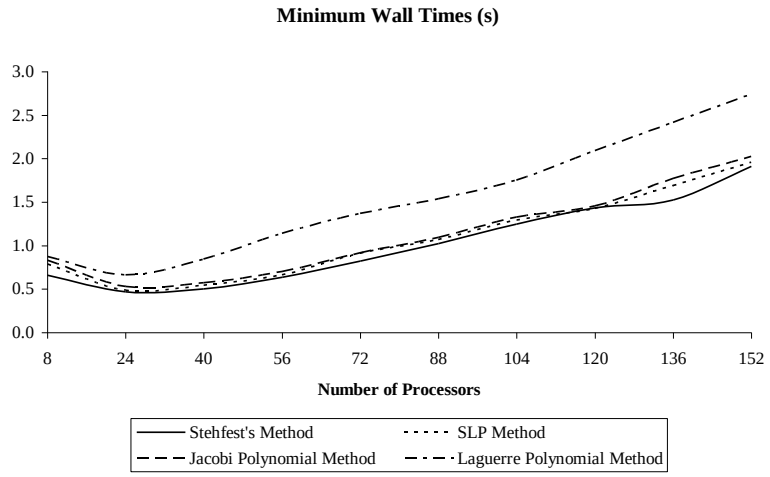

Figure 7.50 Minimum Wall Times (Barles and Soner) (8-152 Processors)

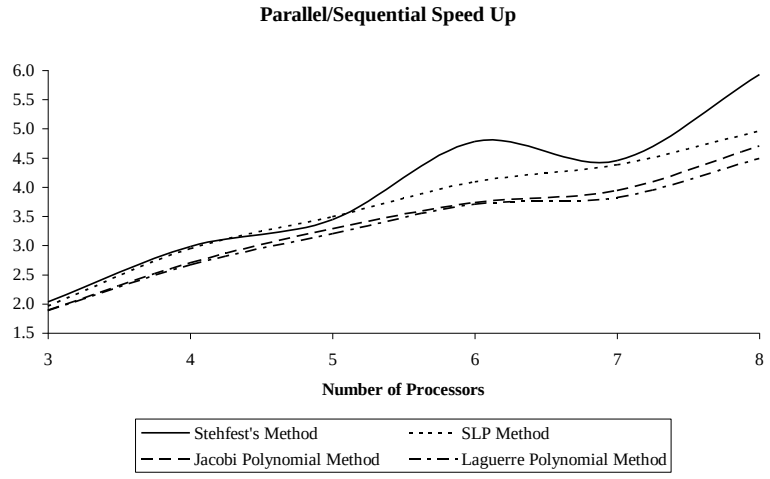

Figure 7.51 Parallel/Sequential Speed Up (Barles and Soner) (3-8 Processors)

---

[85] The NRMSD values for Stehfest's method follow a slight upward trend. For the SLP method and the Jacobi polynomial method the NRMSD values oscillate slightly.

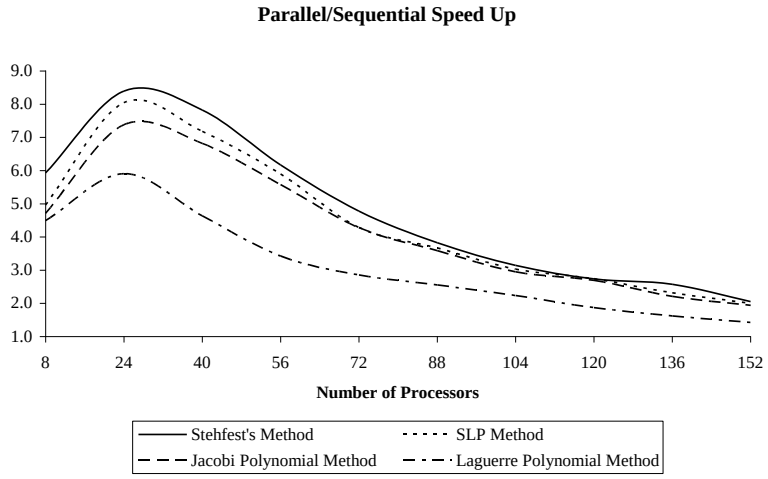

Figure 7.52 Parallel/Sequential Speed Up (Barles and Soner) (8-152 Processors)

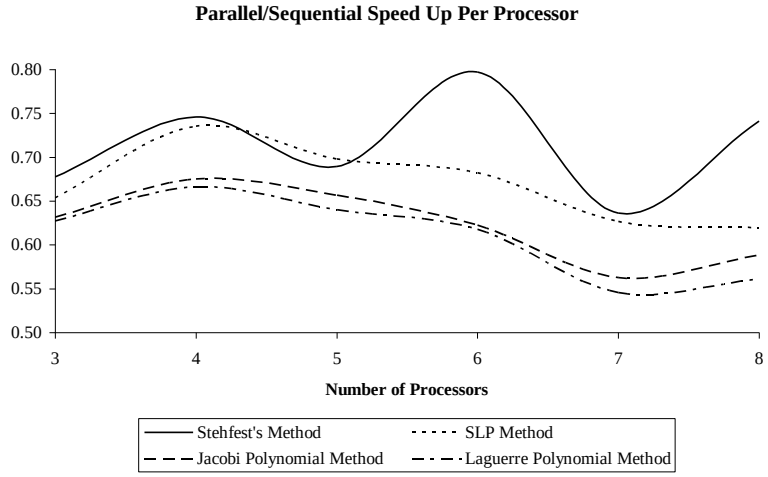

Figure 7.53 Parallel/Sequential Speed Up Per Processor (Barles and Soner) (3-8 Processors)

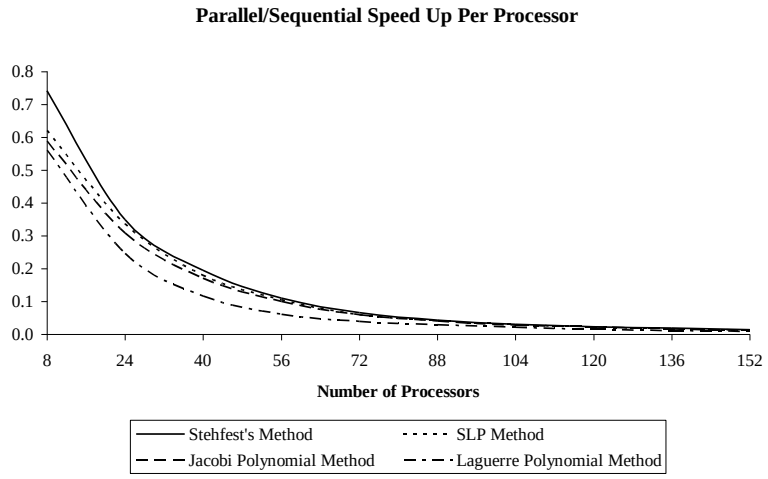

Figure 7.54 Parallel/Sequential Speed Up Per Processor (Barles and Soner) (8-152 Processors)

7.4.4.4.4 Optimal Parallel Programs Data

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Processors	Optimal Value
NRMSD :	Jacobi	16	21	0.00743902902
Minimum Wall Time (s) :	Stehfest	6	21	0.42293095590
Parallel/Sequential Speed Up :	Stehfest	6	21	9.26509371172
Parallel/Sequential Speed Up/Processor :	Stehfest	6	6	0.79714300681

Table 7.10 Optimal Parallel Programs Data (Barles and Soner)

7.4.4.5 Modified Volatility Model : Risk Adjusted Pricing Methodology

7.4.4.5.1 Sequential Program Data

Table 7.11 below gives the data collected using the sequential, finite difference program.

	Value
NRMSD :	0.08908669343
Minimum Wall Time (s) :	1.17682810000

Table 7.11 Sequential, Finite Difference Program Data (RAPM)

7.4.4.5.2 Part 1 - Varying the Number of Weights/Terms Used

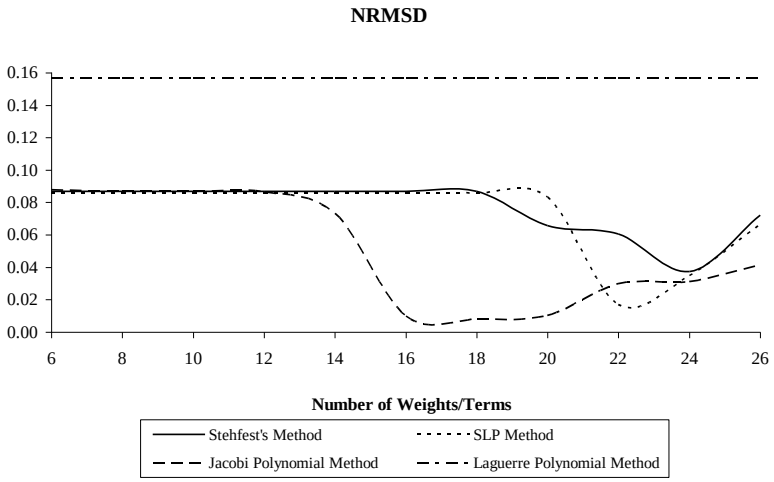

Figure 7.55 Normalised Root Mean Square Deviation, Parallel Programs (RAPM)[86]

[86] The NRMSD values for all numerical inversion algorithms follow oscillating trends.

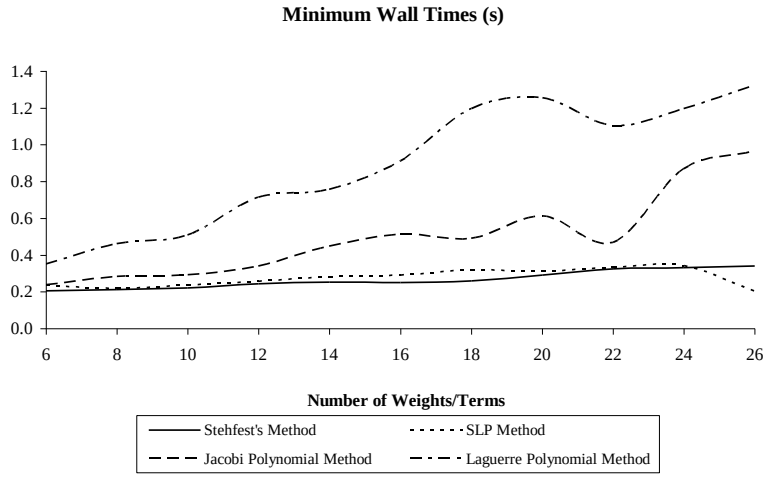

Figure 7.56 Minimum Wall Times, Parallel Programs (RAPM)

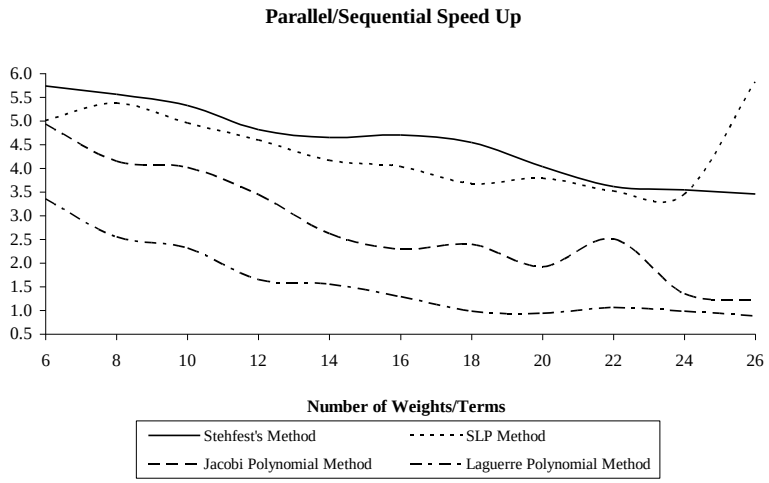

Figure 7.57 Parallel/Sequential Speed Up (RAPM)

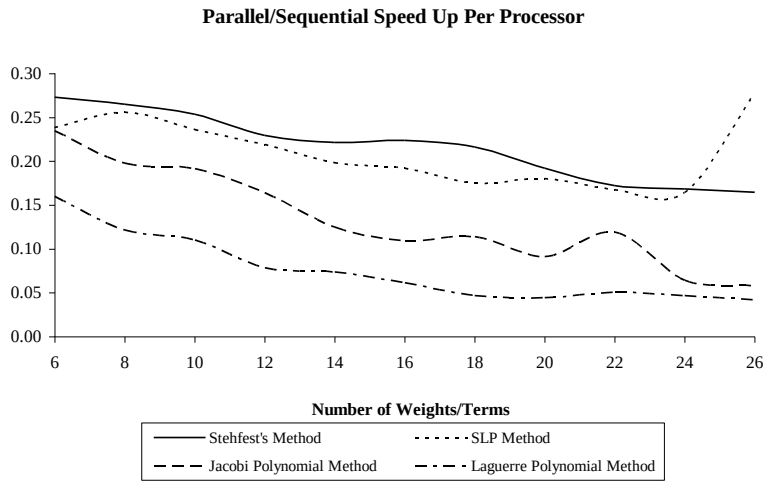

Figure 7.58 Parallel/Sequential Speed Up Per Processor (RAPM)

7.4.4.5.3 Part 2 - Varying the Number of Processors Used

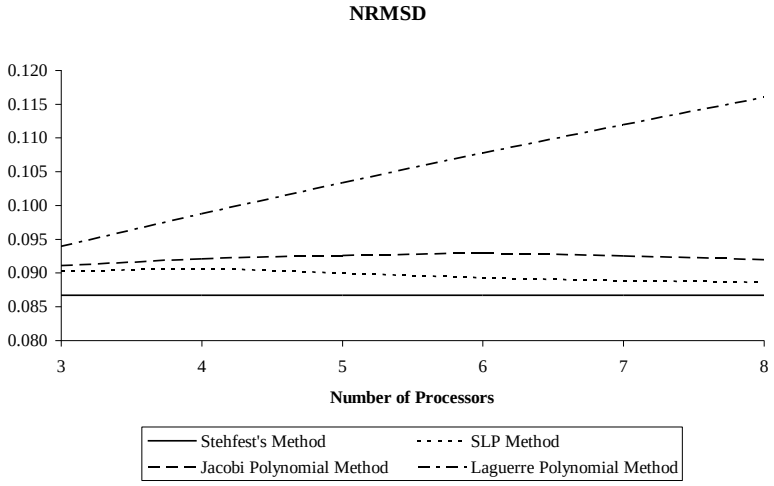

Figure 7.59 Normalised Root Mean Square Deviation (RAPM) (3-8 Processors)[87]

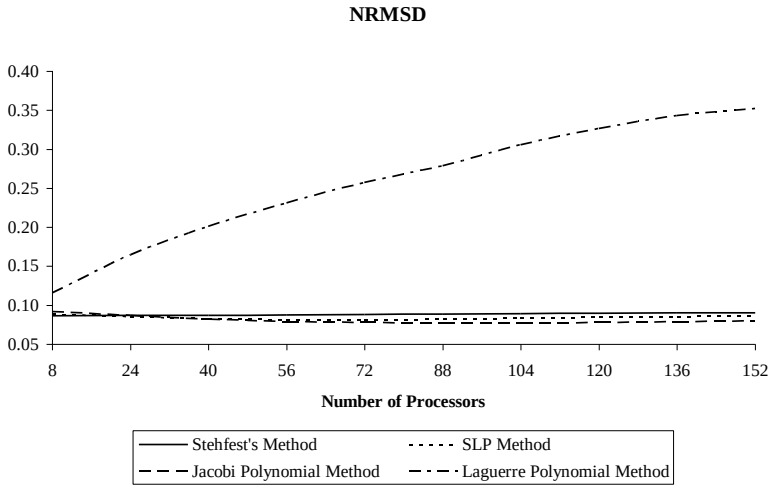

Figure 7.60 Normalised Root Mean Square Deviation (RAPM) (8-152 Processors)[88]

[87] The NRMSD values for Stehfest's method follow a slight upward trend. For the SLP method and the Jacobi polynomial method the NRMSD values oscillate slightly.

[88] The NRMSD values for Stehfest's method, the SLP method and the Jacobi polynomial method follow slight oscillating trends.

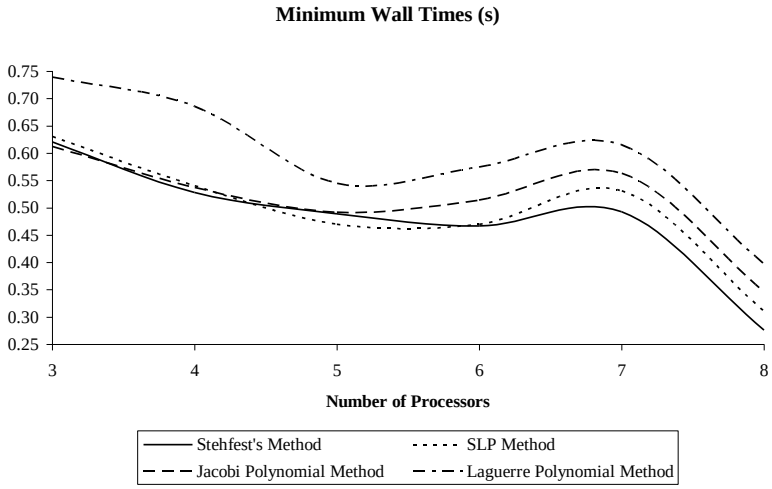

Figure 7.61 Minimum Wall Times (RAPM) (3-8 Processors)

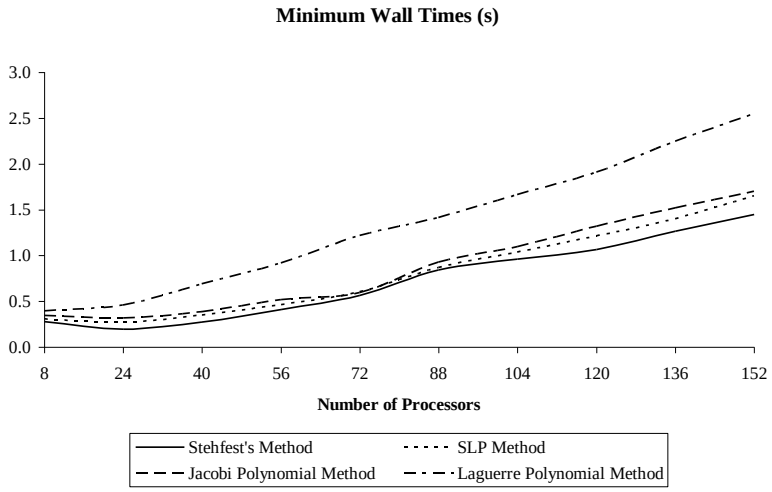

Figure 7.62 Minimum Wall Times (RAPM) (8-152 Processors)

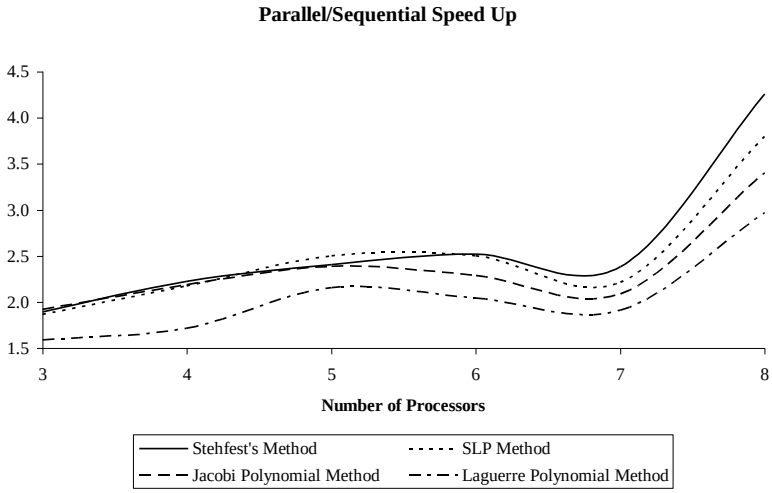

Figure 7.63 Parallel/Sequential Speed Up (RAPM) (3-8 Processors)

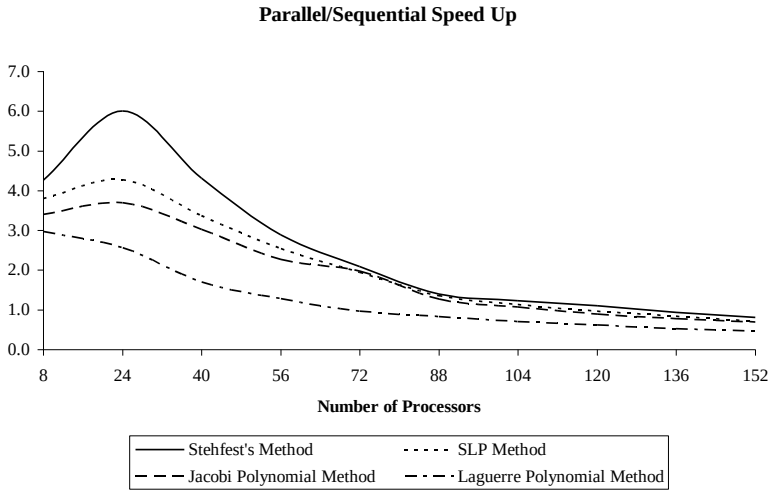

Figure 7.64 Parallel/Sequential Speed Up (RAPM) (8-152 Processors)

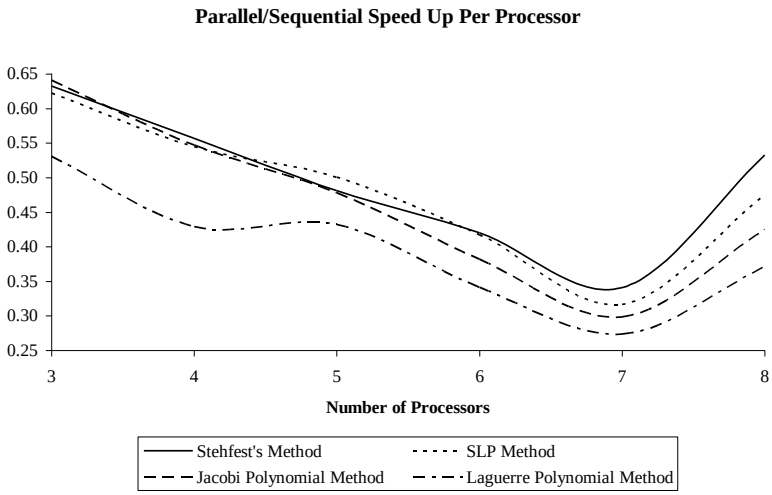

Figure 7.65 Parallel/Sequential Speed Up Per Processor (RAPM) (3-8 Processors)

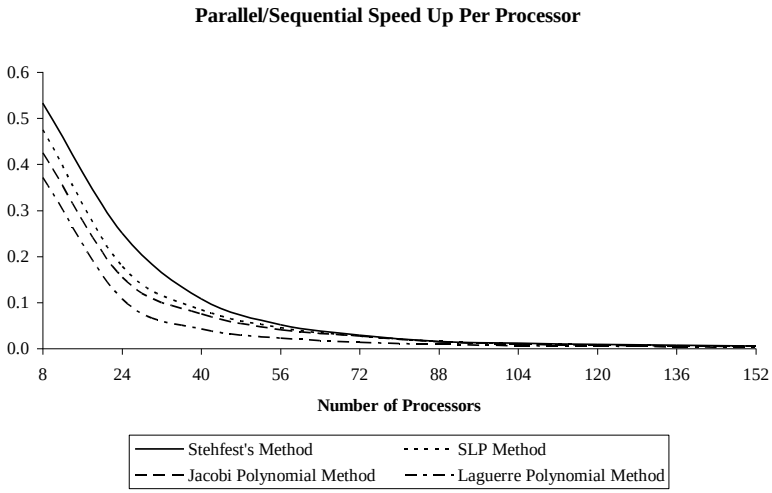

Figure 7.66 Parallel/Sequential Speed Up Per Processor (RAPM) (8-152 Processors)

7.4.4.5.4 Optimal Parallel Programs Data

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Processors	Optimal Value
NRMSD :	Jacobi	18	21	0.00801089964
Minimum Wall Time (s) :	Stehfest	6	24	0.19593906400
Parallel/Sequential Speed Up :	Stehfest	6	24	6.00609228183
Parallel/Sequential Speed Up/Processor :	Jacobi	6	3	0.64110825564

Table 7.12 Optimal Parallel Programs Data (RAPM)

7.4.5 Conclusions

Tables 7.4, 7.6, 7.8, 7.10 and 7.12 above give the optimal number of weights/terms and processors to use in/with the LTFD algorithm when it is used to solve one-dimensional nonlinear Black-Scholes equations containing the modified volatility models proposed by Lai *et al.*, Leland, Boyle and Vorst, Barles and Soner and Kratka (*i.e.* the Risk Adjusted Pricing Methodology model).

It can be seen from the graphs and tables in this chapter that :

- the LTFD algorithm is faster and generally more accurate, relative to the analytical or reference solutions, than the sequential finite difference algorithm. Whenever the LTFD solutions are less accurate than the finite difference solutions, the differences are small
- the nonlinear data exhibits behaviour seen in earlier investigations. This behaviour is described and explained in Chapter 10
- the *best* all-round numerical algorithm for inverting the Laplace transforms arising in the solution of one-dimensional, nonlinear Black-Scholes equations is Stehfest's method. Whenever another inversion algorithm performs better than Stehfest, the difference is marginal.

7.4.6 The Numerical Solutions

Figure 7.67 and Figure 7.68 show the numerical solutions of the nonlinear Black-Scholes equations in the case when $S = 1$. The solutions were calculated using the LTFD algorithm. The inverse Laplace transform values were calculated using Stehfest's method with 6 weights and 21 processors.

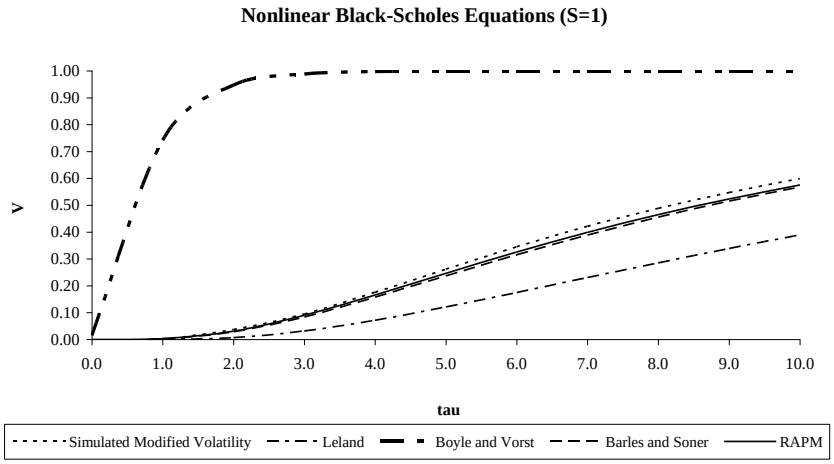

Figure 7.67 Numerical Solutions of the Nonlinear Black-Scholes Equations for $\tau \in [0,10]$

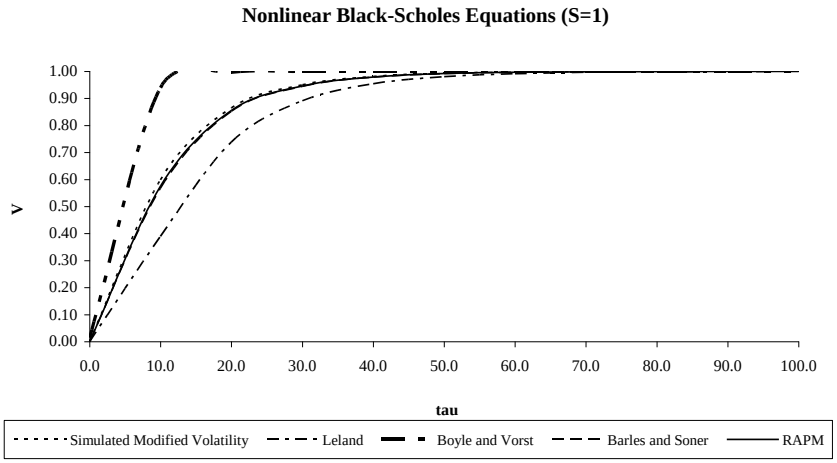

Figure 7.68 Numerical Solutions of the Nonlinear Black-Scholes Equations for $\tau \in [0,100]$

The behaviour shown is the same for all other values of *S i.e.* as $\tau \to \infty$, $V \to S$. With the exception of the graph for Boyle and Vorst, the solutions exhibit similar behaviour. Initially, the graphs have the characteristic *hockey stick* shape associated with European call options (Wilmot, 2000) and as the expiry date of the option approaches, the value of the option approaches the value of the underlying asset (as predicted by the boundary condition of the Black-Scholes equation (32)). In the case of the Boyle and Vorst solution the value of the option appears to approach the value of the underlying asset much more quickly. The likely

reason for this is that the volatility in this case is much larger than in the other solutions. When the volatility is large the second derivative in the Black-Scholes equation (32) becomes dominant so that it behaves like the diffusion equation. It is known that when the diffusivity in equations of this type is large, the dependent variable increases quickly over time. It appears that the same behaviour is occurring here.

7.5 Chapter Summary

This chapter has shown that a modified version of the LTFD algorithm described earlier can be used to solve one-dimensional, nonlinear Black-Scholes equations and that accurate reference solutions can be calculated using Monte Carlo methods. The next question to be answered is *can Laplace transform - finite difference algorithms be used to solve two-dimensional Black-Scholes equations* ?

7.6 Contribution to Knowledge

This chapter has :

- explained how Monte Carlo methods can be used for calculating accurate reference solutions of one-dimensional, nonlinear Black-Scholes equations
- developed and evaluated a parallel, Laplace transform-finite difference algorithm for solving one-dimensional, nonlinear Black-Scholes equations and shown that this algorithm produces fast and accurate solutions
- used the LTFD algorithm for solving one-dimensional nonlinear Black-Scholes equations containing the modified volatility models proposed by Leland, Boyle and Vorst, Barles and Soner and Kratka (*i.e.* the Risk Adjusted Pricing Methodology model)
- evaluated Stehfest's method, the shifted Legendre polynomial method, the Jacobi polynomial method and the Laguerre polynomial method when these methods are used to invert the Laplace transforms arising in the LTFD algorithm
- determined the optimal number of weights/terms and processors to use with each of the numerical inversion algorithms when they are used in the LTFD algorithm
- established the advantages of using the LTFD algorithm for solving one-dimensional nonlinear Black-Scholes equations.

Chapter 8

The Two-Dimensional, Laplace Transform-Finite Difference Algorithm

8.0 Introduction

The previous investigative chapters of this dissertation have shown that LTFD algorithms can be used for solving one-dimensional, linear and nonlinear Black-Scholes equations. For two-dimensional Black-Scholes equations, *i.e.* equations that model financial options written on two underlying assets, the solution domain, for a range of τ values, is three-dimensional. However, this chapter will show that the Laplace transform-finite difference approach can also be used for calculating option prices in this more complicated case.

8.1 Background

Black-Scholes equations that model financial options written on more than two underlying assets must be solved using Monte Carlo methods, Wilmot (2000). However, in the two-dimensional case a range of numerical methods can be used for finding the solution.

A legitimate criticism of finite difference methods for solving two-dimensional diffusion equations is that they can be inaccurate. To avoid this inaccuracy very small step lengths must be used and this increases the amount of computational effort required to find the solution. For example, suppose that the explicit finite difference method is used to solve the two-dimensional heat equation :

$$\frac{\partial u}{\partial t} = a\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \quad a \in \mathbb{R}$$

on an unit square defined by $0 < x < 1, 0 < y < 1$. Sharcnet (2008) shows that to produce an accurate solution, the time step δt must be chosen so that :

$$\delta t \leq \frac{1}{2a}\left(\frac{\delta x^2 \delta y^2}{\delta x^2 + \delta y^2}\right)$$

Here, δx and δy are the step lengths in the *x* and *y* directions respectively. For example, if $\delta x = \delta y = 0.05$ then the maximum value of δt is 0.000625.

The advantage of using Laplace transform based algorithms for solving problems of this type is that the time domain is removed from the equation. This eases the restrictions that must be

placed upon the step lengths. For example, Morton and Mayers (2008) show that if the time domain is removed from the two-dimensional heat equation above, the explicit finite difference solution is accurate provided that $\delta x \leq 0.25$ and $\delta y \leq 0.25$. However, the disadvantages of Laplace transform based algorithms are that additional computational effort is required to find the inverse Laplace transforms and that Laplace transform inversion is itself, ill-posed.

Parallel, Laplace transform based algorithms for solving two-dimensional, linear Black-Scholes equations have been developed previously. For example, Lee and Sheen (2009) describe an algorithm in which Laplace transform/contour integral methods are used to decompose an equation of this type into an equivalent system of elliptic PDEs. These equations are independent and can therefore be solved on separate processors. However, their algorithm is complicated and is unlikely to be competitive with the LTFD algorithm described here. Furthermore, the authors have not shown how their algorithm can be used for solving two-dimensional, nonlinear Black-Scholes equations *i.e.* those involving transaction costs.

8.2 The Algorithm

The forward, two-dimensional, linear Black-Scholes equation is :

$$-\frac{\partial V}{\partial \tau}+\frac{1}{2}\sigma_1^2 S_1^2\frac{\partial^2 V}{\partial S_1^2}+\frac{1}{2}\sigma_2^2 S_2^2\frac{\partial^2 V}{\partial S_2^2}+\sigma_1\sigma_2\rho_{12}S_1S_2\frac{\partial^2 V}{\partial S_1\partial S_2}+rS_1\frac{\partial V}{\partial S_1}+rS_2\frac{\partial V}{\partial S_2}-rV=0 \quad \text{---- (26)}$$

Subject to the initial condition :

$$V\left(S_1,S_2,0\right)=\max\left(\max\left(S_1,S_2\right)-E,0\right) \quad \text{---- (27)}$$

and the boundary conditions :

$$V\left(S_1,0,\tau\right)=Vs_1\,\tau \quad \text{---- (28)} \qquad V\left(0,S_2,\tau\right)=Vs_2\,\tau \quad \text{---- (29)}$$

$$V\left(S_1,S_2,\tau\right)\approx S_1 \text{ as } S_1\to\infty \quad \text{---- (30)} \qquad V\left(S_1,S_2,\tau\right)\approx S_2 \text{ as } S_2\to\infty \quad \text{---- (31)}$$

To solve this equation using the Laplace transform-finite difference algorithm :

- take Laplace transforms with respect to τ. Equation (26) then becomes :

$$-\lambda\bar{V}+V_{0,0}+\frac{1}{2}\sigma_1^2 S_1^2\frac{\partial^2 \bar{V}}{\partial S_1^2}+\frac{1}{2}\sigma_2^2 S_2^2\frac{\partial^2 \bar{V}}{\partial S_2^2}+\sigma_1\sigma_2\rho_{12}S_1S_2\frac{\partial^2 \bar{V}}{\partial S_1\partial S_2}+rS_1\frac{\partial \bar{V}}{\partial S_1}+rS_2\frac{\partial \bar{V}}{\partial S_2}-r\bar{V}=0$$

$$i.e.\ -\lambda\bar{V} + \frac{1}{2}\sigma_1^2 S_1^2 \frac{\partial^2 \bar{V}}{\partial S_1^2} + \frac{1}{2}\sigma_2^2 S_2^2 \frac{\partial^2 \bar{V}}{\partial S_2^2} + \sigma_1\sigma_2\rho_{12} S_1 S_2 \frac{\partial^2 \bar{V}}{\partial S_1 \partial S_2} + rS_1 \frac{\partial \bar{V}}{\partial S_1} + rS_2 \frac{\partial \bar{V}}{\partial S_2} - r\bar{V} = 0^{89}$$

---- (68)

The initial condition (27) becomes :

$$\bar{V}(S_1, S_2, 0) = \max\left(\max\left(\frac{S_1}{\lambda}, \frac{S_2}{\lambda}\right) - \frac{E}{\lambda}, 0\right)$$

and the boundary conditions (28), (29), (30) and (31) become :

$$\bar{V}(S_1, 0, \tau) = \bar{V}s_1\,\tau \qquad \bar{V}(0, S_2, \tau) = \bar{V}s_2\,\tau$$

$$\bar{V}(S_1, S_2, \tau) \approx \frac{S_1}{\lambda} \text{ as } S_1 \to \infty \qquad \bar{V}(S_1, S_2, \tau) \approx \frac{S_2}{\lambda} \text{ as } S_2 \to \infty$$

where λ is the transform variable and $\bar{V}s_1\,\tau$ and $\bar{V}s_2\,\tau$ are the Laplace transforms of the solutions of the one-dimensional, linear Black-Scholes equation (10) written on the first and second assets respectively. See Chapter 2 for further details. The easiest way to find these values is to use the Laplace transform of the analytical solution *i.e.* expression (44)

- create and initialise the solution domain in Laplace space. For a particular value of λ, the initial solution domain for equation (68) is :

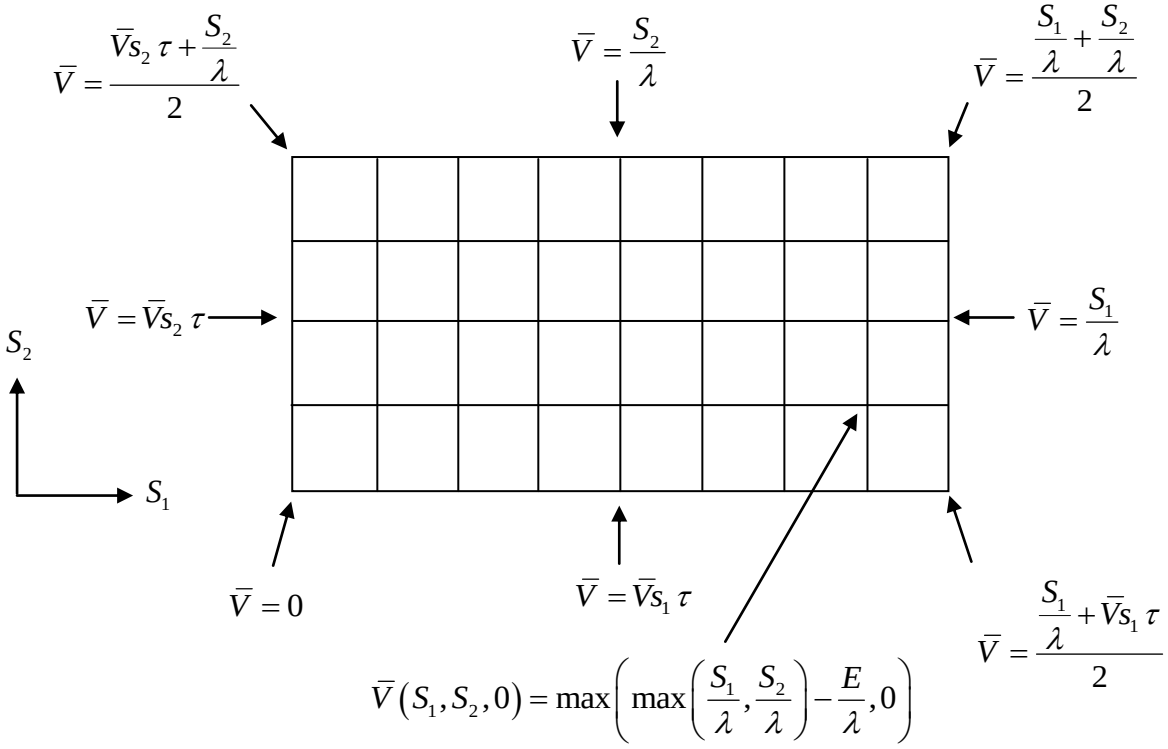

Figure 8.1 The Initial Solution Domain in Laplace Space for the Two-Dimensional, Linear Black-Scholes Equation

---

89 Since $V_{0,0} = 0$.

- use finite difference methods to calculate the internal $\bar{V}$ values

In Chapter 6 it was shown that the most computationally efficient finite difference method for solving the one-dimensional, linear Black-Scholes equation (10) is the explicit method. This procedure will also be used in the two-dimensional case. In this method first partial derivatives are replaced with forward difference approximations and second partial derivatives are replaced with central difference approximations. Hence, if *h* is the step length in the S_1 and S_2 directions then :

$$\frac{\partial \bar{V}}{\partial S_1} \approx \frac{\bar{V}_{i+1,j} - \bar{V}_{i,j}}{h} \qquad \frac{\partial \bar{V}}{\partial S_2} \approx \frac{\bar{V}_{i,j+1} - \bar{V}_{i,j}}{h}$$

$$\frac{\partial^2 \bar{V}}{\partial S_1^2} \approx \frac{\bar{V}_{i+1,j} - 2\bar{V}_{i,j} + \bar{V}_{i-1,j}}{h^2} \qquad \frac{\partial^2 \bar{V}}{\partial S_2^2} \approx \frac{\bar{V}_{i,j+1} - 2\bar{V}_{i,j} + \bar{V}_{i,j-1}}{h^2}$$

Smith (2004) also shows that the mixed partial derivative can be approximated as :

$$\frac{\partial^2 \bar{V}}{\partial S_1 \partial S_2} \approx \frac{\bar{V}_{i+1,j+1} - \bar{V}_{i+1,j-1} - \bar{V}_{i-1,j+1} + \bar{V}_{i-1,j-1}}{4h^2}$$

Substituting these approximations into equation (68) :

$$-\lambda \bar{V} + \frac{1}{2}\sigma_1^2 S_1^2 \left(\frac{\bar{V}_{i+1,j} - 2\bar{V}_{i,j} + \bar{V}_{i-1,j}}{h^2} \right) + \frac{1}{2}\sigma_2^2 S_2^2 \left(\frac{\bar{V}_{i,j+1} - 2\bar{V}_{i,j} + \bar{V}_{i,j-1}}{h^2} \right) +$$
$$\sigma_1 \sigma_2 \rho_{12} S_1 S_2 \left(\frac{\bar{V}_{i+1,j+1} - \bar{V}_{i+1,j-1} - \bar{V}_{i-1,j+1} + \bar{V}_{i-1,j-1}}{4h^2} \right) +$$
$$rS_1 \left(\frac{\bar{V}_{i+1,j} - \bar{V}_{i,j}}{h} \right) + rS_2 \left(\frac{\bar{V}_{i,j+1} - \bar{V}_{i,j}}{h} \right) - r\bar{V} = 0 \quad \text{---- (69)}$$

One way of calculating the internal $\bar{V}$ values is to use expression (69) to form a system of linear equations and to solve this system using an appropriate numerical method. In the case where $\rho_{12} = 0$, the matrix of coefficients in the system of equations is tridiagonal and the system can be solved efficiently using the Thomas algorithm. However, this approach has two major disadvantages. Firstly, it requires the solution domain to be square, which is not always the case, and secondly, if $\rho_{12} \neq 0$, the tridiagonal structure is lost and the system of equations becomes more difficult and time-consuming to solve. To overcome these problems the approach used by authors such as Horak and Gruber (2002)

can be used. When solving two-dimensional diffusion equations of this type they use an iterative procedure, similar to the Jacobi method for solving systems of linear equations, to calculate the values of the dependent variables *i.e.* a procedure in which the new values of these variables are calculated using their values at the previous iteration.

Rearranging expression (69) the required iterative formula in this case becomes :

$$\bar{V}_{new_{i,j}} = \frac{1}{\left(-\lambda - 2a - 2b - d - e - r\right)}\left(-c\bar{V}_{old_{i+1,j+1}} - \left(a+d\right)\bar{V}_{old_{i+1,j}} + c\bar{V}_{old_{i+i,j-1}} - \left(b+e\right)\bar{V}_{old_{i,j+1}} - b\bar{V}_{old_{i,j-1}} + c\bar{V}_{old_{i-1,j+1}} - a\bar{V}_{old_{i-1,j}} - c\bar{V}_{old_{i-1,j-1}}\right) \quad \text{---- (70)}$$

i.e. $\bar{V}_{new} = f\left(\bar{V}_{old}\right)$ where :

$$a = \frac{\sigma_1^2 S_1^2}{2h^2}, \quad b = \frac{\sigma_2^2 S_2^2}{2h^2}, \quad c = \frac{\sigma_1\sigma_2\rho_{12}S_1S_2}{4h^2}, \quad d = \frac{rS_1}{h} \quad \text{and} \quad e = \frac{rS_2}{h}$$

The iterative formula (70) is applied until the accuracy criterion :

$$\left|\frac{\max\left(\text{abs}\left(\bar{V}_{new_{i,j}} - \bar{V}_{old_{i,j}}\right)\right)}{\max\left(\text{abs}\left(\bar{V}_{new_{i,j}} + \bar{V}_{old_{i,j}}\right)\right)}\right| < \varepsilon$$

is satisfied. Here, *i* and *j* increment the values of S_1 and S_2 and ε is a small positive number

- find the inverse Laplace transforms of the $\bar{V}$ values to give the option values.

  If the inverse Laplace transforms are found using a numerical inversion algorithm, the solution domain in Laplace space will be three-dimensional *i.e.* initially, there will be a grid *i.e.* rank in the form of Figure 8.1 for each value of the transform variable. For example, using Stehfest's method with $m = 6$ this solution domain will have the form :

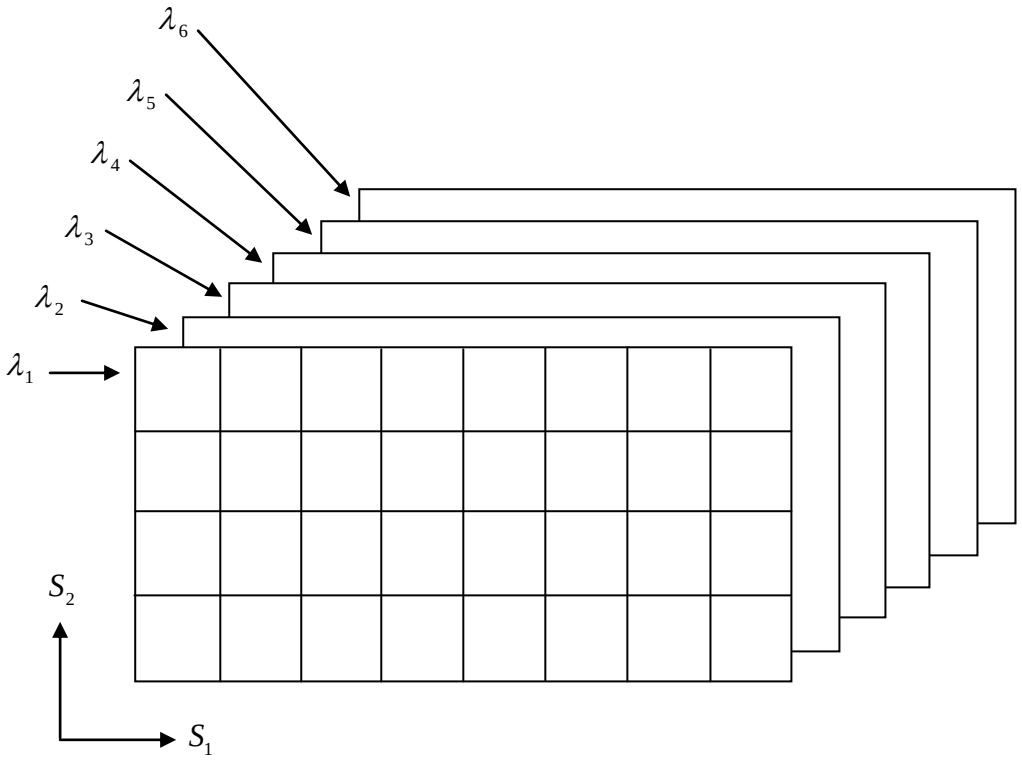

Figure 8.2 The Decomposed Solution Domain in Laplace Space for Two-Dimensional Black-Scholes Equations

The option values are found by applying the numerical inversion algorithm to the corresponding $\bar{V}$ values in each rank. Using the example in Figure 8.2 above, this operation can be visualised as follows :

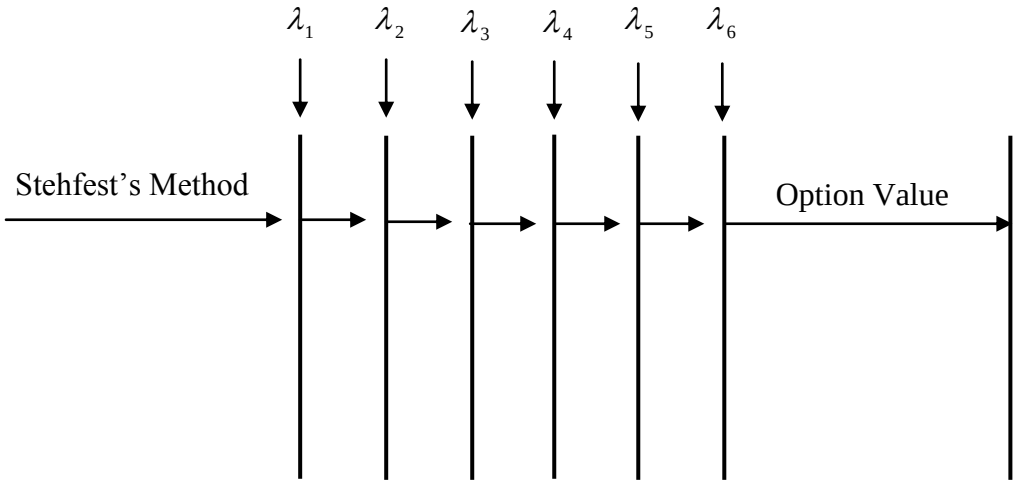

Figure 8.3 Calculating the Option Values in the Two-Dimensional LTFD Algorithm

8.3 The Sequential LTFD Algorithm

In the sequential LTFD algorithm the ranks within the solution domain are processed in order. Once the final rank has been solved in Laplace space, the numerical inversion algorithm is used to calculate the option values. Finally, the sequential algorithm reads in the reference solution, calculates the NRMSD value and then calculates and stores the program wall time.

8.4 The Parallel LTFD Algorithm

In the parallel LTFD algorithm each slave processor is responsible for processing one rank within the solution domain. Each rank is then passed back to the master processor. Once all ranks have been received, the master processor uses the numerical inversion algorithm to calculate the option values and then calculates and stores the speed and accuracy data.

8.5 Calculating the Reference Solution

No analytical solutions are available for solving two-dimensional Black-Scholes equations. To calculate the accurate reference solutions required for calculating the NRMSD in the two-dimensional case Monte Carlo algorithms must be used.

8.5.1 The Monte Carlo Algorithm for Solving Two-Dimensional, Linear Black-Scholes Equations

Suppose that the basket option contains *d* underling assets. Then, the future value of the *i*th underlying asset is given by the stochastic differential equation :

$$dS_{i,\tau} = rS_i d\tau + \sigma_i S_i dW_i(\tau) \quad \text{---- (71)}$$

where *r* is the risk-free interest rate, S_i is the current value of the *i*th underlying asset, τ is the time to expiry, σ_i is the volatility of the *i*th underlying asset and $W_i(\tau)$ is a Brownian motion. The solution of equation (71) is :

$$S_{i,\tau} = S_i \exp\left(\left[r - \frac{1}{2}\sigma_i^2\right]\tau + \sigma_i W_i(\tau)\right) \quad \text{---- (72)}$$

In the multi-dimensional case it is assumed that the future value of the basket option depends upon the average of the future values of the underlying assets *i.e.*

$$\frac{1}{d}\sum_{j=1}^{d} S_{j,\tau}$$ [90]

[90] If some underlying assets have more influence on the future value of the option than others, a weighted average of the future values is used.

The payoff of a basket option at the expiry time is :

$$\max\left(\frac{1}{d}\sum_{j=1}^{d} S_{j,\tau} - E, 0\right) \quad \text{---- (73)}$$

where *E* is the exercise price. The seller of the option will wish to know the current value of the option. This can be calculated by pre-multiplying expression (73) by the discount factor $e^{-r\tau}$. Hence, the current value of the basket option *V* is :

$$V = e^{-r\tau} \max\left(\frac{1}{d}\sum_{j=1}^{d} S_{j,\tau} - E, 0\right) \quad \text{---- (74)}$$

The problem in the multi-dimensional case is that the values of $x_i = \sigma_i W_i(\tau)$ in expression (71) are correlated *i.e.*

$$\mathrm{E}\left(x_i x_j\right) = \rho_{ij}$$

where ρ_{ij} is the correlation coefficient between the *i*th and *j*th underlying assets. To calculate the option values in this case correlated standard normal values must be used. To find these values :

- decompose the correlation matrix into a product :

  $$\Sigma = MM^T$$

  where *M* is a lower triangular matrix
- let $\underline{y}$ be a column vector containing *d* uncorrelated standard normal random variables Z_i
- let $\underline{\phi}$ be a column vector defined as $\underline{\phi} = M\underline{y}$.

Then, the vector $\underline{\phi}$ contains the correlated standard normal random variables required.

Proof

$$\mathrm{E}\left(\underline{\phi}\,\underline{\phi}^T\right) = \mathrm{E}\left(M\underline{y}\,\underline{y}^T M^T\right) = M\mathrm{E}\left(\underline{y}\,\underline{y}^T\right)M^T = MIM^T = MM^T = \Sigma$$

Glasserman (2003).

Using the vector $\underline{\phi}$ the future value of the *i*th underlying asset is given by :

$$S_{i,\tau} = S_i \exp\left(\left[r - \frac{1}{2}\sigma_i^2\right]\tau + \phi_i\right) \quad \text{----(75)}$$

The decomposition of the correlation matrix is non-unique. The most commonly used method

is to decompose Σ into its Cholesky factors *i.e.* to write :

$$\Sigma = LL^T$$

Expressions (74) and (75) form the basis of the Monte Carlo algorithm. Given a mechanism for generating correlated standard normal random variables ϕ_i, expressions (74) and (75) can be used to produce sample values of *V* that can then be averaged to give an estimate of the current value of the basket option. More formally, the algorithm can be written as :

decompose $\Sigma = LL^T$

loop i = 1 to n

loop j=1 to d

generate $y_j = Z_j$ [91]

end loop

$\underline{\phi} = L\underline{y}$

loop k=1 to d

$$S_{k,\tau} = S_k \exp\left(\left[r - \frac{1}{2}\sigma_k^2\right]\tau + \phi_k\right)$$

end loop

$$V_i = e^{-r\tau} \max\left(\frac{1}{d}\sum_{j=1}^{d} S_{j,\tau} - E, 0\right)$$

end loop

$$\hat{V} = \frac{\left(V_1 + V_2 + V_3 + \ldots + V_n\right)}{n} = \frac{1}{n}\sum_{i=1}^{n} V_i$$

Figure 8.4 The Monte Carlo Algorithm for Solving Two-Dimensional, Linear Black-Scholes Equations[92]

Glasserman (2003) shows that by the strong law of large numbers[93], the estimator $\hat{V}$ is unbiased *i.e.* $\mathrm{E}\left(\hat{V}\right) = V$ and strongly consistent *i.e.* as $n \to \infty$, $\hat{V} \to V$ with a probability of one.

---

91 The Z_j values can be generated in the same way as the standard normal random values in the one-dimensional Monte Carlo algorithm.

92 This algorithm was developed by Wilmot *et al.* (1999) and Glasserman (2003).

93 The average of the results obtained from a large number of trials will become closer to the expected value as the number of trials increases.

8.6 Solving Two-Dimensional, Linear Black-Scholes Equations

8.6.1 Parameter Values

The parameter values used in the Black-Scholes equation (26) are :

Parameter	Values
S_1, S_2	0.0(0.1)20.0
E	1.0
r	0.05
σ_1, σ_2	0.25
ρ_{12}	0.1
τ	1.0

Table 8.1 Parameter Values Used in the Two-Dimensional, Linear Black-Scholes Equation

8.6.2 Aim

The aim of this investigation is to collect speed and accuracy data for sequential and parallel algorithms for solving the two-dimensional, linear Black-Scholes equation (26) and use it to compare their relative performances.

8.6.3 Preliminary Notes

The linear equation is solved to demonstrate the potential of the LTFD approach and also because some nonlinear, two-dimensional Black-Scholes equations become linear under the financial assumptions described in 2.3.2 and 2.3.3.

8.6.4 Performance Data

The graphs and tables below provide a summary of the data collected. Once again, the graphs show visually, the relative performances of the numerical inversion algorithms. As before, only the graphs showing the minimum wall times are included. Detailed results are given in Appendix E.

8.6.4.1 Sequential Programs Data

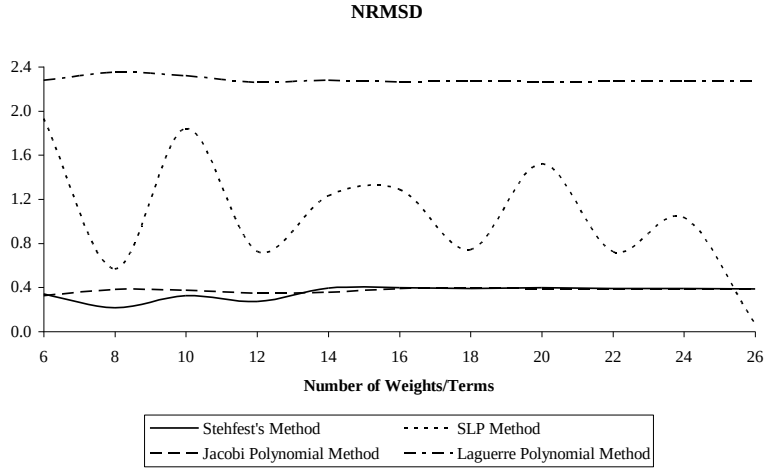

Figure 8.5 Normalised Root Mean Square Deviation, Sequential Programs[94]

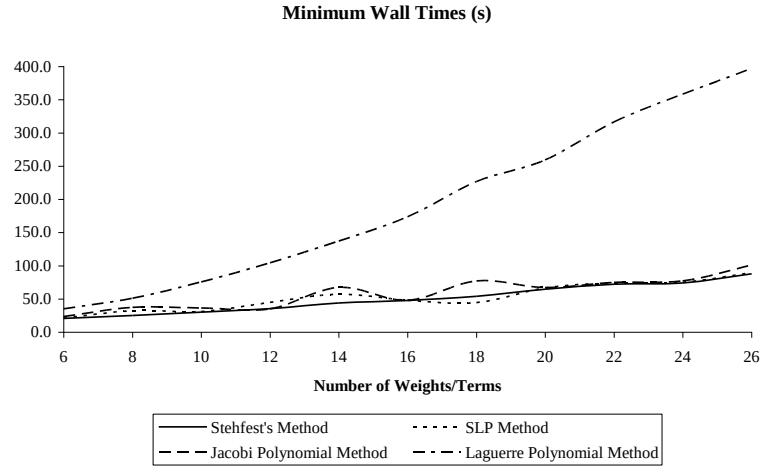

Figure 8.6 Minimum Wall Times, Sequential Programs

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Value
NRMSD :	SLP	26	0.06469810112
Minimum Wall Time (s) :	Stehfest	6	20.80843711000

Table 8.2 Optimal Sequential Programs Data

[94] The NRMSD values for all of the numerical inversion algorithms follow oscillating trends.

8.6.4.2 Parallel Programs Data

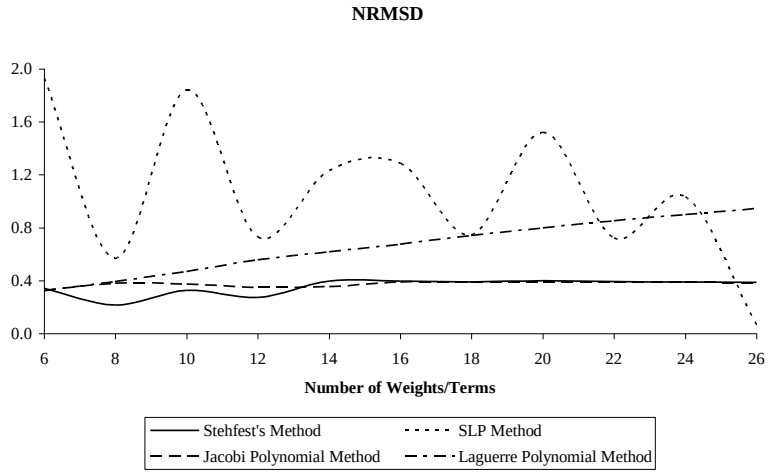

Figure 8.7 Normalised Root Mean Square Deviation, Parallel Programs[95]

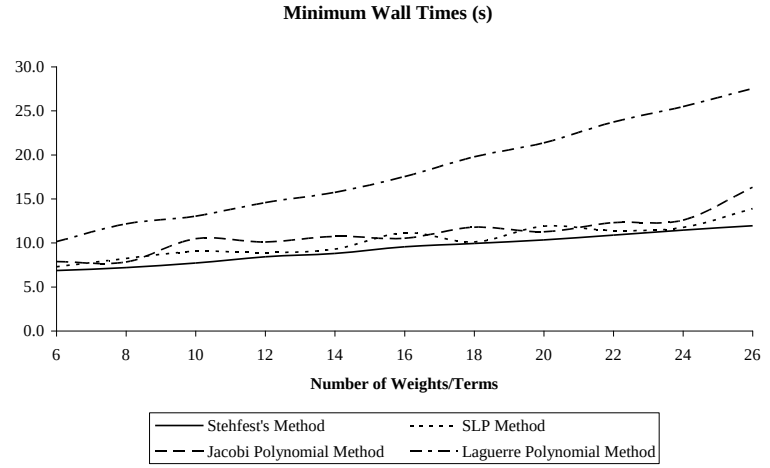

Figure 8.8 Minimum Wall Times, Parallel Programs

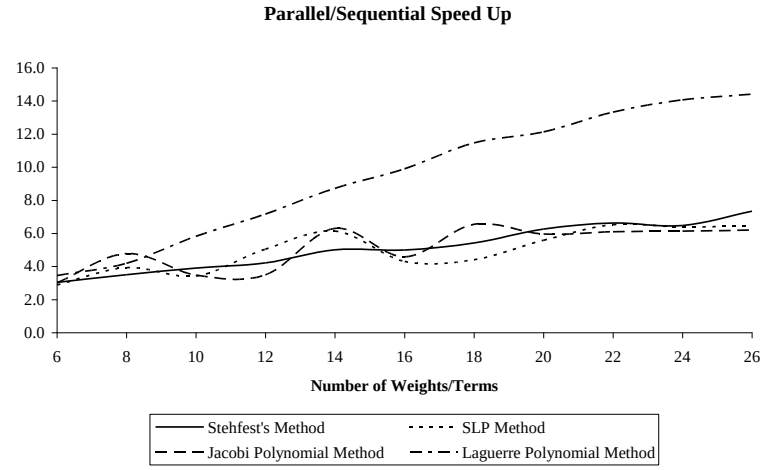

Figure 8.9 Parallel/Sequential Speed Up

[95] The NRMSD values for the Laguerre polynomial method follow an upward trend. The NRMSD values for the other numerical inversion algorithms follow oscillating trends.

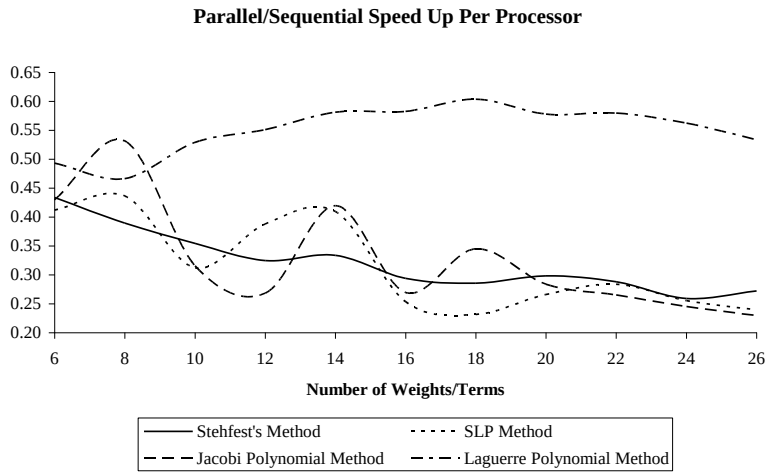

Figure 8.10 Parallel/Sequential Speed Up Per Processor

	Optimal Inversion Algorithm	Optimal Weights/Terms (Processors)	Optimal Value
NRMSD :	SLP	26 (27)	0.06469810112
Minimum Wall Time (s) :	Stehfest	6 (7)	6.85409307500
Parallel/Sequential Speed Up :	Laguerre	26 (27)	14.40714439882
Parallel/Sequential Speed Up/Processor :	Laguerre	18 (19)	0.60357633598

Table 8.3 Optimal Parallel Programs Data

8.6.5 Conclusions

Tables 8.2 and 8.3 above give the optimal number of weights/terms and processors to use in/with the LTFD algorithms when they are used to solve the two-dimensional, linear Black-Scholes equation (26).

It can be seen from the data collected that :

- the parallel programs are as accurate as the corresponding sequential programs but faster
- overall, the solutions obtained are not as accurate as those obtained in the one-dimensional case, particularly when the inverse Laplace transforms are found using the Laguerre polynomial method. However, the solutions are still sufficiently accurate for their purpose. See 11.1. More work is needed to determine the reason for the reduced accuracy in the two-dimensional case and establish what can be done to improve it. This is left for future work. See 11.2.
- as the number of weights/terms used in the numerical inversion algorithm increases the measures of speed and accuracy used oscillate, sometimes with an upward trend and sometimes with a downward trend. The likely reasons for this behaviour is explained in Chapter 10.

8.7 Chapter Summary

The two-dimensional, Laplace transform-finite difference algorithms can accurately and quickly solve two-dimensional, linear Black-Scholes equations. Chapter 9 will determine whether these algorithms will be equally successful when they are used to solve nonlinear Black-Scholes equations of this type.

8.8 Contribution to Knowledge

This chapter has :

- explained how Monte Carlo methods can be used for calculating accurate reference solutions of two-dimensional, linear Black-Scholes equations
- developed and evaluated sequential and parallel, Laplace transform-finite difference algorithms for solving two-dimensional, linear Black-Scholes equations and shown that these algorithms produce fast and sufficiently accurate solutions[96]
- evaluated Stehfest's method, the shifted Legendre polynomial method, the Jacobi polynomial method and the Laguerre polynomial method when these methods are used to invert the Laplace transforms arising in the LTFD algorithms
- determined the optimal number of weights/terms and processors to use with each of the numerical inversion algorithms when they are used in the LTFD algorithms
- established the advantages of using the LTFD algorithm for solving two-dimensional linear Black-Scholes equations.

[96] See 11.1

Chapter 9

Solving Two-Dimensional, Nonlinear Black-Scholes Equations

9.0 Introduction

Two-dimensional, nonlinear Black-Scholes equations are more difficult to solve than linear equations of this type. However, this chapter will show that even problems of this complexity can be solved using the Laplace transform-finite difference approach.

9.1 Practical Difficulties When Solving the Nonlinear Form

The problems encountered when solving two-dimensional, nonlinear Black-Scholes equations are the same as those encountered in the one-dimensional case *i.e.* dealing with the nonlinear modified volatility terms and calculating accurate reference solutions. Both problems can be dealt with in the same way as before *i.e.* by incorporating a linearisation technique into the basic algorithms described in Chapter 8 and by using a Monte Carlo algorithm. In Chapter 7 it was shown that the simplest linearisation technique is direct iteration. This technique is also used in the two-dimensional case. Please see 7.2.1.1 and 7.2.1.4 for a description of this method and its termination. Please see 7.2.1.6 for details of the relationship between the accuracy of the option values and the accuracy of the volatility forecasts. Using direct iteration in the two-dimensional case :

- the LTFD algorithm for solving the nonlinear equation (33) becomes :

initialise $\tilde{\sigma}_1$ and $\tilde{\sigma}_2$

repeat

use the 2-d linear LTFD algorithm to calculate the V values

use the V values to recalculate $\tilde{\sigma}_1$ and $\tilde{\sigma}_2$

until $\left| \dfrac{\max\left(\text{abs}\left(V^{(r-1)} - V^{(r)}\right)\right)}{\max\left(\text{abs}\left(V^{(r-1)} + V^{(r)}\right)\right)} \right| < \varepsilon$

Figure 9.1 The Laplace Transform-Finite Difference Algorithm for Solving Two-Dimensional, Nonlinear Black-Scholes Equations

Here, $V^{(r)}$ denotes the value of V at the rth iteration and ε is a small positive number

- the proposed Monte Carlo algorithm for calculating the reference solutions is :

decompose $\sum = LL^T$

initialise $\tilde{\sigma}_1$ and $\tilde{\sigma}_2$

repeat

loop i = 1 to n

loop j=1 to d

generate $y_j = Z_j$

end loop

$\underline{\phi} = L\underline{y}$

loop k=1 to d

$$S_{k,\tau} = S_k \exp\left(\left[r - \frac{1}{2}\sigma_k^2\right]\tau + \phi_k\right)$$

end loop

$$V_i = e^{-r\tau} \max\left(\frac{1}{d}\sum_{j=1}^{d} S_{j,\tau} - E, 0\right)$$

end loop

$$\hat{V} = \frac{\left(V_1 + V_2 + V_3 + \ldots + V_n\right)}{n} = \frac{1}{n}\sum_{i=1}^{n} V_i$$

recalculate $\tilde{\sigma}_1$ and $\tilde{\sigma}_2$

until $$\left|\frac{\max\left(\text{abs}\left(\hat{V}^{(r-1)} - \hat{V}^{(r)}\right)\right)}{\max\left(\text{abs}\left(\hat{V}^{(r-1)} + \hat{V}^{(r)}\right)\right)}\right| < \varepsilon$$

Figure 9.2 A Monte Carlo Algorithm for Solving Two-Dimensional, Nonlinear Black-Scholes Equations[97]

Here $\tilde{\sigma}_1$ and $\tilde{\sigma}_2$ are the modified volatilities, $\hat{V}^{(r)}$ denotes the value of $\hat{V}$ at the rth iteration and ε is a small positive number.

[97] This algorithm was invented by the author *i.e.* is a contribution to knowledge. It is based upon the algorithm for solving two-dimensional, linear Black-Scholes equations developed by Wilmot *et al.* (1999) and Glasserman (2003).

9.2 Solving Two-Dimensional, Nonlinear Black-Scholes Equations

9.2.1 Parameter Values

The parameter values used in the Black-Scholes equation (33) are :

Parameter	Values
S_1, S_2	0.0(0.1)5.0[98]
E	1.0
r	0.05
ρ_{12}	0.1
τ	1.0

Table 9.1 Parameter Values Used in the Two-Dimensional, Nonlinear Black-Scholes Equation

The parameter values used in the modified volatility models are the same as those given in Table 7.1. For a European call option the modified volatilities proposed by Leland and Boyle and Vorst become constants. The data for these functions is therefore collected using the LTFD program for the two-dimensional, linear Black-Scholes equation (26) and the parameter values given in Table 8.1.

9.2.2 Aim

The aim of this investigation is to collect speed and accuracy data for sequential and parallel algorithms for solving two-dimensional, nonlinear Black-Scholes equations and use it to compare their relative performances.

9.2.3 Preliminary Notes

- the gamma term in the Barles and Soner and Risk Adjusted Pricing Methodology modified volatility models is calculated using the finite difference approximation given in 7.4.3
- the reference solutions used to determine the NRMSD values for the nonlinear Black-Scholes equations are calculated using the Monte Carlo algorithm given in Figure 9.2.

9.2.4 Performance Data

The graphs and tables below provide a summary of the data collected. Once again, the graphs show visually, the relative performances of the numerical inversion algorithms. As before, only the graphs showing the minimum wall times are included. Detailed results are given in Appendix F.

[98] As in the one-dimensional case, the *S*-range is reduced for the nonlinear modified volatility models because of the slow speed of the Monte Carlo algorithm for calculating the reference solutions.

9.2.4.1 Modified Volatility Model : Simulated Modified Volatility[99]

9.2.4.1.1 Sequential Programs Data

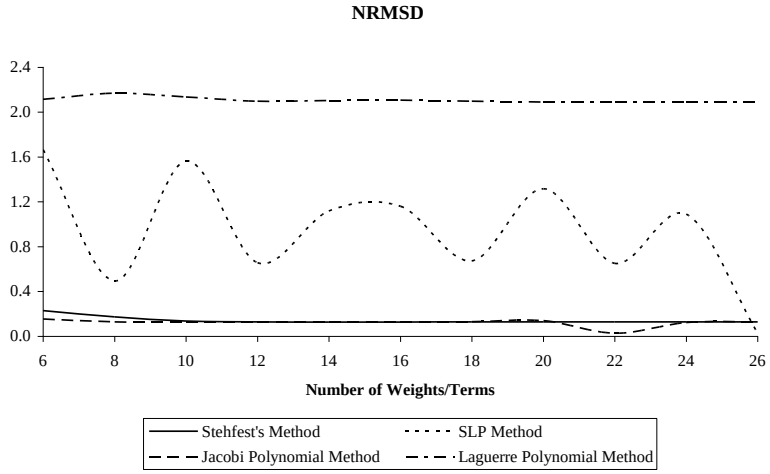

Figure 9.3 Normalised Root Mean Square Deviation, Sequential Programs (Simulated Modified Volatility)[100]

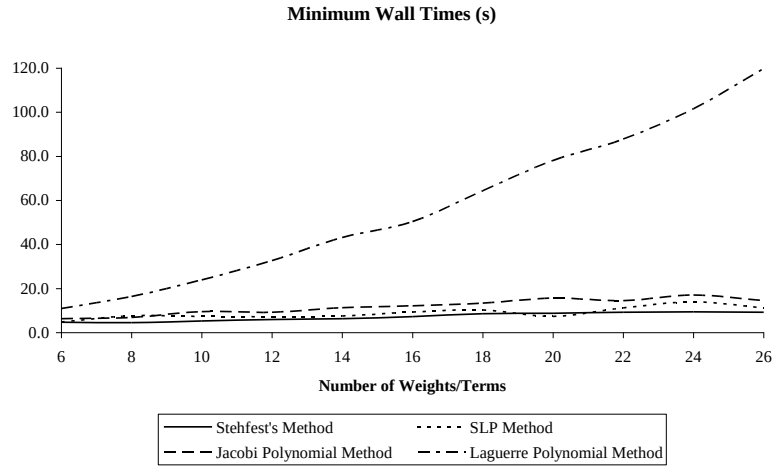

Figure 9.4 Minimum Wall Times, Sequential Programs (Simulated Modified Volatility)

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Value
NRMSD :	Jacobi	22	0.02772967769
Minimum Wall Time (s) :	Stehfest	8	4.55571794500

Table 9.2 Optimal Sequential Programs Data (Simulated Modified Volatility)

[99] See Lai *et al.* (2005).

[100] The NRMSD values for all methods follow oscillating trends.

9.2.4.1.2 Parallel Programs Data

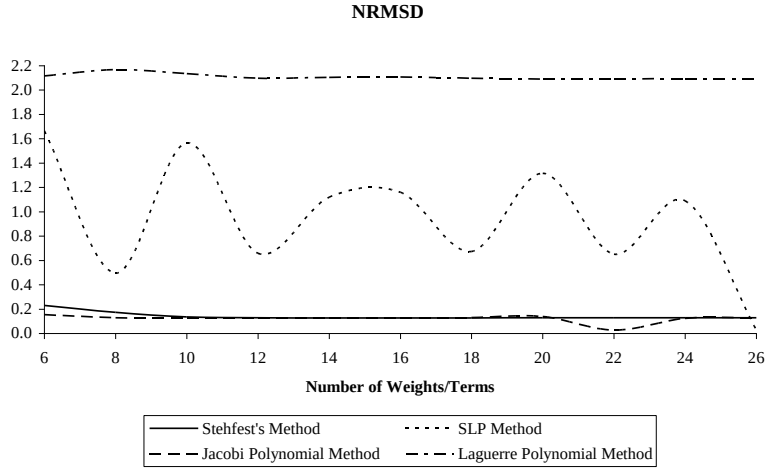

Figure 9.5 Normalised Root Mean Square Deviation, Parallel Programs (Simulated Modified Volatility)[101]

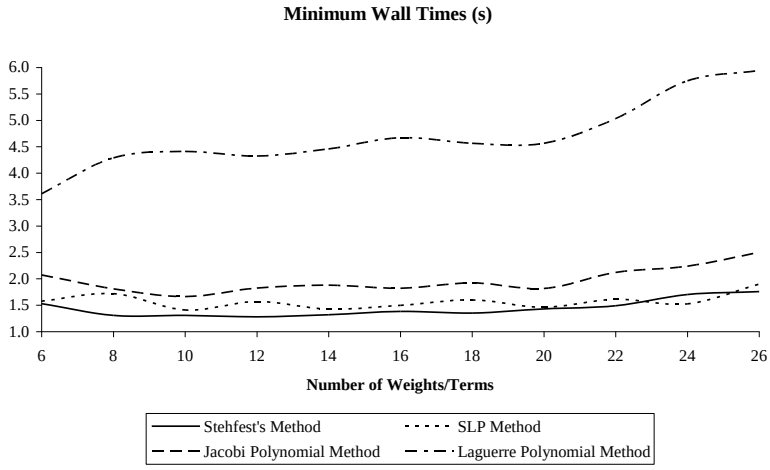

Figure 9.6 Minimum Wall Times, Parallel Programs (Simulated Modified Volatility)

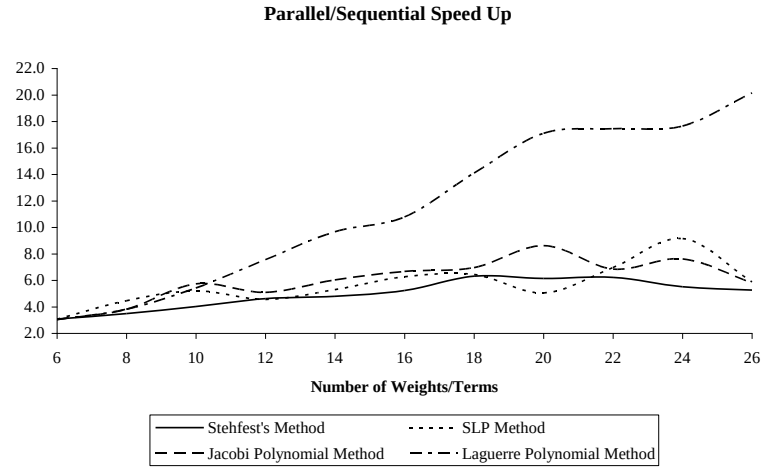

Figure 9.7 Parallel/Sequential Speed Up (Simulated Modified Volatility)

[101] The NRMSD values for all methods follow oscillating trends.

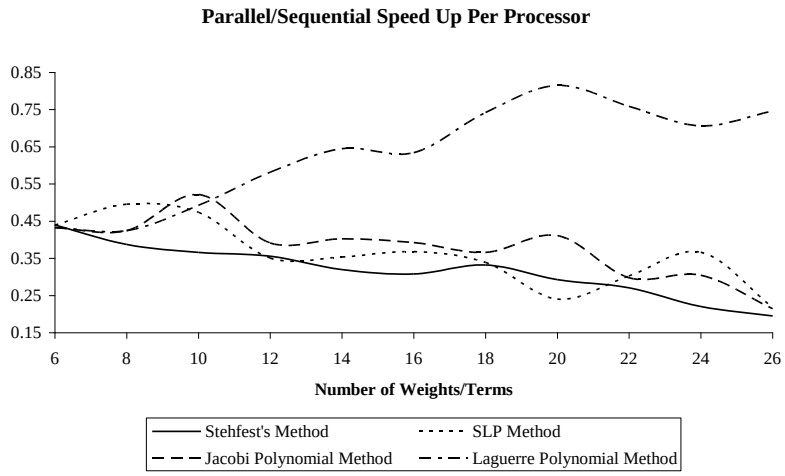

Figure 9.8 Parallel/Sequential Speed Up Per Processor (Simulated Modified Volatility)

	Optimal Inversion Algorithm	Optimal Weights/Terms (Processors)	Optimal Value
NRMSD :	Jacobi	22 (23)	0.02772967769
Minimum Wall Time (s) :	Stehfest	12 (13)	1.28097391100
Parallel/Sequential Speed Up :	Laguerre	26 (27)	20.15634733328
Parallel/Sequential Speed Up/Processor :	Laguerre	20 (21)	0.81481848339

Table 9.3 Optimal Parallel Programs Data (Simulated Modified Volatility)

9.2.4.2 Modified Volatility Model : Leland

9.2.4.2.1 Sequential Programs Data

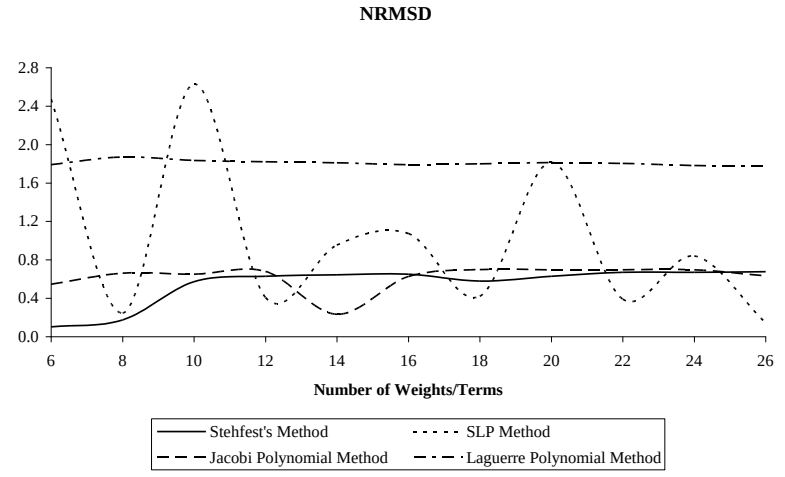

Figure 9.9 Normalised Root Mean Square Deviation, Sequential Programs (Leland)[102]

[102] The NRMSD values for all methods follow oscillating trends.

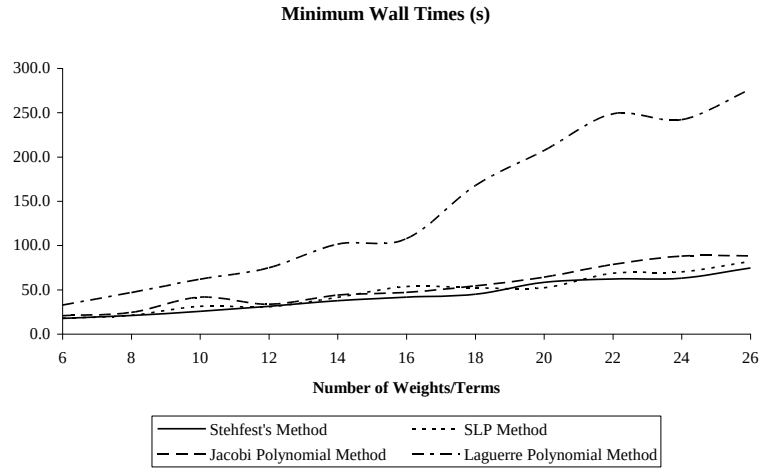

Figure 9.10 Minimum Wall Times, Sequential Programs (Leland)

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Value
NRMSD :	Stehfest	6	0.10201155790
Minimum Wall Time (s) :	Stehfest	6	17.76280676000

Table 9.4 Optimal Sequential Programs Data (Leland)

9.2.4.2.2 Parallel Programs Data

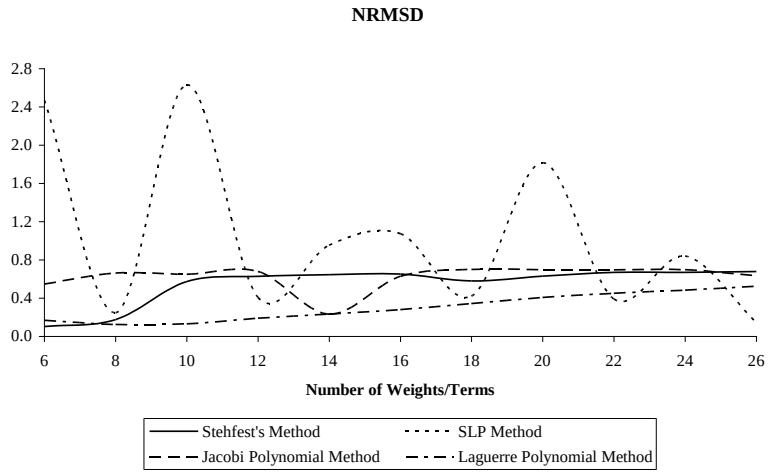

Figure 9.11 Normalised Root Mean Square Deviation, Parallel Programs (Leland)

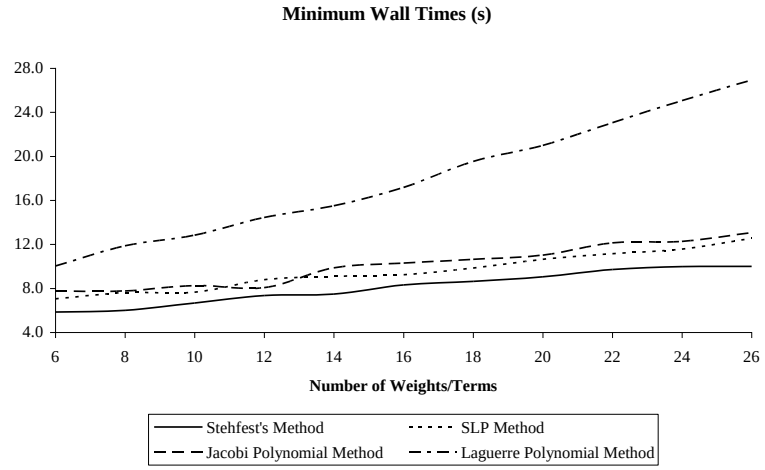

Figure 9.12 Minimum Wall Times, Parallel Programs (Leland)

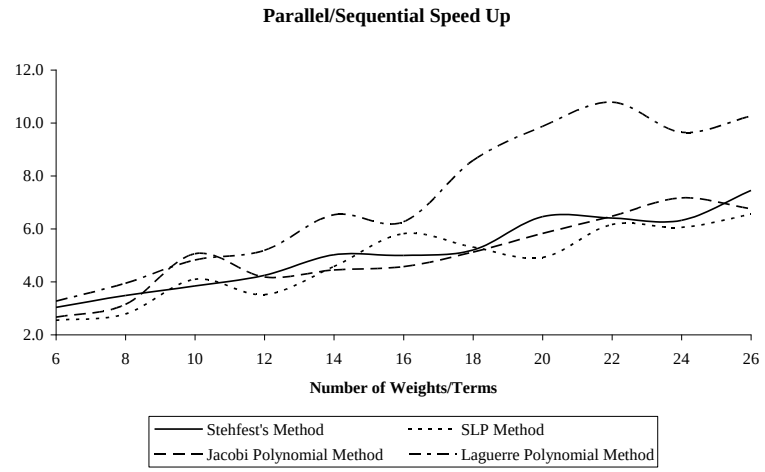

Figure 9.13 Parallel/Sequential Speed Up (Leland)

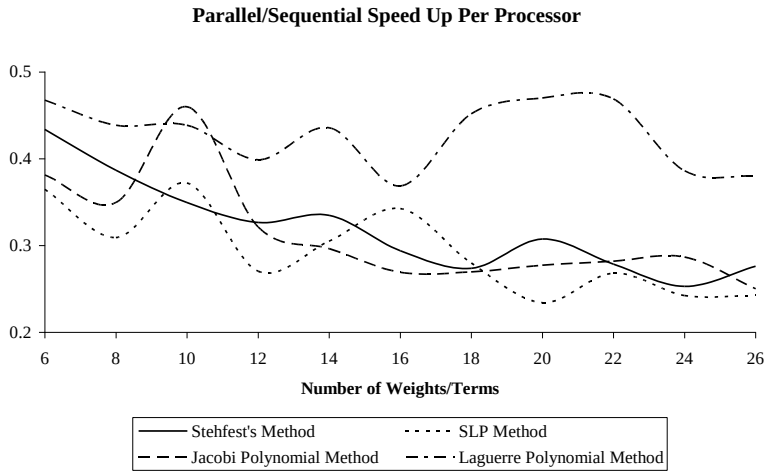

Figure 9.14 Parallel/Sequential Speed Up Per Processor (Leland)

	Optimal Inversion Algorithm	Optimal Weights/Terms (Processors)	Optimal Value
NRMSD :	Stehfest	6 (7)	0.10201155790
Minimum Wall Time (s) :	Stehfest	6 (7)	5.85089260600
Parallel/Sequential Speed Up :	Laguerre	22 (23)	10.78343361760
Parallel/Sequential Speed Up/Processor :	Laguerre	20 (21)	0.47000707041

Table 9.5 Optimal Parallel Programs Data (Leland)

9.2.4.3 Modified Volatility Model : Boyle and Vorst

9.2.4.3.1 Sequential Programs Data

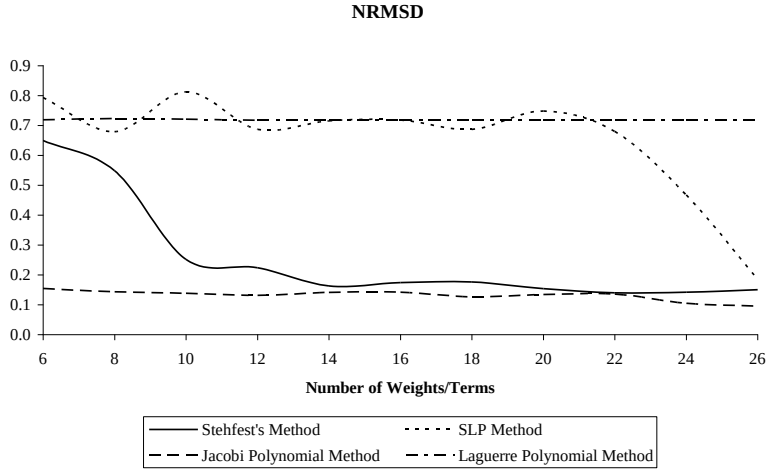

Figure 9.15 Normalised Root Mean Square Deviation, Sequential Programs (Boyle and Vorst)[103]

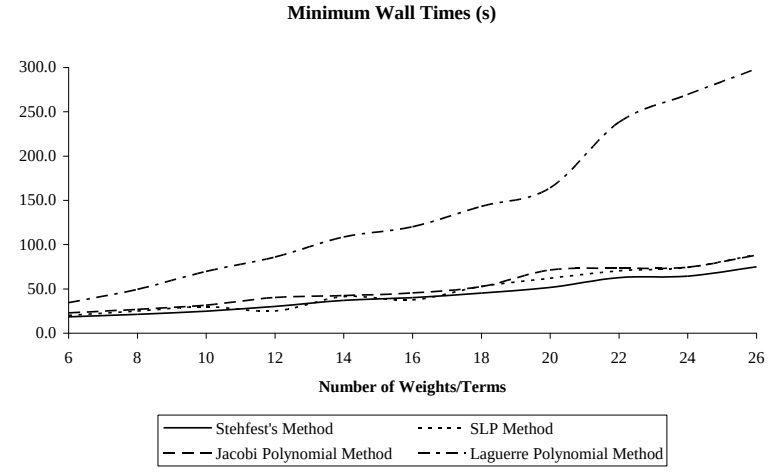

Figure 9.16 Minimum Wall Times, Sequential Programs (Boyle and Vorst)

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Value
NRMSD :	Jacobi	26	0.09513891832
Minimum Wall Time (s) :	Stehfest	6	18.27469492000

Table 9.6 Optimal Sequential Programs Data (Boyle and Vorst)

[103] The NRMSD values for the Laguerre polynomial method follow an almost constant trend.

9.2.4.3.2 Parallel Programs Data

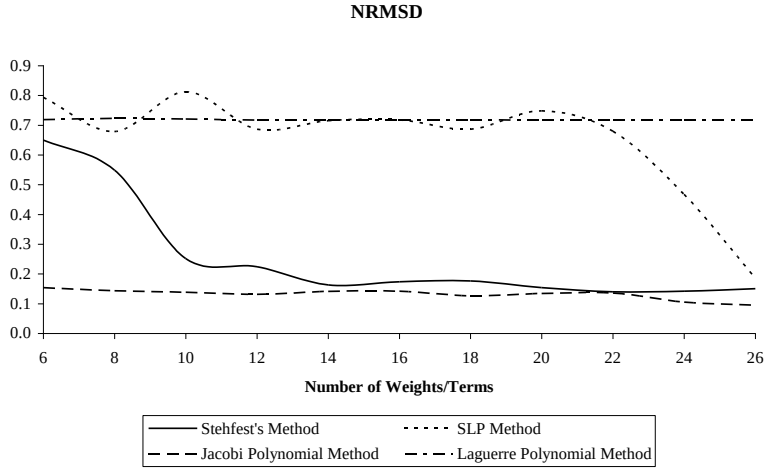

Figure 9.17 Normalised Root Mean Square Deviation, Parallel Programs (Boyle and Vorst)

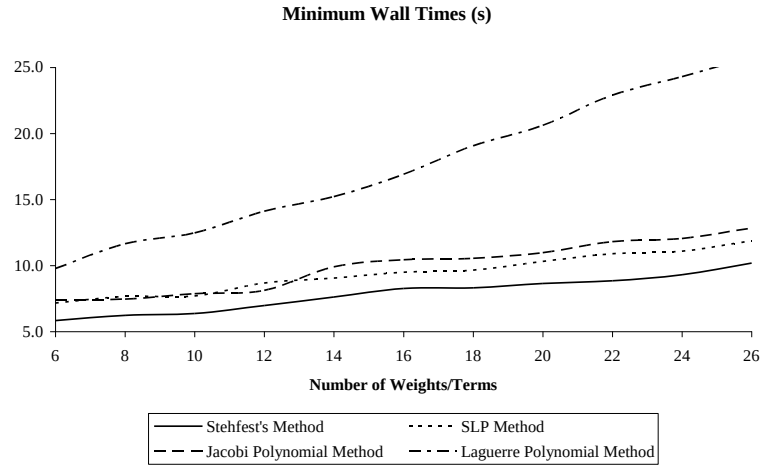

Figure 9.18 Minimum Wall Times, Parallel Programs (Boyle and Vorst)

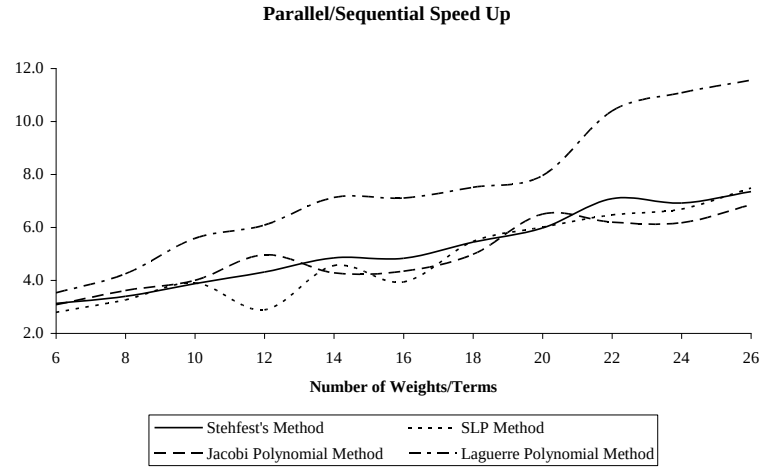

Figure 9.19 Parallel/Sequential Speed Up (Boyle and Vorst)

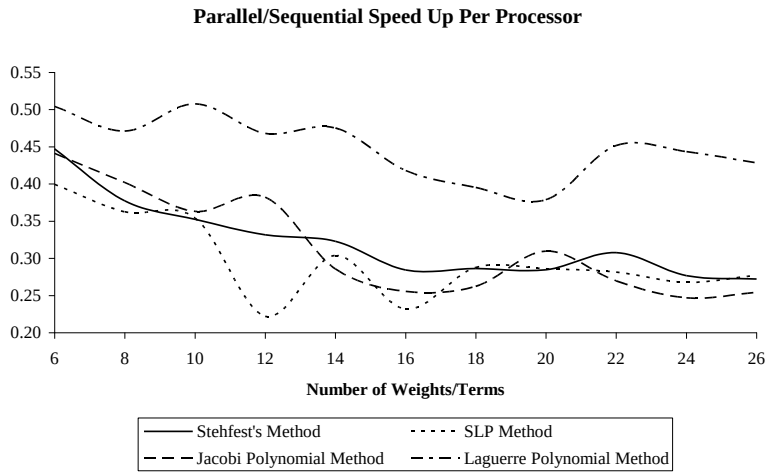

Figure 9.20 Parallel/Sequential Speed Up Per Processor (Boyle and Vorst)

	Optimal Inversion Algorithm	Optimal Weights/Terms (Processors)	Optimal Value
NRMSD :	Jacobi	26 (27)	0.09513891832
Minimum Wall Time (s) :	Stehfest	6 (7)	5.83855523400
Parallel/Sequential Speed Up :	Laguerre	26 (27)	11.56211680346
Parallel/Sequential Speed Up/Processor :	Laguerre	10 (11)	0.50765187507

Table 9.7 Optimal Parallel Programs Data (Boyle and Vorst)

9.2.4.4 Modified Volatility Model : Barles and Soner

9.2.4.4.1 Sequential Programs Data

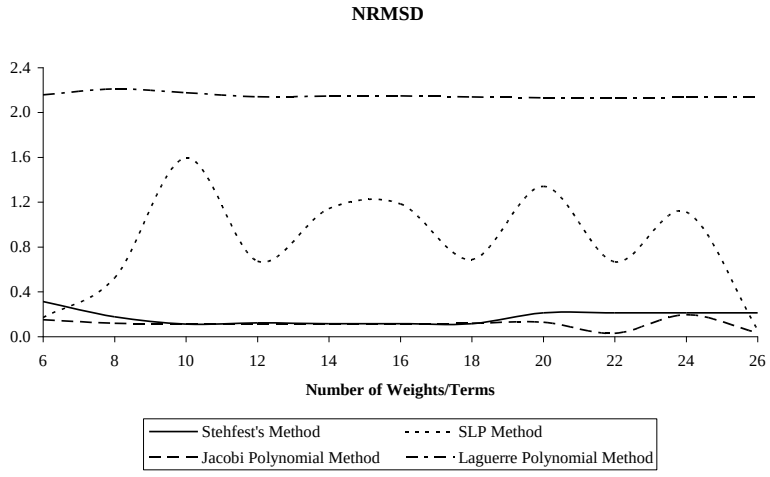

Figure 9.21 Normalised Root Mean Square Deviation, Sequential Programs (Barles and Soner)[104]

[104] The NRMSD values for the Laguerre polynomial method follow a slight oscillating trend.

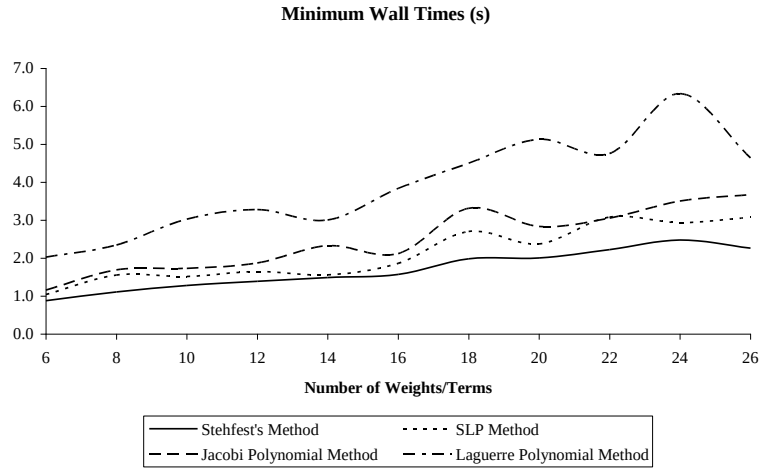

Figure 9.22 Minimum Wall Times, Sequential Programs (Barles and Soner)

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Value
NRMSD :	Jacobi	22	0.02922910136
Minimum Wall Time (s) :	Stehfest	6	0.88142395020

Table 9.8 Optimal Sequential Programs Data (Barles and Soner)

9.2.4.4.2 Parallel Programs Data

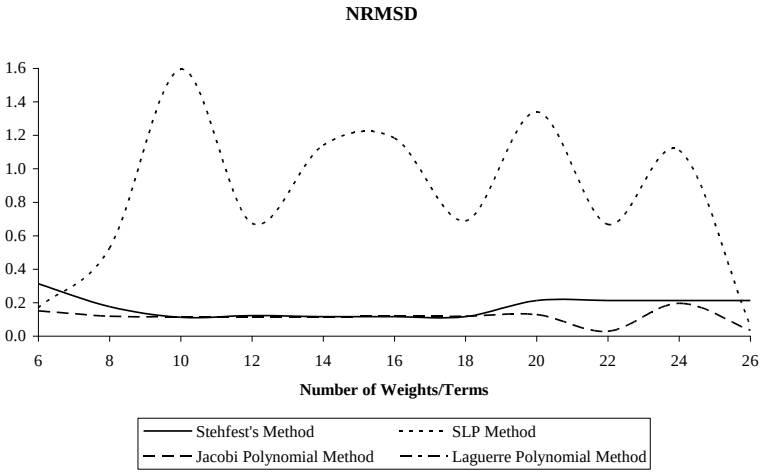

Figure 9.23 Normalised Root Mean Square Deviation, Parallel Programs (Barles and Soner)

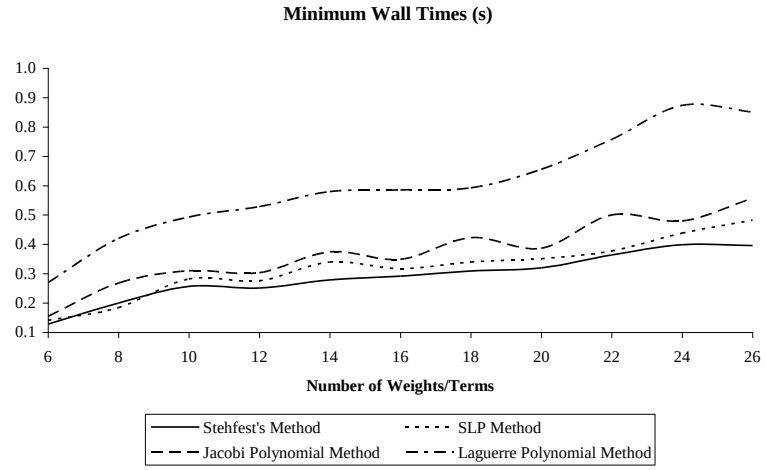

Figure 9.24 Minimum Wall Times, Parallel Programs (Barles and Soner)

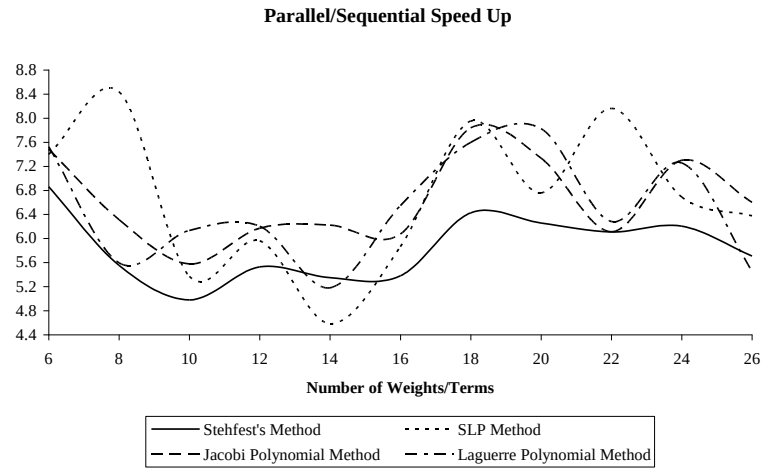

Figure 9.25 Parallel/Sequential Speed Up (Barles and Soner)

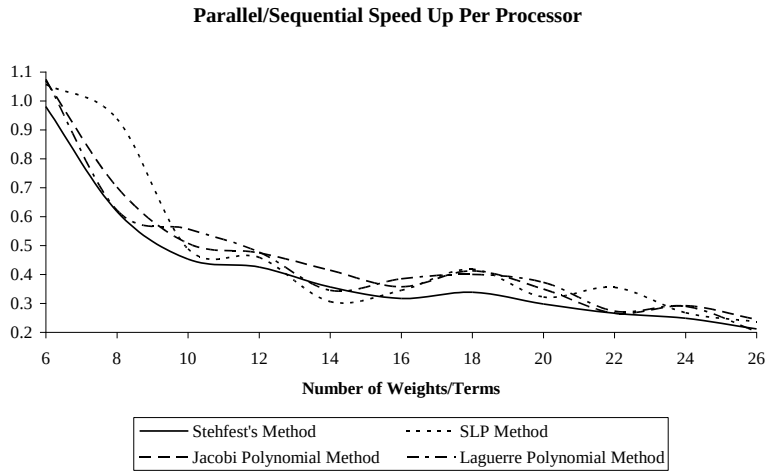

Figure 9.26 Parallel/Sequential Speed Up Per Processor (Barles and Soner)

	Optimal Inversion Algorithm	Optimal Weights/Terms (Processors)	Optimal Value
NRMSD :	Jacobi	22 (23)	0.02922910136
Minimum Wall Time (s) :	Stehfest	6 (7)	0.12844991680
Parallel/Sequential Speed Up :	SLP	8 (9)	8.43622492843
Parallel/Sequential Speed Up/Processor :	Laguerre	6 (7)	1.07483109645

Table 9.9 Optimal Parallel Programs Data (Barles and Soner)

9.2.4.5 Modified Volatility Model : Risk Adjusted Pricing Methodology

9.2.4.5.1 Sequential Programs Data

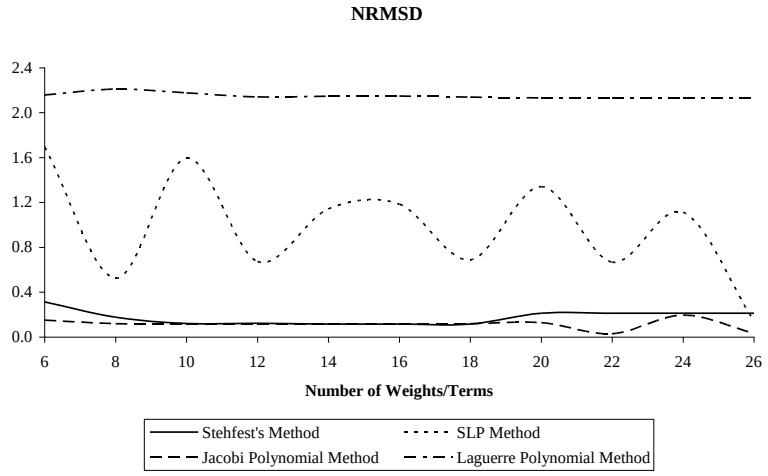

Figure 9.27 Normalised Root Mean Square Deviation, Sequential Programs (RAPM)[105]

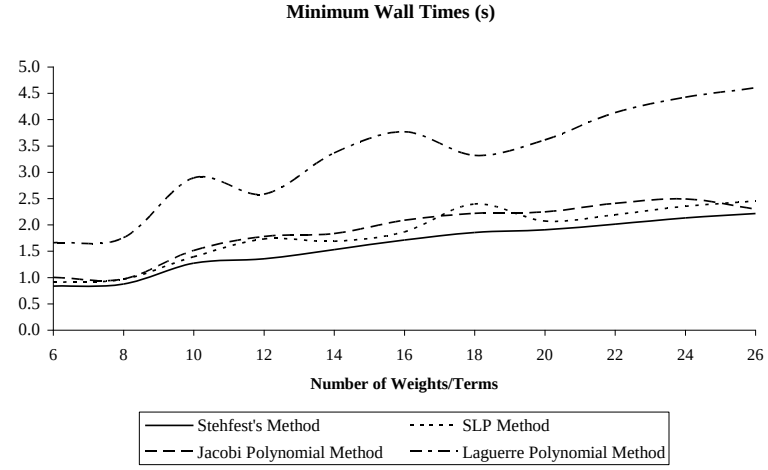

Figure 9.28 Minimum Wall Times, Sequential Programs (RAPM)

	Optimal Inversion Algorithm	Optimal Weights/Terms	Optimal Value
NRMSD :	Jacobi	22	0.02922910136
Minimum Wall Time (s) :	Stehfest	6	0.83622002600

Table 9.10 Optimal Sequential Programs Data (RAPM)

[105] The NRMSD values for all methods follow oscillating trends.

9.2.4.5.2 Parallel Programs Data

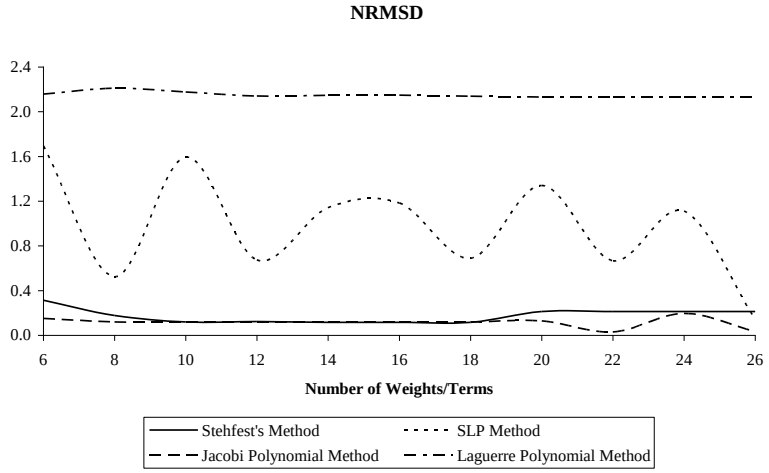

Figure 9.29 Normalised Root Mean Square Deviation, Parallel Programs (RAPM)[106]

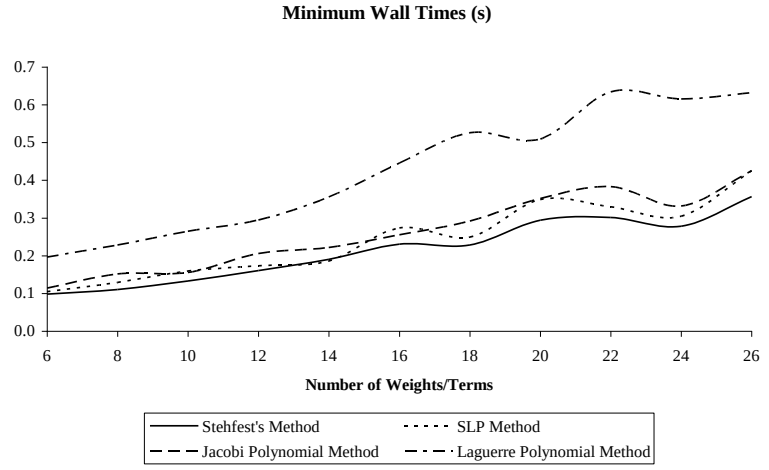

Figure 9.30 Minimum Wall Times, Parallel Programs (RAPM)

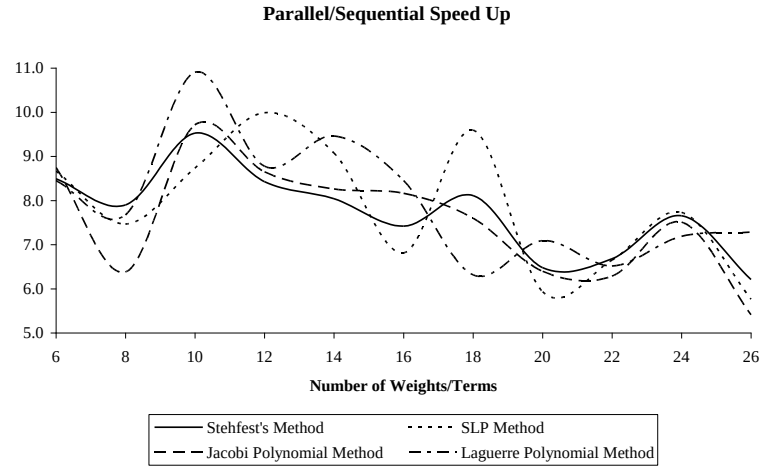

Figure 9.31 Parallel/Sequential Speed Up (RAPM)

[106] The NRMSD values for all methods follow oscillating trends.

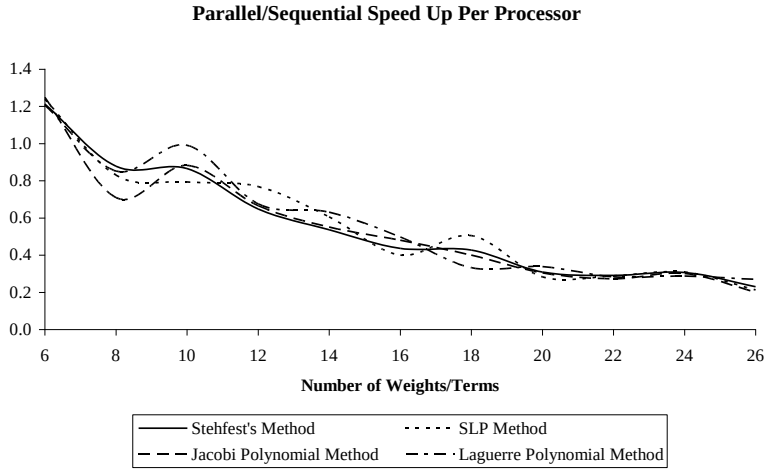

Figure 9.32 Parallel/Sequential Speed Up Per Processor (RAPM)

	Optimal Inversion Algorithm	Optimal Weights/Terms (Processors)	Optimal Value
NRMSD :	Jacobi	22 (23)	0.02922910136
Minimum Wall Time (s) :	Stehfest	6 (7)	0.09848403931
Parallel/Sequential Speed Up :	Laguerre	10 (11)	10.90384985034
Parallel/Sequential Speed Up/Processor :	Jacobi	6 (7)	1.24914529301

Table 9.11 Optimal Parallel Programs Data (RAPM)

9.2.5 Conclusions

Tables 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9, 9.10 and 9.11 above give the optimal number of weights/terms and processors to use in/with sequential and parallel programs for solving two-dimensional, nonlinear Black-Scholes equations containing the modified volatility models proposed by Lai *et al.*, Leland, Boyle and Vorst, Barles and Soner and Kratka (*i.e.* the Risk Adjusted Pricing Methodology model).

It can be seen from the graphs and tables in this chapter that :

- as in the two-dimensional, linear case the solutions obtained are not as accurate as those obtained in the one-dimensional case, particularly when the inverse Laplace transforms are found using the Laguerre polynomial method. The likely reasons for this are explained in 8.6.5
- the parallel programs are as accurate as the corresponding sequential programs but faster

- as seen in previous investigations, the Laguerre polynomial method only features in the parallel/sequential speed up categories because it is significantly slower than the other numerical inversion algorithms tested.

9.3 Chapter Summary

This chapter has shown that the Laplace transform-finite difference algorithm can be used to solve two-dimensional, nonlinear Black-Scholes equations. The investigative chapters of this dissertation have identified consistent patterns of behaviour within the numerical solutions. Chapter 10 identifies these patterns and explains how they are related to the number of weights/terms used in the numerical inversion algorithms and the number of processors used with the parallel Laplace transform based algorithms.

9.4 Contribution to Knowledge

This chapter has :

- explained how Monte Carlo methods can be used for calculating accurate reference solutions of two-dimensional, nonlinear Black-Scholes equations
- developed and evaluated a parallel, Laplace transform-finite difference algorithm for solving two-dimensional, nonlinear Black-Scholes equations and shown that this algorithm produces fast and sufficiently accurate solutions[107]
- used the LTFD algorithm for solving two-dimensional nonlinear Black-Scholes equations containing the modified volatility models proposed by Lai *et al*., Leland, Boyle and Vorst, Barles and Soner and Kratka (*i.e.* the Risk Adjusted Pricing Methodology model)
- evaluated Stehfest's method, the shifted Legendre polynomial method, the Jacobi polynomial method and the Laguerre polynomial method when these methods are used to invert the Laplace transforms arising in the LTFD algorithm
- determined the optimal number of weights/terms and processors to use with each of the numerical inversion algorithms when they are used in the LTFD algorithm
- established the advantages of using the LTFD algorithm for solving two-dimensional nonlinear Black-Scholes equations.

[107] See 11.1

Chapter 10

Program Performance

10.0 Introduction

In the investigative chapters of this dissertation a significant amount of performance data was collected for the Laplace transform based algorithms for solving one-dimensional and two-dimensional, linear and nonlinear Black-Scholes equations. It was seen from this data that the execution speeds and accuracies of the algorithms were related to the number of weights/ terms used in the numerical inversion algorithms and the number of processors used on the cluster. This chapter attempts to explain this behaviour.

10.1 The Effects of the Number of Weights/Terms Used

10.1.1 On Execution Speeds and Parallel/Sequential Speed Ups

Increasing the number of weights/terms used in a numerical inversion algorithm increases the number of calculations that must be performed. This in turn, increases the execution time of the procedure. The size of the increase depends upon the computational intensity of the algorithm. The computational intensity of a numerical inversion algorithm depends upon :

- the nature of the calculations that must be performed. All four algorithms require the calculation of recursive coefficients/weights. However, the Laguerre polynomial method requires significantly more calculations of this type than the Jacobi polynomial method. This method in turn, requires significantly more recursive calculations than the SLP method and Stehfest's method. Although the recursive coefficients/weights are calculated (and broadcast) at the beginning of each program, recursive calculations are notoriously slow compared with iterative calculations, even on modern computer systems. This is due to the need to maintain the stack *i.e.* the need to push intermediate values onto the stack and then pop these values from the stack as the recursion unwinds[108], (Lantzman 2007)
- the frequency with which those calculations must be performed.

[108] Lantzman (2007) compares the speeds (in clock ticks) of iterative and recursive implementations of a number of standard algorithms. He shows that the recursive implementations are between 3 and 30 times slower than their iterative counterparts.

Consider the one-dimensional case. In the Laguerre polynomial method the coefficients a_k must be calculated for each value of *S* and the polynomials L_k^{α} must be calculated for each value of τ. In the Jacobi polynomial method the coefficients c_n must be calculated for each value of *S* and the polynomials $P_n^{(0,\beta)}\left(2e^{-\delta\tau}-1\right)$ must be calculated for each value of τ. In the SLP method the coefficients C_k must be calculated for each value of *S* and the polynomials $P_k\left(z\right)$ must be calculated for each value of τ. However, in Stehfest's method no additional coefficients, polynomials or weights need to be calculated for each value of *S* and τ. In the two-dimensional case the frequencies are exactly the same. Here, the corresponding calculations must be performed for each pair of S_1 and S_2 values and for each value of τ.

Hence, in ascending order of computational intensity, the four numerical inversion algorithms can be ranked as Stehfest's method, the SLP method, the Jacobi polynomial method and the Laguerre polynomial method. This order corresponds with the minimum program wall time data shown on the graphs within this dissertation.

The relationship between the number of weights/terms used and the parallel/sequential speed up obtained is less clear. For example, Figure 5.4 and Figure 5.16 show that when the numerical inversion algorithms are used to invert the Laplace transform of the analytical solution of the one-dimensional, linear Black-Scholes equation (10) and the Laplace transforms arising in the finite difference solution of the ODE BVP form of this equation, the parallel/sequential speed up values oscillate as the number of weights/terms used increases. However, Figure 8.9 shows that in the two-dimensional linear case, the parallel/sequential speed up values increase monotonically as the number of weights/terms used increases.

10.1.2 On Accuracies

Increasing the number of weights/terms should increase the accuracy of the solution. However, as stated above, increasing the number of weights/terms increases the number of calculations that must be performed *i.e.* increases the rounding errors in the solution. Hence, for each numerical inversion algorithm there is an optimal number of weights/terms *i.e.* a number beyond which the inaccuracy lost due to rounding errors (and the ill-posed nature of numerical Laplace transform inversion) exceeds the accuracy gained by increasing the number of weights/terms. This behaviour can be seen in the Optimal Parallel Program Data tables included in this dissertation.

10.2 The Effects of the Number of Processors Used

10.2.1 On Execution Speeds and Parallel/Sequential Speed Ups

Before any performance data was collected it was reasonable to assume that the minimum program wall time in both the one-dimensional and the two-dimensional cases would be obtained when the number of processors used was the maximum number available. However, in reality this was not the case. The tables giving the optimal parallel programs data show that the minimum program wall time was usually obtained when the number of processors used was significantly lower than this maximum number. The likely reason for this is that beyond a certain number of processors, the communication overhead, that is, the time required to manage the parallel tasks, starts to increase the execution time. Barney (2010) states that the factors contributing to the communication overhead are :

- task start-up times
- task termination times
- the software overhead imposed by parallel compilers, libraries, operating systems, *etc.*
- inter-task communication times *i.e.*
    - the times taken to send and receive data
    - synchronisations between tasks such as those associated with blocking communications. These can result in tasks waiting instead of doing useful work
    - lack of network bandwidth. This can result in tasks waiting until the required communication resources become available.

Naturally, as the execution time of a parallel program increases, the parallel/sequential speed up obtained decreases.

10.2.2 On Accuracies

By looking at the graphs showing the normalised root mean square deviation values in both the one-dimensional and the two-dimensional cases it can be seen that beyond a certain number of processors, the accuracy of the numerical solutions usually decreases. More work is needed to establish the reason for this behaviour and determine what can be done to correct it. This is left for future work. See 11.2.

10.3 Chapter Summary

The explanations given in this chapter are correct in theory and explain the behaviour shown on/in the graphs and tables summarising the performance data in both the one-dimensional and the two dimensional cases. While the relative execution speeds of the numerical inversion algorithms was consistent in all investigations completed, the algorithms took turns at giving the most accurate results. This confirms the assertion made by Davies and Martin in their 1979 review paper that "*there is no single best numerical inversion algorithm*". The final chapter of this dissertation will give the overall contribution to knowledge made by this research programme and to explain how it can be extended further.

10.4 Contribution to Knowledge

This chapter has provided detailed explanations of how the performances of the sequential and parallel programs used to solve one-dimensional and two-dimensional Black-Scholes equations are effected by the number of weights/terms used in the numerical inversion algorithms by the number of processors used on the cluster.

Chapter 11

Conclusions and Future Work

11.0 Introduction

The final chapter of this dissertation gives the overall contribution to knowledge made by this research programme and explains how those interested can extend it further.

11.1 Overall Contribution to Knowledge

This research programme has achieved the aims given in Chapter 1 *i.e.* it has :

- developed and evaluated sequential finite difference algorithms and sequential and parallel Laplace transform based algorithms for solving one-dimensional and two-dimensional, linear and nonlinear diffusion equations. In particular, Black-Scholes equations of these types
- determined the optimal numerical inversion algorithms to use in the Laplace transform based algorithms for solving these equations and the optimal number of weights/terms and processors to use in each case
- provided the evidence to support my thesis *i.e.* shown that Laplace transform based algorithms produce fast and accurate solutions of the Black-Scholes equations mentioned above. In financial markets the solutions of Black-Scholes equations are used by traders as *guide prices* for the values of financial options. Even the least accurate solutions produced by the Laplace transform based algorithms *e.g.* those produced using the Laguerre polynomial method in the two-dimensional case, are adequate for this purpose.

11.2 Future Work

Those wishing to extend this research programme could :

- investigate the performance of the Laplace transform based algorithms when they are used to solve Black-Scholes equations in which the volatility is modelled by a stochastic process
- perform an error analysis of the numerical inversion algorithms. One way to do this is to use rounded interval arithmetic. This is an accepted and widely used error bounding method that has been in use since the early 1960's, (Moore *et al.* 2009). In this method

numerical values are stored as intervals and rules are available for adding, subtracting, multiplying and dividing intervals. Whenever an upper/lower end point of an interval does not have an exact binary representation (*i.e.* is not an integral power of 2), it is rounded up/down to the nearest representable value. A common rounding system is to add/subtract a small constant to/from the end point called the *unit-in-the-last-place*, (Abrams *et al.* 1998). Computer programs implementing the numerical inversion algorithms in this way could be used to :

  - obtain error bounds on the inverse Laplace transform values calculated
  - identify the types of problems for which numerical Laplace transform inversion is ill-posed
  - determine where the ill-posedness occurs in the inversion process
  - investigate how or if the ill-posedness can be controlled

- Validate the accuracy of the solutions obtained using the Laplace transform based algorithms using corresponding algorithms based on the use of Fourier and Fast Fourier transforms
- Investigate and evaluate other linearisation techniques. Possible methods are exact linearisation, direct linearisation, Newton's method, Picard iteration and explicit time integration
- Investigate methods for accelerating the convergence of linearisation techniques. Possible methods are Richardson extrapolation, the Aitken delta-squared process, Wynn's epsilon algorithm, the Levin u-transform and the Wilf-Zeilberger-Ekhad method
- Determine the reason for the reduced accuracy in the two-dimensional case and establish what can be done to improve it
- Establish why the accuracy of a parallel numerical solution usually decreases as the number of processors used increases and determine what can be done to correct this behaviour
- Investigate whether Automatic Differentiation can be used for obtaining highly accurate solutions of option pricing problems such as those described in this dissertation. Solutions of this kind may be required for calculating the ‘Greeks’.

11.3 Chapter Summary

This chapter has shown that this research programme has made significant contributions to knowledge and that there is scope for it to be extended further. Most importantly for me, it has shown that my thesis, stated in Chapter1, is correct and I can now return to *looking out of the window and watching other boys playing football* ! Having had tea with Mrs Catherall recently I know that I can now do this with her permission.

References

Abate, J. and Whitt, W., 2006. A Unified Framework for Numerically Inverting Laplace Transforms. *INFORMS Journal on Computing*, 18(4), pp.408-421.

Abraham, S., 2010. *The History of Options Contracts*.
Available at :
<http://www.investopedia.com/articles/optioninvestor/10/history-options-futures.asp>
[Accessed 29th May 2013]

Abrams, S.L., Cho, W., Hu, C.-Y., Maekawa, X.Y. and Patrikalakis, N.M., 1998. Efficient and Reliable Methods for Rounded Interval Arithmetic. *Computer-Aided Design*, 30(8), pp.657-665.

Abramowitz, M. and Stegum, T.,1972. *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables*.
Available at : <http://www.math.ucla.edu/~cbm/aands/page_775.htm>
[Accessed 26th November 2010]

Ankudinova, J. and Ehrhardt, M., 2008. On the Numerical Solution of Nonlinear Black-Scholes Equations. *Computers and Mathematics with Applications*, 56(3), pp.799-812.

Aral, M. and Gülçat, U., 1977. A finite element Laplace transform solution technique for the wave equation. *International Journal for Numerical Methods in Engineering*, 11(11), pp.1719-1732.

Ashi, H., 2008. *Numerical Methods for Stiff Systems*. Ph.D. University of Nottingham.

Bachelier, L., 1900. Théorie de la Spéculation, Annales de l'Ecole Normale Supérieure, 17.

Bal, G. and Maday, Y., 2002. A "parareal" time discretization for non-linear PDEs with applications to the pricing of an American put. *Lecture Notes in Computer Science and Engineering*, 23, pp.189-202.

Baldo, L., Brenner, L., Fernandes, L., Fernandes, P. and Sales, A., 2005. Performance Models for Master/Slave Parallel Programs. *Electronic Notes in Theoretical Computer Science*, 128(4), pp.101-121.

Barles, G. and Soner, H.M., 1998. Option Pricing with Transaction Costs and a Nonlinear Black-Scholes Equation. *Finance and Stochastics*, 42, pp.369-397.

Barney, B., 2010. *Introduction to Parallel Computing*.
Available at : <https://computing.llnl.gov/tutorials/parallel_comp/#ExamplesHeat>
[Accessed 13th December 2010]

Barua, S., Thulasiram, R.K., and Thulasiraman, P., 2004. Fast Fourier Transform for Option Pricing : Improved Mathematical Modeling and Design of Efficient Parallel Algorithm. *Lecture Notes in Computer Science*, 3045, pp.686-695.

Beerends, R.J., ter Morsche, H.G., van den Berg, J.C., and van de Vrie, E.M., 2003. *Fourier and Laplace Transforms*. Cambridge. Cambridge University Press.

Benhamou, E., 2008. *Options, pre Black-Scholes*.
Available at : <http://www.ericbenhamou.net/documents/Encyclo/PreBlack-Scholes.pdf>
[Accessed 10th May 2013]

Black, F. and Scholes, M., 1973. The Pricing of Options and Corporate Liabilities. *The Journal of Political Economy*, 81(3), pp.636-654.

Box, G.E.P., and Muller, M.E., 1958. A note on the generation of random normal deviates. *Annals of Mathematical Statistics*, 29, pp.610-611.

Boyle, P.P., 1977. Options : A Monte Carlo Approach. *Journal of Financial Economics*, 4(3), pp.323-338.

Boyle, P.P. and Vorst, T., 1992. Option Replication in Discrete Time with Transaction Costs. *The Journal of Finance*, XLVII(1), pp.271-293.

Buetow, G. and Sochacki, J., 2000. The trade-offs between alternative finite difference techniques used to price derivative securities. *Applied Mathematics and Computation*, 115(2000), pp.177-190.

Burkardt, J., 2010. *TIMER Compute Elapsed Time*.
Available at : <http://people.sc.fsu.edu/~jburkardt/f_src/timer/timer.html>
[Accessed 2nd July 2010]

Chapra, S.C. and Canale, R.P., 2010. *Numerical Methods for Engineers*. 6th ed. New York. McGraw-Hill.

Chen, H-T. and Lin, J-Y., 1991. Hybid Laplace Transform Techniques for Nonlinear Transient Thermal Problems. *International Journal of Heat Mass Transfer*, 34(4/5), pp.1301-1308.

Craddock, M.J., Heath, D.P. and Platen, E., 2000. Numerical Inversion of Laplace Transforms : A Survey of Techniques with Applications to Derivative Pricing. *Journal of Computational Finance*, 4(1), pp.57-81.

Crann, D., 2005. *The Laplace Transform Boundary Element Method For Diffusion-type Problems*. Ph.D. University of Hertfordshire.

Crann, D., Davies, A., Lai, C-H. and Leong, S., 1998. Time Domain Decomposition for European Options in Financial Modelling. *Contemporary Mathematics, American Mathematical Society*, 18, pp.486-491.

Crann, D., Kane, S., Davies, A. and Lai, C-H., 2007. A time-domain decomposition method for the parallel boundary element solution of diffusion problems. In: Advances in Boundary Element Techniques VIII, *8th International Conference on Boundary Element Techniques*. Naples, Italy, 24-26 July 2007, Eastleigh: EC Ltd.

Davies, B., 2002. *Integral Transforms and Their Applications*. 3rd ed. New York. Springer-Verlag.

Davies, A. and Crann, D., 2004. *A handbook of essential mathematical formulae*. Hertfordshire. University of Hertfordshire Press.

Davies, A. and Crann, D., 2010. On Laplace transform time-domain decomposition for dual reciprocity solution of diffusion problems. *Workshop on Applied and Numerical Mathematics*. Greenwich University, UK, 15 April 2010, Greenwich: Greenwich University.

Davies, A., Crann, D., Kane, S. and Lai, C-H., 2007. *A hybrid Laplace transform/finite difference boundary element method for diffusion problems.*
Available at : <http://www.techscience.com/cmes/2007/v18n2_index.html>
[Accessed 5th May 2011]

Davies, B. and Martin, B., 1979. Numerical Inversion of the Laplace Transform : A Survey and Comparison of Methods. *Journal of Computational Physics*, 33(1), pp.1-32.

Deakin, M., 1992. The Ascendancy of the Laplace Transform and how it Came About. *The Archive for History* of *Exact Sciences*, 44(3), pp.265-286.

Düring, B., Fournié, M. and Jüngel, A., 2003. High Order Compact Finite Difference Schemes for a Nonlinear Black-Scholes Equation. *International Journal of Applied Theoretical Finance*, 7, pp.767-789.

Edwards, C. and Penney, D., 2008. *Elementary Differential Equations*. 6th ed. Upper Saddle River, New Jersey. Pearson Education Inc.

Epstein, C. and Schotland, J., 2008. The Bad Truth about Laplace's Transform. *SIAM*, 50(3), pp. 504-520.

Fitzharris, A., Kane, S. and Lai, C-H., 2012. Numerically Inverting Laplace Transforms in a Distributed Computing Environment. *Workshop on Applied and Numerical Mathematics*. Greenwich University, UK, 8 June 2012, Greenwich: Greenwich University.

Gaver, D., 1966. Observing stochastic processes, and approximate transform inversion. *Operational Research*, 14(3), pp.444-459.

Geist, G., Kohl, J. and Papadopoulos,P.,1996. *PVM and MPI: A Comparison of Features*.
Available at :
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.4809&rep=rep1&type=pdf>
[Accessed 8th November 2010]

Geske, R., 1979. The Valuation of Compound Options. *Journal of Financial Economics*, 7, pp.63-81.

Gladwell, I. and Sayers, D.K. (ed.), 1980. *Computational Techniques for Ordinary Differential Equations*. Waltham, Massachusetts. Academic Press.

Glasserman, P., 2003. *Monte Carlo Methods in Financial Engineering*. New York. Springer.

Grama, A., Gupta, A., Karypis, G. and Kumar, V., 2003. *Introduction to Parallel Computing*. 2nd ed. Harlow. Pearson Education.

Gropp, W., Lusk, E. and Skjellum, A., 1999. *Using MPI : Portable Parallel Programming with the Message Passing Interface.* London. MIT Press.

Hadamard, J., 1923. *Lectures on Cauchy's problem in linear partial differential equations*. New York. Dover Publications.

Heston, S., 1993. A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *The Review of Financial Studies,* 6(2), pp.327-343.

Hirsa, A.,2012. *Computational Methods in Finance*. New York. Chapman & Hall.

Horak, V. and Gruber, P., 2002. Parallel Numerical Solution of 2-D Heat Equation. *Parallel Numerics*, (5), pp.47-56.

Hull, J. and White, A., 1990. Valuing Derivative Securities Using the Explicit Finite Difference Method. *Journal of Financial and Quantitative Analysis*, 25(1), pp.87-100.

Jäckel , P., 2002. *Monte Carlo Methods in Finance*. London. John Wiley & Sons, Inc.

Jäckel , P., 2005. *Stochastic Volatility Models : Past, Present and Future.*
Available at :
<http://www.bfi.d/papers/Jackel-Stochastic Volatility Models Past Present And Future.pdf>
[Accessed 1st June 2013]

Jandačka, M. and Ševčovič, D., 2005. On the Risk-Adjusted Pricing-Methodology of Vanilla Options and Explanation of the Volatility Smile. *Journal of Applied Mathematics*, pp.235-258.

Jeong, D. Kim, J. and Wee, I-S. 2009. An Accurate and Efficient Numerical Method For Black-Scholes Equations. *Communications of the Korean Mathematical Society*, 24(4), pp.617-628.

Kano, P.O., 2010. *Numerical Laplace Transform Inversion and Selected Applications.* [pdf]
Available at :
<http://math.arizona.edu/~brio/WEEKS_METHOD_PAGE/NLAPcolloqium2010.pdf>
[Accessed 25th June 2012]

Kratka, M. 1998. No Mystery Behind the Smile. *Risk*, 9, pp. 67-71.

Kuhlman, K.L., 2012. Review of Inverse Laplace Transform Algorithms for Laplace-Space Numerical Approaches. *Numerical Algorithms*, 61(174).

Lai, C-H., Parrott, A., Rout, S. and Honnor, M., 2005. A Distributed Algorithm for European Options with Non-linear Volatility. *Computers and Mathematics with Applications*, 49, pp.885-894.

Lantzman, E. 2007. *Iterative vs Recursive Approaches*.
Available at : <www.codeproject.com/Articles/21194/Iterative-vs-Recursive-Approaches>
[Accessed 20th December 2012]

Laverty, R., 2003. Laplace Transform Inversion and Viscoelastic Wave Propagation. In: *Proceedings of the 11th Annual ARL/USMA Technical Symposium.* Aberdeen, Maryland, USA, November 2003, (unpublished).

Lee, H. and Sheen, D., 2009. Laplace Transform Method for the Black-Scholes Equation. *International Journal of Numerical Analysis and Modelling*, 6(4), pp 1-17.

L'Ecuyer, P., 1999. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 18, pp 816-822.

Leentvaar, C.C.W. and Oosterlee, C.W., 2008. Multi-Asset Option Pricing Using a Parallel Fourier-Based Technique. *Journal of Computational Finance*, 12(1), pp.1-26.

Leland, H.E., 1985. Option Replication in Discrete Time with Transaction Costs. *Journal of Finance*, 40, pp.1283-1301.

Liao, W., Zhu, J. and Khaliq, A., 2001. An Efficient High-Order Algorithm for Solving Systems of Reaction-Diffusion Equations. *Numerical Methods for Partial Differential Equations*, 18(3), pp.340-354.

Mackay, D.J.C., 2004. *Information Theory, Inference and Learning Algorithms*. Cambridge. Cambridge University Press.

Magoules, F. ed., 2010. *Fundamentals of Grid Computing, Theory, Algorithms and Technologies*. Boca Raton:USA. Chapman and Hall.

McKhann, C., 2006. *Introducing the VIX Options*. Available at :
<http://www.investopedia.com/articles/optioninvestor/06/NewVIX.asp#axzz1PmwsKR1s>
[Accessed 12th June 2011]

McWhirter, J. and Pike, E. 1978. On the numerical inversion of the Laplace transform and similar Fredholm integral equations of the first kind. *Journal of Physics and Applied Mathematics*, (11), pp 1729-1745.

Metcalf, M. and Reid, J., 2006. *Fortran 90/95 Explained*. 2nd ed. Oxford. Oxford University Press.

Miller, M. and Guy, W., 1966. Numerical Inversion of the Laplace Transform by Use of Jacobi Polynomials. *SIAM Journal on Numerical Analysis*, 3(4), pp. 624-635.

Moore, R.E., Kearfott, R.B. and Cloud, M.J., 2009. *Introduction to Interval Analysis*. Philadelphia. SIAM Press.

Morton, K.W. and Mayers, D.F., 2008. *Numerical Solution of Partial Differential Equations - An Introduction*. Cambridge. Cambridge University Press.

NanoDotTek, 2007. *The Laplace Transform Approach to Linear Transmission Line Analysis* [pdf] Available at : <http://www.nanodottek.com/NDT18_09_2007.pdf> [Accessed 24th June 2012].

Natkunam, K., 2009. *Numerical Analysis of the Diffusion Equation Using Laplace Transform*. M.Sc. Greenwich University.

Piessens, R. and Branders, M., 1971. Numerical Inversion of the Laplace Transform Using Generalized Laguerre Polynomials. *Proceedings of the IEE,* 118, pp.1517-1522.

Pliska, S.R., 2010. *A History of Options from the Middle Ages to Harrison and Kreps*. Available at : <http://www.fields.utoronto.ca/programs/scientific/09-10/finance/courses/pliska1.pdf> [Accessed 3rd May 2013].

Quarteroni, A. and Valli, A.,1999. *Domain Decomposition Methods for Partial Differential Equations*. Oxford. Oxford University Press.

Reid, J., 1994. *The Advantages of Fortran 90*. Available at : <ftp://ftp.numerical.rl.ac.uk/pub/reports/rRAL92017.pdf> [Accessed 18th August 2011]

Risken, P., 1996. *The Fokker-Planck Equation*. London. Springer-Verlag.

Sauer, T., 2012. Numerical solution of stochastic differential equations in finance. *Handbook of Computational Finance*, pp 529-550.

Schiller, J.J., Srinivason, R.A. and Spiegel, M., 2008. *Probability and Statistics*. 3rd ed. London. McGraw-Hill Book Company.

Sharcnet, G.B., 2008. *Parallel Numerical Solution of PDEs with Message Passing.pdf* Available at : <https://www.sharcnet.ca/help/images/4/4f/Parallel_Numerical_Solution_ of_PDEs_with_Message_Passing> [Accessed 3rd May 2013]

Shreve, S.E., 2010. *Stochastic Calculus for Finance II*. New York. Springer.

Skachkov, I., 2002. *Black-Scholes Equation in Laplace Transform Domain*. Available at : <http://www.wilmott.com> [Accessed 1st August 2011]

Smith,G.D., 2004. *Numerical Solution of Partial Differential Equations : Finite Difference Methods.* 3rd ed. Oxford. Oxford University Press.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J., 1996. *MPI : The Complete Reference*. London. The MIT Press.

Stehfest, H., 1970. Numerical Inversion of Laplace Transforms. *Communications of the ACM,* 13(1), pp.47-49.

Sunderam, V., 1990. PVM : A framework for parallel distributed computing. *Concurrency : Practice & Experience*, 2(4), pp.315-339.

Tagliani, A. and Milev, M., 2012. Laplace Transform and Finite Difference Methods for the Black-Scholes Equation. *Journal of Applied Mathematics and Computations* (Submitted).

Ulam, S., 1983. *Adventures of a Mathematician*. New York. Charles Scribner's Sons.

Wang, Y., Hua, K. and Zhang, J., 2011. Fast and High Accuracy Numerical Methods for Solving PDEs in Computational Finance. In: *Proceedings of the 2011 International Conference on Business Computing and Global Informatization*. Shanghai, China, July 2011.

Wang, A.M-L., Liu, Y-H. and Hsiao, Y-L., 2009. Barrier Option Pricing : A Hybrid Approach. *Quantitative Finance*, 9(3), pp.341-352.

Weeks, W., 1966. Numerical Inversion of Laplace Transforms Using Laguerre Functions. *Journal of the Association for Computing Machinery*, 13(3), pp.419-426.

Widder, D., 1946. *The Laplace Transform*. Princeton. Princeton University Press.

Wilmot, P., 2000. *Derivatives. The Theory and Practice of Financial Engineering*. Chichester. John Wiley.

Wilmot, P., Howison, S. and Dewynne, J., 1999. *The Mathematics of Financial Derivatives*. Cambridge. Cambridge University Press.

Wing, G., 1991. *A primer on integral equations of the first kind : the problem of deconvolution and unfolding*. Philadelphia : SIAM.

Wood, W.L., 1990. *Practical Time-stepping Schemes*. Oxford. Clarendon Press.

Zakian, V. and Littlewood, R.,1973. Numerical inversion of Laplace transforms by weighted least-squares approximation. *The Computer Journal*, 16(1), pp.66-68.

Zhu, S-P., 1999. Time-dependent reaction-diffusion problems and the LT-DRM approach. *Boundary Integral Methods, Numerical and Mathematical Aspects*, pp.1-35.

Appendix A

STRI Cluster Specification

Hardware

An 96-node cluster/blade system in which 80 nodes each contain two Xeon E5520, 2.27 GHz, quad-core processors and 16 nodes each contain two Xeon X5650, 2.67 GHz, 6-core processors. The PAM sub-system contains 52 nodes. The CAIR sub-system contains 44 nodes.

Memory

24Gb RAM running over Infini band High Speed Interconnect (to provide low latency/fast inter-node communications).

Operating System

64-bit Red HAT Linux

User Software

Fortran 90 and MPI CH2.

Appendix B

Performance Data for Laplace Transform Solutions - Initial Investigations

Analytical LT

Optimal Sequential Programs Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	6	0.00760937300
Minimum Wall Time (s) :	6	7.40825700800
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	26	0.02683993270
Minimum Wall Time (s) :	6	24.54517198000
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	16	0.09222718870
Minimum Wall Time (s) :	6	8.39083504700
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	12	0.08118930914
Minimum Wall Time (s) :	6	124.50031500000

Table B.1 Optimal Sequential Programs Data (Analytical LT)

Part 1 : Optimal Weights/Terms Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	6	0.00753181310
Minimum Wall Time (s) :	6	0.64976596830
Parallel/Sequential Speed Up :	14	16.28454153804
Parallel/Sequential Speed Up/Processor :	14	0.77545435895
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	24	0.12780485760
Minimum Wall Time (s) :	6	1.76125001900
Parallel/Sequential Speed Up :	22	19.52100606309
Parallel/Sequential Speed Up/Processor :	22	0.92957171729
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	16	0.09132087274
Minimum Wall Time (s) :	6	0.73899102210
Parallel/Sequential Speed Up :	24	22.16991136229
Parallel/Sequential Speed Up/Processor :	24	1.05571006487
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	12	0.08118930914
Minimum Wall Time (s) :	6	7.27635908100
Parallel/Sequential Speed Up :	22	19.36801828272
Parallel/Sequential Speed Up/Processor :	22	0.92228658489

Table B.2 Optimal Weights/Terms Data, Parallel Programs (Analytical LT)

Part 2 : Optimal Processors Data

	Stehfest's Method	
	Optimal Processors	**Optimal Value**
NRMSD :	24	0.00756876468
Minimum Wall Time (s) :	40	0.50650691880
Parallel/Sequential Speed Up :	40	14.62617139673
Parallel/Sequential Speed Up/Processor :	5	1.05105894946
	SLP Method	
	Optimal Processors	**Optimal Value**
NRMSD :	6	0.12879715500
Minimum Wall Time (s) :	8	3.20535020800
Parallel/Sequential Speed Up :	8	7.65756325744
Parallel/Sequential Speed Up/Processor :	8	0.95719540718
	Jacobi Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	40	0.09168081284
Minimum Wall Time (s) :	56	1.38076806100
Parallel/Sequential Speed Up :	56	6.07693303749
Parallel/Sequential Speed Up/Processor :	4	0.94456032006
	Laguerre Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	136	0.07924151668
Minimum Wall Time (s) :	136	4.13765192000
Parallel/Sequential Speed Up :	136	30.08960574915
Parallel/Sequential Speed Up/Processor :	7	0.79203734133

Table B.3 Optimal Processors Data (Analytical LT)

BVP LT

Optimal Sequential Programs Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	6	0.04177589251
Minimum Wall Time (s) :	6	2.13941407200
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	14	0.08125714247
Minimum Wall Time (s) :	8	2.91407489800
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	10	0.12070521050
Minimum Wall Time (s) :	6	3.27581691700
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	24	0.03691509723
Minimum Wall Time (s) :	6	25.10763097000

Table B.4 Optimal Sequential Programs Data (BVP LT)

Part 1 : Optimal Weights/Terms Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	6	0.04181540563
Minimum Wall Time (s) :	6	0.19930100440
Parallel/Sequential Speed Up :	16	12.78437411915
Parallel/Sequential Speed Up/Processor :	16	0.60877971996
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	26	0.03302565905
Minimum Wall Time (s) :	6	0.33695888520
Parallel/Sequential Speed Up :	6	15.88224930417
Parallel/Sequential Speed Up/Processor :	6	0.75629758591
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	10	0.12082799120
Minimum Wall Time (s) :	6	0.43632006650
Parallel/Sequential Speed Up :	16	15.78114067238
Parallel/Sequential Speed Up/Processor :	16	0.75148288916
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	24	0.03692263067
Minimum Wall Time (s) :	6	1.69342899300
Parallel/Sequential Speed Up :	20	18.95694137668
Parallel/Sequential Speed Up/Processor :	20	0.90271149413

Table B.5 Optimal Weights/Terms Data, Parallel Programs (BVP LT)

Part 2 : Optimal Processors Data

	Stehfest's Method	
	Optimal Processors	**Optimal Value**
NRMSD :	136	0.04078000884
Minimum Wall Time (s) :	24	0.29325103760
Parallel/Sequential Speed Up :	24	7.29550384377
Parallel/Sequential Speed Up/Processor :	5	0.93928715166
	SLP Method	
	Optimal Processors	**Optimal Value**
NRMSD :	24	0.08084034111
Minimum Wall Time (s) :	40	0.46209406850
Parallel/Sequential Speed Up :	40	6.30623740196
Parallel/Sequential Speed Up/Processor :	6	1.00435101795
	Jacobi Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	24	0.12008411140
Minimum Wall Time (s) :	40	0.73716402050
Parallel/Sequential Speed Up :	40	4.44381009640
Parallel/Sequential Speed Up/Processor :	5	0.70881039099
	Laguerre Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	7	0.03812053239
Minimum Wall Time (s) :	136	1.58626890200
Parallel/Sequential Speed Up :	136	15.82810514557
Parallel/Sequential Speed Up/Processor :	8	1.47589113525

Table B.6 Optimal Processors Data (BVP LT)

Appendix C

Performance Data for the One-Dimensional, Linear Black-Scholes Equation

Part 1 : Optimal Weights/Terms Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	6	0.03282266316
Minimum Wall Time (s) :	6	1.21327510800
Parallel/Sequential Speed Up :	6	5.61809507510
Parallel/Sequential Speed Up/Processor :	6	0.26752833691
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	14	0.06796623244
Minimum Wall Time (s) :	10	1.32123372700
Parallel/Sequential Speed Up :	10	5.15903792774
Parallel/Sequential Speed Up/Processor :	10	0.24566847275
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	10	0.02524845549
Minimum Wall Time (s) :	8	1.35029285600
Parallel/Sequential Speed Up :	8	5.04801227283
Parallel/Sequential Speed Up/Processor :	8	0.24038153680
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	20	0.00247189988
Minimum Wall Time (s) :	6	1.38424802200
Parallel/Sequential Speed Up :	6	4.92418612898
Parallel/Sequential Speed Up/Processor :	6	0.23448505376

Table C.1 Optimal Weights/Terms Data, Parallel Programs

Part 2 : Optimal Processors Data

	Stehfest's Method	
	Optimal Processors	**Optimal Value**
NRMSD :	4	0.03190892874
Minimum Wall Time (s) :	40	1.20931506200
Parallel/Sequential Speed Up :	40	5.63649219561
Parallel/Sequential Speed Up/Processor :	4	0.59781910392
	SLP Method	
	Optimal Processors	**Optimal Value**
NRMSD :	3	0.03834257471
Minimum Wall Time (s) :	72	1.43498492200
Parallel/Sequential Speed Up :	72	4.75008120608
Parallel/Sequential Speed Up/Processor :	3	0.58250843613
	Jacobi Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	4	0.00088366262
Minimum Wall Time (s) :	56	1.35022829500
Parallel/Sequential Speed Up :	56	5.04825364291
Parallel/Sequential Speed Up/Processor :	4	0.54960908927
	Laguerre Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	24	0.00275564186
Minimum Wall Time (s) :	88	1.80055131100
Parallel/Sequential Speed Up :	88	3.78567101496
Parallel/Sequential Speed Up/Processor :	3	0.57348748911

Table C.2 Optimal Processors Data

Appendix D

Performance Data for the One-Dimensional, Nonlinear Black-Scholes Equations

Modified Volatility Function : Simulated Modified Volatility

Part 1 : Optimal Weights/Terms Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	24	0.02799328572
Minimum Wall Time (s) :	6	0.12257361320
Parallel/Sequential Speed Up :	6	4.36005651827
Parallel/Sequential Speed Up/Processor :	6	0.20762173897
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	24	0.01175761424
Minimum Wall Time (s) :	6	0.13552594180
Parallel/Sequential Speed Up :	6	3.94336223827
Parallel/Sequential Speed Up/Processor :	6	0.18777915420
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	20	0.01464839485
Minimum Wall Time (s) :	6	0.15520596500
Parallel/Sequential Speed Up :	6	3.44334627345
Parallel/Sequential Speed Up/Processor :	6	0.16396887016
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	6	0.15206770190
Minimum Wall Time (s) :	6	0.31046605110
Parallel/Sequential Speed Up :	6	1.72137301102
Parallel/Sequential Speed Up/Processor :	6	0.08197014338

Table D.1 Optimal Weights/Terms Data, Parallel Programs (Simulated Modified Volatility)

Part 2 : Optimal Processors Data

	Stehfest's Method	
	Optimal Processors	**Optimal Value**
NRMSD :	3	0.08441731708
Minimum Wall Time (s) :	8	0.14818406110
Parallel/Sequential Speed Up :	8	3.60651393431
Parallel/Sequential Speed Up/Processor :	5	0.54030901625
	SLP Method	
	Optimal Processors	**Optimal Value**
NRMSD :	72	0.07853856512
Minimum Wall Time (s) :	8	0.15288754380
Parallel/Sequential Speed Up :	8	3.49556195303
Parallel/Sequential Speed Up/Processor :	5	0.52253271423
	Jacobi Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	88	0.07562827257
Minimum Wall Time (s) :	8	0.15593290330
Parallel/Sequential Speed Up :	8	3.42729385454
Parallel/Sequential Speed Up/Processor :	4	0.50692215624
	Laguerre Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	3	0.09052401633
Minimum Wall Time (s) :	8	0.21030807500
Parallel/Sequential Speed Up :	8	2.54116672030
Parallel/Sequential Speed Up/Processor :	4	0.46816060980

Table D.2 Optimal Processors Data (Simulated Modified Volatility)

Modified Volatility Function : Leland

Part 1 : Optimal Weights/Terms Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	6	0.01974218390
Minimum Wall Time (s) :	6	1.59266995400
Parallel/Sequential Speed Up :	6	5.03589956717
Parallel/Sequential Speed Up/Processor :	6	0.23980474129
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	12	0.02785605212
Minimum Wall Time (s) :	6	1.61791706100
Parallel/Sequential Speed Up :	6	4.95731587566
Parallel/Sequential Speed Up/Processor :	6	0.23606266075
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	10	0.03238814544
Minimum Wall Time (s) :	6	1.70022711800
Parallel/Sequential Speed Up :	6	4.71732620136
Parallel/Sequential Speed Up/Processor :	6	0.22463458102
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	20	0.00769955575
Minimum Wall Time (s) :	6	1.71153502500
Parallel/Sequential Speed Up :	6	4.68615939192
Parallel/Sequential Speed Up/Processor :	6	0.22315044723

Table D.3 Optimal Weights/Terms Data, Parallel Programs (Leland)

Part 2 : Optimal Processors Data

	Stehfest's Method	
	Optimal Processors	**Optimal Value**
NRMSD :	6	0.01043302431
Minimum Wall Time (s) :	40	1.24268794100
Parallel/Sequential Speed Up :	40	6.45417539463
Parallel/Sequential Speed Up/Processor :	6	0.62382862880
	SLP Method	
	Optimal Processors	**Optimal Value**
NRMSD :	3	0.01656271483
Minimum Wall Time (s) :	40	1.28281211900
Parallel/Sequential Speed Up :	40	6.25229978202
Parallel/Sequential Speed Up/Processor :	4	0.60435156532
	Jacobi Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	3	0.00869205162
Minimum Wall Time (s) :	24	1.29155802700
Parallel/Sequential Speed Up :	24	6.20996173949
Parallel/Sequential Speed Up/Processor :	4	0.58678285572
	Laguerre Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	3	0.07142338569
Minimum Wall Time (s) :	24	1.31553101500
Parallel/Sequential Speed Up :	24	6.09679729368
Parallel/Sequential Speed Up/Processor :	4	0.58272848756

Table D.4 Optimal Processors Data (Leland)

Modified Volatility Function : Boyle and Vorst

Part 1 : Optimal Weights/Terms Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	18	0.03814824315
Minimum Wall Time (s) :	6	1.36069107100
Parallel/Sequential Speed Up :	6	5.65549822220
Parallel/Sequential Speed Up/Processor :	6	0.26930943915
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	20	0.04437350128
Minimum Wall Time (s) :	6	1.42684006700
Parallel/Sequential Speed Up :	6	5.39330658774
Parallel/Sequential Speed Up/Processor :	6	0.25682412323
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	14	0.03398463438
Minimum Wall Time (s) :	6	1.44844105700
Parallel/Sequential Speed Up :	6	5.31287475994
Parallel/Sequential Speed Up/Processor :	6	0.25299403619
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	16	0.01247211126
Minimum Wall Time (s) :	6	1.48429989800
Parallel/Sequential Speed Up :	6	5.18452230804
Parallel/Sequential Speed Up/Processor :	6	0.24688201467

Table D.5 Optimal Weights/Terms Data, Parallel Programs (Boyle and Vorst)

Part 2 : Optimal Processors Data

	Stehfest's Method	
	Optimal Processors	**Optimal Value**
NRMSD :	7	0.03483109684
Minimum Wall Time (s) :	40	1.43203806900
Parallel/Sequential Speed Up :	40	5.37373000033
Parallel/Sequential Speed Up/Processor :	5	0.56982627545
	SLP Method	
	Optimal Processors	**Optimal Value**
NRMSD :	136	0.01167072834
Minimum Wall Time (s) :	40	1.44086366710
Parallel/Sequential Speed Up :	40	5.34081475487
Parallel/Sequential Speed Up/Processor :	5	0.56499791628
	Jacobi Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	3	0.02169106686
Minimum Wall Time (s) :	40	1.50040064800
Parallel/Sequential Speed Up :	40	5.12888736969
Parallel/Sequential Speed Up/Processor :	3	0.55831331777
	Laguerre Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	3	0.05148669002
Minimum Wall Time (s) :	40	1.53728062900
Parallel/Sequential Speed Up :	40	5.00584329746
Parallel/Sequential Speed Up/Processor :	3	0.55399455492

Table D.6 Optimal Processors Data (Boyle and Vorst)

Modified Volatility Function : Barles and Soner

Part 1 : Optimal Weights/Terms Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	24	0.03671890285
Minimum Wall Time (s) :	6	0.42293095590
Parallel/Sequential Speed Up :	6	9.26509371172
Parallel/Sequential Speed Up/Processor :	6	0.44119493865
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	22	0.01325309981
Minimum Wall Time (s) :	6	0.45432901380
Parallel/Sequential Speed Up :	6	8.62479573388
Parallel/Sequential Speed Up/Processor :	6	0.41070455876
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	16	0.00743902902
Minimum Wall Time (s) :	6	0.47075891490
Parallel/Sequential Speed Up :	6	8.32378276008
Parallel/Sequential Speed Up/Processor :	6	0.39637060762
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	16	0.14538297780
Minimum Wall Time (s) :	6	0.55315589900
Parallel/Sequential Speed Up :	6	7.08388891284
Parallel/Sequential Speed Up/Processor :	6	0.33732804347

Table D.7 Optimal Weights/Terms Data, Parallel Programs (Barles and Soner)

Part 2 : Optimal Processors Data

	Stehfest's Method	
	Optimal Processors	**Optimal Value**
NRMSD :	3	0.08733824007
Minimum Wall Time (s) :	24	0.46691107500
Parallel/Sequential Speed Up :	24	8.39237951252
Parallel/Sequential Speed Up/Processor :	6	0.79714300681
	SLP Method	
	Optimal Processors	**Optimal Value**
NRMSD :	72	0.08105728083
Minimum Wall Time (s) :	24	0.48729245500
Parallel/Sequential Speed Up :	24	8.04136181423
Parallel/Sequential Speed Up/Processor :	4	0.73557864637
	Jacobi Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	88	0.07783334599
Minimum Wall Time (s) :	24	0.53114756650
Parallel/Sequential Speed Up :	24	7.37741295855
Parallel/Sequential Speed Up/Processor :	4	0.67525103505
	Laguerre Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	3	0.09290810470
Minimum Wall Time (s) :	24	0.66324496270
Parallel/Sequential Speed Up :	24	5.90806588873
Parallel/Sequential Speed Up/Processor :	4	0.66626441644

Table D.8 Optimal Processors Data (Barles and Soner)

Modified Volatility Function : RAPM

Part 1 : Optimal Weights/Terms Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	24	0.03746163650
Minimum Wall Time (s) :	6	0.20508289340
Parallel/Sequential Speed Up :	6	5.73830454842
Parallel/Sequential Speed Up/Processor :	6	0.27325259754
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	22	0.01711176881
Minimum Wall Time (s) :	8	0.21877288820
Parallel/Sequential Speed Up :	8	5.37922276239
Parallel/Sequential Speed Up/Processor :	8	0.25615346488
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	18	0.00801089964
Minimum Wall Time (s) :	6	0.23860311510
Parallel/Sequential Speed Up :	6	4.93215731700
Parallel/Sequential Speed Up/Processor :	6	0.23486463414
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	6	0.15699258450
Minimum Wall Time (s) :	6	0.35088992120
Parallel/Sequential Speed Up :	6	3.35383842310
Parallel/Sequential Speed Up/Processor :	6	0.15970659158

Table D.9 Optimal Weights/Terms Data, Parallel Programs (RAPM)

Part 2 : Optimal Processors Data

	Stehfest's Method	
	Optimal Processors	**Optimal Value**
NRMSD :	3	0.08667780605
Minimum Wall Time (s) :	24	0.19593906400
Parallel/Sequential Speed Up :	24	6.00609228183
Parallel/Sequential Speed Up/Processor :	3	0.63232795845
	SLP Method	
	Optimal Processors	**Optimal Value**
NRMSD :	56	0.08140492274
Minimum Wall Time (s) :	24	0.27529001240
Parallel/Sequential Speed Up :	24	4.27486667511
Parallel/Sequential Speed Up/Processor :	3	0.62214548859
	Jacobi Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	88	0.07725559308
Minimum Wall Time (s) :	24	0.31821552970
Parallel/Sequential Speed Up :	24	3.69821077277
Parallel/Sequential Speed Up/Processor :	3	0.64110825564
	Laguerre Polynomial Method	
	Optimal Processors	**Optimal Value**
NRMSD :	3	0.09390578313
Minimum Wall Time (s) :	8	0.39627796010
Parallel/Sequential Speed Up :	8	2.96970363858
Parallel/Sequential Speed Up/Processor :	3	0.53086674771

Table D.10 Optimal Processors Data (RAPM)

Appendix E

Performance Data for the Two-Dimensional, Linear Black-Scholes Equation

Optimal Sequential Programs Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	8	0.21517680360
Minimum Wall Time (s) :	6	20.80843711000
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	26	0.06469810112
Minimum Wall Time (s) :	6	21.01788760000
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	6	0.32694111930
Minimum Wall Time (s) :	6	23.67294897000
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	12	2.26040329500
Minimum Wall Time (s) :	6	35.01823711000

Table E.1 Optimal Sequential Programs Data

Optimal Parallel Programs Data

	Stehfest's Method	
	Optimal Weights (Processors)	**Optimal Value**
NRMSD :	8 (9)	0.21517680360
Minimum Wall Time (s) :	6 (7)	6.85409307500
Parallel/Sequential Speed Up :	26 (27)	7.34611276972
Parallel/Sequential Speed Up/Processor :	6 (7)	0.43370199971
	SLP Method	
	Optimal Weights (Processors)	**Optimal Value**
NRMSD :	26 (27)	0.06469810112
Minimum Wall Time (s) :	6 (7)	7.29012602700
Parallel/Sequential Speed Up :	22 (23)	6.52842579693
Parallel/Sequential Speed Up/Processor :	8 (9)	0.43612267555
	Jacobi Polynomial Method	
	Optimal Terms (Processors)	**Optimal Value**
NRMSD :	6 (7)	0.32694111930
Minimum Wall Time (s) :	8 (9)	7.83274439800
Parallel/Sequential Speed Up :	18 (19)	6.55196378857
Parallel/Sequential Speed Up/Processor :	8 (9)	0.53161789599
	Laguerre Polynomial Method	
	Optimal Terms (Processors)	**Optimal Value**
NRMSD :	6 (7)	0.32345620810
Minimum Wall Time (s) :	6 (7)	10.14124416000
Parallel/Sequential Speed Up :	26 (27)	14.40714439882
Parallel/Sequential Speed Up/Processor :	18 (19)	0.60357633598

Table E.2 Optimal Parallel Programs Data

Appendix F

Performance Data for the Two-Dimensional, Nonlinear Black-Scholes Equations

Modified Volatility Function : Simulated Modified Volatility

Optimal Sequential Programs Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	14	0.12686007900
Minimum Wall Time (s) :	8	4.55571794500
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	26	0.02777761927
Minimum Wall Time (s) :	6	4.84069204300
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	22	0.02772967769
Minimum Wall Time (s) :	6	6.28118519900
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	20	2.09002997900
Minimum Wall Time (s) :	6	10.92732000000

Table F.1 Optimal Sequential Programs Data (Simulated Modified Volatility)

Optimal Parallel Programs Data

	Stehfest's Method	
	Optimal Weights (Processors)	**Optimal Value**
NRMSD :	14 (15)	0.12686007900
Minimum Wall Time (s) :	12 (13)	1.28097391100
Parallel/Sequential Speed Up :	18 (19)	6.31066925025
Parallel/Sequential Speed Up/Processor :	6 (7)	0.43969769459
	SLP Method	
	Optimal Weights (Processors)	**Optimal Value**
NRMSD :	26 (27)	0.02777761927
Minimum Wall Time (s) :	10 (11)	1.41183612800
Parallel/Sequential Speed Up :	24 (25)	9.15900100771
Parallel/Sequential Speed Up/Processor :	8 (9)	0.49544555076
	Jacobi Polynomial Method	
	Optimal Terms (Processors)	**Optimal Value**
NRMSD :	22 (23)	0.02772967769
Minimum Wall Time (s) :	10 (11)	1.66243479100
Parallel/Sequential Speed Up :	20 (21)	8.62655461940
Parallel/Sequential Speed Up/Processor :	10 (11)	0.52162442114
	Laguerre Polynomial Method	
	Optimal Terms (Processors)	**Optimal Value**
NRMSD :	20 (21)	2.09002997900
Minimum Wall Time (s) :	6 (7)	3.60725692600
Parallel/Sequential Speed Up :	26 (27)	20.15634733328
Parallel/Sequential Speed Up/Processor :	20 (21)	0.81481848339

Table F.2 Optimal Parallel Programs Data (Simulated Modified Volatility)

Modified Volatility Function : Leland

Optimal Sequential Programs Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	6	0.10201155790
Minimum Wall Time (s) :	6	17.76280676000
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	26	0.14420396170
Minimum Wall Time (s) :	6	18.00465510000
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	14	0.23239645250
Minimum Wall Time (s) :	6	20.68151240000
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	26	1.77848125200
Minimum Wall Time (s) :	6	32.82941570000

Table F.3 Optimal Sequential Programs Data (Leland)

Optimal Parallel Programs Data

	Stehfest's Method	
	Optimal Weights (Processors)	**Optimal Value**
NRMSD :	6 (7)	0.10201155790
Minimum Wall Time (s) :	6 (7)	5.85089260600
Parallel/Sequential Speed Up :	26 (27)	7.45135324540
Parallel/Sequential Speed Up/Processor :	6 (7)	0.43370199963
	SLP Method	
	Optimal Weights (Processors)	**Optimal Value**
NRMSD :	26 (27)	0.14420396170
Minimum Wall Time (s) :	6 (7)	7.05418005300
Parallel/Sequential Speed Up :	26 (27)	6.54876426281
Parallel/Sequential Speed Up/Processor :	10 (11)	0.37210124750
	Jacobi Polynomial Method	
	Optimal Terms (Processors)	**Optimal Value**
NRMSD :	14 (15)	0.23239645250
Minimum Wall Time (s) :	6 (7)	7.75259545700
Parallel/Sequential Speed Up :	24 (25)	7.16688033736
Parallel/Sequential Speed Up/Processor :	10 (11)	0.45982989223
	Laguerre Polynomial Method	
	Optimal Terms (Processors)	**Optimal Value**
NRMSD :	8 (9)	0.12429605370
Minimum Wall Time (s) :	6 (7)	10.03207229000
Parallel/Sequential Speed Up :	22 (23)	10.78343361760
Parallel/Sequential Speed Up/Processor :	20 (21)	0.47000707041

Table F.4 Optimal Parallel Programs Data (Leland)

Modified Volatility Function : Boyle and Vorst

Optimal Sequential Programs Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	22	0.13956131970
Minimum Wall Time (s) :	6	18.27469492000
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	26	0.18538908410
Minimum Wall Time (s) :	6	20.03498296000
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	26	0.09513891832
Minimum Wall Time (s) :	6	22.91150455000
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	20	0.71707085730
Minimum Wall Time (s) :	6	34.48718029000

Table F.5 Optimal Sequential Programs Data (Boyle and Vorst)

Optimal Parallel Programs Data

	Stehfest's Method	
	Optimal Weights (Processors)	**Optimal Value**
NRMSD :	22 (23)	0.13956131970
Minimum Wall Time (s) :	6 (7)	5.83855523400
Parallel/Sequential Speed Up :	26 (27)	7.34697347177
Parallel/Sequential Speed Up/Processor :	6 (7)	0.44714327402
	SLP Method	
	Optimal Weights (Processors)	**Optimal Value**
NRMSD :	26 (27)	0.18538908410
Minimum Wall Time (s) :	6 (7)	7.16717707300
Parallel/Sequential Speed Up :	26 (27)	7.48280603716
Parallel/Sequential Speed Up/Processor :	6 (7)	0.39933999031
	Jacobi Polynomial Method	
	Optimal Terms (Processors)	**Optimal Value**
NRMSD :	26 (27)	0.09513891832
Minimum Wall Time (s) :	6 (7)	7.42525433300
Parallel/Sequential Speed Up :	26 (27)	6.85768528732
Parallel/Sequential Speed Up/Processor :	6 (7)	0.44080268928
	Laguerre Polynomial Method	
	Optimal Terms (Processors)	**Optimal Value**
NRMSD :	20 (21)	0.71707085730
Minimum Wall Time (s) :	6 (7)	9.77245628100
Parallel/Sequential Speed Up :	26 (27)	11.56211680346
Parallel/Sequential Speed Up/Processor :	10 (11)	0.50765187507

Table F.6 Optimal Parallel Programs Data (Boyle and Vorst)

Modified Volatility Function : Barles and Soner

Optimal Sequential Programs Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	10	0.11209377880
Minimum Wall Time (s) :	6	0.88142395020
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	26	0.06046745961
Minimum Wall Time (s) :	6	1.04478973300
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	22	0.02922910136
Minimum Wall Time (s) :	6	1.15965708400
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	20	2.13122478900
Minimum Wall Time (s) :	6	2.02804581800

Table F.7 Optimal Sequential Programs Data (Barles and Soner)

Optimal Parallel Programs Data

	Stehfest's Method	
	Optimal Weights (Processors)	**Optimal Value**
NRMSD :	10 (11)	0.11209377880
Minimum Wall Time (s) :	6 (7)	0.12844991680
Parallel/Sequential Speed Up :	6 (7)	6.86200483549
Parallel/Sequential Speed Up/Processor :	6 (7)	0.98028640507
	SLP Method	
	Optimal Weights (Processors)	**Optimal Value**
NRMSD :	26 (27)	0.06046745961
Minimum Wall Time (s) :	6 (7)	0.14122467350
Parallel/Sequential Speed Up :	8 (9)	8.43622492843
Parallel/Sequential Speed Up/Processor :	6 (7)	1.05686685226
	Jacobi Polynomial Method	
	Optimal Terms (Processors)	**Optimal Value**
NRMSD :	22 (23)	0.02922910136
Minimum Wall Time (s) :	6 (7)	0.15491573980
Parallel/Sequential Speed Up :	18 (19)	7.83658191133
Parallel/Sequential Speed Up/Processor :	6 (7)	1.06938970777
	Laguerre Polynomial Method	
	Optimal Terms (Processors)	**Optimal Value**
NRMSD :	20 (21)	2.13122478900
Minimum Wall Time (s) :	6 (7)	0.26955010150
Parallel/Sequential Speed Up :	20 (21)	7.82297027179
Parallel/Sequential Speed Up/Processor :	6 (7)	1.07483109645

Table F.8 Optimal Parallel Programs Data (Barles and Soner)

Modified Volatility Function : RAPM

Optimal Sequential Programs Data

	Stehfest's Method	
	Optimal Weights	**Optimal Value**
NRMSD :	14	0.11578045140
Minimum Wall Time (s) :	6	0.83622002600
	SLP Method	
	Optimal Weights	**Optimal Value**
NRMSD :	26	0.14152665260
Minimum Wall Time (s) :	6	0.90784164570
	Jacobi Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	22	0.02922910136
Minimum Wall Time (s) :	8	0.96888566930
	Laguerre Polynomial Method	
	Optimal Terms	**Optimal Value**
NRMSD :	20	2.13104093100
Minimum Wall Time (s) :	6	1.66152111900

Table F.9 Optimal Sequential Programs Data (RAPM)

Optimal Parallel Programs Data

	Stehfest's Method	
	Optimal Weights (Processors)	**Optimal Value**
NRMSD :	14 (15)	0.11578045140
Minimum Wall Time (s) :	6 (7)	0.09848403931
Parallel/Sequential Speed Up :	10 (11)	9.52713916082
Parallel/Sequential Speed Up/Processor :	6 (7)	1.21298846545
	SLP Method	
	Optimal Weights (Processors)	**Optimal Value**
NRMSD :	26 (27)	0.14152665260
Minimum Wall Time (s) :	6 (7)	0.10474230040
Parallel/Sequential Speed Up :	12 (13)	9.98875642001
Parallel/Sequential Speed Up/Processor :	6 (7)	1.23819758757
	Jacobi Polynomial Method	
	Optimal Terms (Processors)	**Optimal Value**
NRMSD :	22 (23)	0.02922910136
Minimum Wall Time (s) :	8 (9)	0.11454052710
Parallel/Sequential Speed Up :	10 (11)	9.71161666231
Parallel/Sequential Speed Up/Processor :	6 (7)	1.24914529301
	Laguerre Polynomial Method	
	Optimal Terms (Processors)	**Optimal Value**
NRMSD :	20 (21)	2.13104093100
Minimum Wall Time (s) :	6 (7)	0.19660789130
Parallel/Sequential Speed Up :	10 (11)	10.90384985034
Parallel/Sequential Speed Up/Processor :	6 (7)	1.20727687118

Table F.10 Optimal Parallel Programs Data (RAPM)

Appendix G

Computer Programs

General Notes

- The correctness of the sequential and parallel programs developed was established in two ways :
  - Firstly, a visual inspection of the solutions produced showed that they behaved as predicted by the boundary conditions of the equation being solved. In the one dimensional case :

    $$V(0,\tau) = 0 \text{ and } V(S,\tau) \approx S \text{ as } S \to \infty .$$

    In the two-dimensional case :

    $$V(S_1,0,\tau) = Vs_1\,\tau \qquad V(0,S_2,\tau) = Vs_2\,\tau$$

    $$V(S_1,S_2,\tau) \approx S_1 \text{ as } S_1 \to \infty \qquad V(S_1,S_2,\tau) \approx S_2 \text{ as } S_2 \to \infty$$

    where $Vs_1\,t$ and $Vs_2\,t$ are the single asset solutions of the one-dimensional, linear Black-Scholes equation
  - Secondly, the solutions produced were in close agreement with those obtained using independent methods. This was either the analytical solution of the equation or a solution obtained using a Monte Carlo algorithm
- All programs contained 'implicit none' statements
- The compiler used was the Fortran 90 compiler provided in the MPICH2 suite. Version 1.4.1p1. No compiler optimisation was performed. All programs were compiled in the same way
- In all codes the main program and all subprograms were contained in the same file. No linking was necessary. If this file was called *prog.f90*, the program was compiled using the command :

  mpif90 *prog.f90* -o *prog*

  If *prog.f90* contained a sequential program, it was run using the command :

  mpiexec -np 1 *prog*

  If *prog.f90* contained a parallel program to be run using 8 processors, it was run using the command :

  mpiexec -np 8 *prog* *etc.*

Parallel Program Templates

The following program templates are written in general terms so that they can be used by anyone wishing to develop parallel, Laplace transform based programs for solving one-dimensional and two-dimensional, linear and nonlinear diffusion equations. The implementation details for the Black-Scholes equations are given in the main chapters of this dissertation. Authors who are not interested in the NRMSD of the errors or the program wall time can omit the corresponding steps. In each template it is assumed that :

- the parallel development environment being used is the MPI. Minor adjustments may be needed for other platforms
- the PDE being solved does not have an analytical solution.

Template for the One-Dimensional, Linear Program

begin *program*

- constant, variable and array declarations (including the solution domain)
- initialise the MPI
- take the program wall time
- read in the reference solution

if *master* then

- calculate the weights/terms required by the numerical inversion algorithm
- send the weights/terms to the *slave*s
- send each *slave* details of the sub-domain for which it is responsible
- receive the NRMSD data from the *slave*s
- calculate the NRMSD
- calculate the program wall time
- display/store the NRMSD and the program wall time
- display/store the numerical solution of the PDE

else *slave*

- receive the weights/terms required by the numerical inversion algorithm from the *master*
- receive the details of the sub-domain to be processed from the *master*

if first sub-domain then

- calculate the initial condition using the initial condition of the PDE

else

- calculate the initial condition by solving the ODE BVP form of the PDE

end if

- use the reference solution to update the NRMSD data for the sub-domain

for all following rows in the sub-domain :

- calculate the solution using a finite difference method
- use the reference solution to update the NRMSD data for the sub-domain

end for

- send the NRMSD data for the sub-domain to the *master*

end if

- finalise the MPI

end *program*

Template for the One-Dimensional, Nonlinear Program

begin *program*

- constant, variable and array declarations (including the solution domain)
- initialise the MPI
- take the program wall time
- read in the reference solution

if *master* then

- send each *slave* details of the sub-domain for which it is responsible
- calculate the weights/terms required by the numerical inversion algorithm
- calculate the initial conditions required by the *slave*s
    - the first initial condition is calculated using the initial condition of the PDE
    - the other initial conditions are calculated by solving the ODE BVP form of the PDE
- send each *slave* the initial condition for its sub-domain
- receive the NRMSD data from the *slave*s
- calculate the NRMSD
- calculate the program wall time
- display/store the NRMSD and the program wall time
- display/store the numerical solution of the PDE

else *slave*

- receive the details of the sub-domain to be processed from the *master*
- receive the initial condition for the sub-domain from the *master*
- use the reference solution to update the NRMSD data for the sub-domain

for all following rows in the sub-domain :

oldsolution = solution in previous row

repeat

- calculate the nonlinear terms in the row using oldsolution
- calculate newsolution using the nonlinear terms and a finite difference method
- oldsolution = newsolution

until the numerical solution of the PDE is sufficiently accurate

- use the reference solution to update the NRMSD data for the sub-domain

end for

- send the NRMSD data for the sub-domain to the *master*

end if

- finalise the MPI

end *program*

Template for the Two-Dimensional, Linear Program

begin *program*

- constant, variable and array declarations (including the solution domains)
- initialise the MPI
- take the program wall time

if *master* then

- calculate the weights/terms required by the numerical inversion algorithm
- receive the ranks of the solution domain in Laplace space from the *slave*s
- calculate the numerical solution of the PDE
- read in the reference solution
- calculate the NRMSD
- calculate the program wall time
- display/store the NRMSD and the program wall time
- display/store the numerical solution of the PDE

else *slave*

- initialise the allocated rank of the solution domain in Laplace space
- calculate the solution in the allocated rank of the solution domain in Laplace space using a finite difference method
- send the allocated rank of the solution domain in Laplace space to the *master*

end if

- finalise the MPI

end *program*

Template for the Two-Dimensional, Nonlinear Program

begin *program*

- constant, variable and array declarations (including the solution domains)
- initialise the MPI
- take the program wall time
- initialise the nonlinear terms

if *master* then

- calculate the weights/terms required by the numerical inversion algorithm

repeat

- send the nonlinear terms to the *slave*s
- receive the ranks of the solution domain in Laplace space from the *slave*s
- calculate the numerical solution of the PDE
- update the nonlinear terms

until the numerical solution of the PDE is sufficiently accurate

- read in the reference solution
- calculate the NRMSD
- calculate the program wall time
- display/store the NRMSD and the program wall time
- display/store the numerical solution of the PDE

else *slave*

- receive the nonlinear terms from the *master*
- initialise the allocated rank of the solution domain in Laplace space
- calculate the solution in the allocated rank of the solution domain in Laplace space using the nonlinear terms and a finite difference method
- send the allocated rank of the solution domain in Laplace space to the *master*

end if

- finalise the MPI

end *program*

Sample Parallel Programs

Sample parallel programs will be put here in the version to be placed in the UHRA. These will be the one-dimensional, linear program, a one-dimensional, nonlinear program, the two-dimensional, linear program and a two-dimensional, nonlinear program. Kathy Lee from the UH Research Office will confirm to Professor Christianson and Dr Sayers that this has been done.