Towards efficient nonlinear option pricing

Shih-Hau Tan

A thesis submitted in partial fulfilment of the requirements of the University of Greenwich for the Degree of Doctor of Philosophy

April 2018

DECLARATION

I certify that the work contained in this thesis, or any part of it, has not been accepted in substance for any previous degree awarded to me, and is not concurrently being submitted for any degree other than that of Doctor of Philosophy being studied at the University of Greenwich. I also declare that this work is the result of my own investigations, except where otherwise identified by references and that the contents are not the outcome of any form of research misconduct.

.....
Shih-Hau Tan (PhD Student)
Date:

.....
Prof. Choi-Hong Lai (First Supervisor)
Date:

.....
Dr Konstantinos Skindilias (Second Supervisor)
Date:

ACKNOWLEDGEMENTS

I would like to thank my first supervisor, Prof. Choi-Hong Lai, for his guidance of the research and I am grateful for his enormous support during my PhD study. I would like to also thank my second supervisor, Dr Konstantinos Skindilias, for his assistance and advice.

I would like to thank Mr. Lung-Sheng Chien from NVIDIA for kindly giving me many practical suggestions on GPU computing. I would like to express my gratitude to Prof. Kevin Parrott, for the collaboration and discussion on numerical PDEs. I would like to also thank Dr André Ribeiro for his support on mathematical finance with his industrial experience. My sincere thanks to Carlos Vázquez Cendón, Ki Wai Chau, Daniel Duffy, Julien Hok, Karel in't Hout, José Antonio García Rodríguez, Daniel Ševčovič, Tai-Ho Wang, for all the fruitful discussions.

Thanks to the financial and research support from the ITN-STRIKE project and the University of Greenwich. Also thanks to all the helps from my colleagues. Special thanks to Dr Zuzana Bučková, Dr Vera Egorova, Dr Álvaro Leitao and Dr Radoslav Valkov for all the useful discussions.

Finally, I would like to thank my father, mother, aunt and brother, for their concern and encouragement.

ABSTRACT

This thesis describes the development of efficient numerical solvers for a wide range of nonlinear option pricing problems, including European options, Asian options and a multi-asset case. The chosen research methodology is the numerical PDE approach which essentially is to solve the nonlinear Black-Scholes equations with the relevant nonlinear volatility functions. The emphasis is on obtaining the numerical solution with reasonable accuracy and using high-performance computation.

The first approach applies the Newton-Raphson method to solve the nonlinear system resulting from the finite difference spatial discretisation. Several adjustments for improving accuracy are examined. These Newton-based solvers are combined with a semi-Lagrangian scheme to deal with the Asian option case, and one Alternative Direction Implicit (ADI) method for pricing options on multiple assets. An approach to solving large-scale problems with nonlinear volatilities with a GPU-based parallelisation framework is also proposed. Implementations on different software platforms are explained and compared. Case studies including large-scale Europe option pricing problems computed using single and multiple GPUs are discussed to demonstrate their performance compared with using sequential algorithms. These constructed solvers with parallel computing implementations essentially contribute to solving nonlinear problems in finance.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Pricing derivatives both accurately and efficiently is always challenging in financial markets. New nonlinear option pricing models developed in recent years offer an approach to obtain realistic and meaningful option prices in an incomplete market. This approach requires the solution of fully nonlinear PDEs numerically, and therefore robust and efficient solvers are highly desirable. Many approaches to nonlinear option pricing result in the formulation of a generalised Black-Scholes equation and an overview is given in this chapter. Different models with various considerations are introduced. Numerical solutions that have appeared recently are also reviewed. Several research questions are addressed which are answered in the chapters hereafter, and an outline of the thesis is given at the end of this chapter.

1.1 Nonlinear Option Pricing Models

An option is one of the most popular financial instruments which gives the owner a right to buy or sell the underlying asset with a certain strike price at or before the predetermined maturity time which depends on the type of option. It is often used to hedge the risk from uncertain movements of the underlying asset price. An essential issue is to compute the option price accurately which sometimes may be computational demanding when an analytical formula does not exist.

According to the classical theory due to Black, Scholes and Merton, an option in an idealised financial market can be priced as a solution $V = V(S,t)$ to the linear Black-Scholes equation [8]:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma_0^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0, \ (S,t) \in [0,\infty) \times [0,T) \tag{1.1}$$

where T is the maturity, $r > 0$ is the risk-free interest rate and $\sigma_0 > 0$ is a constant which measures the volatility of the underlying asset price process $\{S_t, t \geq 0\}$, which is assumed to follow the stochastic differential equation

$$dS_t = rS_t dt + \sigma_0 S_t dW_t, \tag{1.2}$$

where W_t is a standard Brownian motion. A terminal condition $V(S,t=T)$ is given by the payoff function $\Phi(S,K)$ which depends on the type of option contract where K is the strike price of the option. For example, for the European call option, the payoff function is defined as

$$V(S,T) = \Phi(S,K) = \max(S-K,0),$$

and for the European put option is

$$V(S,T) = \Phi(S,K) = \max(K-S,0).$$

The linear Black-Scholes equation (1.1) is derived under several restrictive assumptions, including frictionless, perfectly liquid markets with zero transaction costs and assets with constant volatility. Theses assumptions are not always true in a real market. The motivation behind nonlinear option pricing is to lessen these restrictive assumptions in order to achieve realistic option prices in incomplete markets. These nonlinear option pricing models usually introduce a non-constant volatility function which depends on the underlying asset S, the time t and its sensitivity V_{SS} where

$$V_{SS} = \frac{\partial^2 V}{\partial S^2}$$

is often called the option Gamma in financial markets. For these models the classical Black-Scholes equation (1.1) then becomes a generalised nonlinear Black-Scholes equation

$$\frac{\partial V}{\partial t}+\frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}+rS\frac{\partial V}{\partial S}-rV=0. \tag{1.3}$$

where $\sigma=\sigma(S,t,V_{SS})$.

Leland [80] proposed one of the first nonlinear option pricing models under constant transaction costs where the volatility function is given as

$$\sigma=\sigma_0(1+Le\cdot sign(V_{SS})), \tag{1.4}$$

where $Le=\sqrt{\frac{2}{\pi}}\frac{c}{\sigma_0\sqrt{\Delta t}}$. Here $c>0$ is a constant transaction cost, and Δt is the time interval between portfolio adjustments. A restriction of $Le<1$ must be added to guarantee a positive number of volatility in equation (1.4).

Another model for incoporating transaction costs given by Barles and Soner [7] uses an exponential utility function to characterize the investor's preferences leading to

$$\sigma=\sigma_0(1+\Psi(e^{r(T-t)}a^2S^2V_{SS}))^{1/2}, \tag{1.5}$$

where $a>0$ is a constant to measure the risk aversion, and Ψ is the solution of the following ordinary differential equation:

$$\Psi'(x)=\frac{\Psi(x)+1}{2\sqrt{x\Psi(x)}-x},\ \Psi(0)=0. \tag{1.6}$$

A further class of nonlinear models arise from an uncertain volatility and was studied by Avellaneda *et al* [3, 4] and Lyon [82] where it is assumed that the volatility is bounded by two extremes values $\sigma_{\min}$ and $\sigma_{\max}$. The volatility function was again shown to be determined by

the sign of V_{SS}, namely

$$\sigma = (\sigma_{\max}^2 \mathbf{1}_{\mathbf{V_{SS}>0}} + \sigma_{\min}^2 \mathbf{1}_{\mathbf{V_{SS}<0}})^{1/2}, \tag{1.7}$$

where the function $\mathbf{1}$ represents the indicator function.

The so-called Risk Adjusted Pricing Methodology (RAPM) was proposed by Kratka [73] and further generalised by Jandačka and Ševčovič [67]. The aim is to optimise the time interval between consecutive portfolio adjustments to minimise the sum of the rate of transaction costs and the rate of a risk from an unprotected portfolio, where the volatility function is then

$$\sigma = \sigma_0 \left(1 + \mu (SV_{SS})^{\frac{1}{3}}\right)^{1/2} \tag{1.8}$$

where $\mu = 3\left(\frac{C^2 R}{2\pi}\right)^{1/3}$ and $C, R \geq 0$ are measures of transaction costs and risk premium respectively.

The final example is determined by a model of market illiquidity in which the dynamic of underlying asset is driven by feedback and market effects and was first investigated by Wilmott [113]. A classical example introduced by Frey [42], Frey and Stremme [42], Frey and Patie [43], Frey and Polte [44] considers the effect of the stock trading strategy of a large investor in the illiquid market, and the volatility was derived to be

$$\sigma = \sigma_0 (1 - \rho \lambda(S) S V_{SS})^{-1}, \tag{1.9}$$

where $\rho > 0$ measures the liquidity, and $\lambda(S)$ is a function describing the liquidity profile of the market. The term $(1 - \rho \lambda(S) S V_{SS})$ must be greater than zero to ensure a positive value of the volatility.

Table 1.1 Summary List of nonlinear volatilities.

Model	σ
Leland model	$\sigma_0(1+Le\cdot sign(V_{SS}))$
Barles Soner model	$\sigma_0(1+\Psi(e^{r(T-t)}a^2S^2V_{SS})^{1/2}$
Frey-Patie model	$\sigma_0(1-\rho\lambda(S)SV_{SS})^{-1}$
RAPM model	$\sigma_0((1+\mu(SV_{SS})^{\frac{1}{3}}))^{1/2}$
Uncertain volatility model	$(\sigma_{\max}^2\mathbf{1}_{\mathbf{V_{SS}>0}}+\sigma_{\min}^2\mathbf{1}_{\mathbf{V_{SS}<0}})^{1/2}$

1.2 Viscosity Solutions

The nonlinear option pricing problem can be generalised as the solution of a fully nonlinear PDE, equation (1.3), in which the volatility $\sigma=\sigma(S,t,V_{SS})$ depends on the second derivative of the solution of the equation. However, if the given initial and boundary conditions are not smooth enough, then the classical solution which must be $C^{2,1}([0,\infty)\times[0,T))$, may not exist. One usually considers the viscosity solution for which existence and uniqueness can be established [20, 21] under the Lions framework.

Consider a more general form of the fully nonlinear PDE

$$u_t+\mathscr{F}(x,t,u,Du,D^2u)=0,\ x\in\Omega,\ t\in[0,T), \tag{1.10}$$

where Du, D^2u means the gradient and Hessian of the solution $u(x,t)$ respectively, and $\Omega\subset\mathbb{R}$ is an open domain.

Definition 1. *Viscosity Solution*

Let u be locally bounded and assume that $\mathscr{F}$ satisfies the ellipticity condition, then (i) u is called a viscosity supersolution of equation (1.10) if for all $\phi\in C^{2,1}([0,\infty)\times[0,T))$,

$$\phi_t(x,t)+\mathscr{F}(x,t,u,D\phi,D^2\phi)\geq u_t+\mathscr{F}(x,t,u,D\phi,D^2\phi)\geq 0.$$

(ii) u is called a viscosity subsolution of equation (1.10) if for all $\phi \in C^{2,1}([0,\infty)\times[0,T))$,

$$\phi_t(x,t)+\mathscr{F}(x,t,u,D\phi,D^2\phi)\leq u_t+\mathscr{F}(x,t,u,D\phi,D^2\phi)\leq 0.$$

(iii) u is called a viscosity solution if it is both a viscosity supersolution and a viscosity subsolution.

Viscosity solutions are more suitable and meaningful choices for solving these nonlinear PDEs in finance as addressed in [5, 6, 41], because the solution can be proved existent and obtained uniquely, while in the classical solution sense one of them may be absent. The proof of existence and uniqueness of the viscosity solution for equation (1.10) is usually based on the comparison principle for which more details can be found in [40]. In this thesis the viscosity solution of the nonlinear Black-Scholes PDE is assumed to exist and be unique. The numerical convergence to the viscosity solution is discussed in Chapter 2.

1.3 Numerical Solutions

The example of the nonlinear volatility considered in this thesis has the form

$$\sigma=\sigma(S,t,V_{SS}), \tag{1.11}$$

and usually the exact solution of the fully nonlinear PDE (1.3) cannot be obtained. However, construction of explicit solutions with the nonlinear volatility function (1.9) was recently provided by Bordag [9, 10]. These invariant solutions were constructed by means of the invariant Lie group theory and depend on various parameters restricting the class of solutions, and in general there is no exact pricing formula for the case of a call or put payoff function and therefore the solution cannot be used in the real markets for doing option pricing. Approximate or numerical solutions are needed and different approaches are summarised in [34, 54]. A brief overview is given as follows.

1.3.1 Approximate Formula

A small parameter perturbation method discussed by Ďuriš [29] proposes an approximate formula for volatility functions which can be expressed as the following form:

$$\sigma(S,T-t,V_{SS})^2 = \sigma_0^2 + 2\varepsilon A(T-t)S^{\gamma-1}H^{\delta-1}, \text{ where } H = SV_{SS}. \tag{1.12}$$

The powers γ,δ, the parameter ε as well as the function $A(T-t)$ depend on the chosen nonlinear volatility model. For example, in the case of the Frey-Patie model in equation (1.9) for the case $\lambda(S)=1$, it can be approximated as

$$\sigma(S,t,V_{SS}) = \sigma_0(1-\rho SV_{SS})^{-1} \approx \sigma_0(1+\rho SV_{SS}),$$

so that the parameters in equation (1.12) are

$$\varepsilon = \rho, \gamma = 1, \delta = 2, A(T-t) = \sigma_0^2/2,$$

and the model parameter ε is small provided $0 < \varepsilon \ll 1$.

The idea is to seek the option price as an asymptotic expansion in terms of the small parameter. More precisely,

$$V = V_0 + \sum_{i=1}^{N} \varepsilon^i V_i + O(\varepsilon^{N+1}), \tag{1.13}$$

where the leading term V_0 is simply a solution to the linear Black-Scholes model. When considering $N=1$, an approximate formula can be derived as shown in [29] of the form

$$V(S,t) \approx V_0(S,t) + \varepsilon V_1(S,t), \tag{1.14}$$

where

$$V_1(S,t) = \frac{K^\gamma}{(2\pi\sigma_0^2)^{\frac{\delta}{2}}}\left(\frac{S}{K}\right)^{\frac{\gamma-\delta}{1-\delta}} e^{\left\{\beta+\frac{[\gamma-\delta-\alpha(1-\delta)]^2\sigma_0^2}{2(1-\delta)^2}\right\}(T-t)} \times \int_0^{T-t} \frac{A(\xi)}{\xi^{\frac{\delta-1}{2}}\sqrt{\delta(T-t)+(1-\delta)\xi}} e^{E\xi - M(S)\frac{1}{\delta(T-t)+(1-\delta)\xi}} d\xi, \tag{1.15}$$

E is a constant given by

$$E = \frac{[\gamma-\delta-\alpha(1-\delta)]^2\sigma_0^2}{2(\delta-1)} + \beta(\delta-1)$$

and

$$M(S) = \frac{\delta}{2\sigma_0^2}\left(\log\left(\frac{S}{K}\right)\right)^2 + \frac{[\gamma-\delta-\alpha(1-\delta)]\,\delta(T-t)}{1-\delta}\log\left(\frac{S}{K}\right) + \frac{[\gamma-\delta-\alpha(1-\delta)]^2\sigma_0^2\delta(T-t)^2}{2(1-\delta)^2}.$$

The analytic approximation of the option price $V(S,t)$ can then be evaluated via equation (1.14) using numerical integration. Note that the approximate formula in equation (1.14) in fact works well only when the parameter ε is small. When ε is getting larger, the approximation may be inaccurate which was examined with different examples in [29].

1.3.2 Transformation Method

A numerical method proposed and investigated by Jandačka and Ševčovič [67] is based on the transformation $H = SV_{SS}, x = \log(S/K), \tau = T-t$, which transforms equation (1.3) with $\sigma = \sigma(S,t,V_{SS})$ into a porous media type of quasilinear parabolic equation:

$$\frac{\partial H}{\partial \tau} = \frac{\partial^2}{\partial x^2}\beta(H) + \frac{\partial}{\partial x}\beta(H) + r\frac{\partial H}{\partial x}, \tag{1.16}$$

where $\beta(H) = \frac{1}{2}\sigma^2(H)H$ is an increasing function. For instance, in the case of the volatility function given by (1.9) for $\lambda(S) = 1$, one obtains $\beta(H) = \frac{\sigma_0^2}{2}H(1-\rho H)^{-2}$. In a recent paper [106], Ševčovič and Žitňanská investigated the nonlinear equation (1.16) in the context of

modeling variable transaction costs. The existence of classical Hölder smooth solutions was proved and useful bounds for the solution were derived. The transformation technique developed allows for the construction of a semi-implicit finite volume based numerical scheme for solving (1.16) which more details can be checked in [67, 105].

1.3.3 Backward Stochastic Differential Equation

There exists a connection between the fully nonlinear PDE and the corresponding Backward Stochastic Differential Equation (BSDE) as stated in [37, 93] which provides a probabilistic approach for solving equation (1.3). The BSDE can be first order or second order depending on the chosen nonlinear volatility model as described in [54]. The BSDE approach allows the use of Monte Carlo simulation to solve a wide range of class of nonlinear PDE and has been investigated in recent years. For examples, Bouchard and Touzi introduced a simulation based on the discrete-time approximation [11] and applied it to the solution of fully nonlinear parabolic PDEs in [38]. Gobet proposed a regression-based Monte Carlo method for solving the BSDE in [50] and Alanko [1] gave a higher-order scheme based on this technique and remedied the drawback of small time steps. More details of employing this approach for solving fully nonlinear PDEs can be found in [49, 54].

The BSDE approach has the intrinsic potential of being easily implemented on parallel computing platforms. Labart [77] showed an implementation of using a cluster with multiple CPUs to simulate the BSDE, and with an application of pricing American option in [76]. Gobet also provided a framework of using GPUs for solving the BSDE with the stratified regression Monte-Carlo scheme in [51]. The results of applying parallel computing all show very good performance as the dimension grows.

1.4 Research Objectives and Thesis Outline

In addition to the approximate formula or probabilistic approach, the finite difference method provides a straightforward numerical approach to solve the nonlinear Black-Scholes equation described in equation (1.3). Solving linear PDEs with the finite difference method has been studied for decades and many classical schemes and their numerical analysis have been applied to examples from finance (see [28, 58]). A variety of improvements or updated schemes are emerging in recent years concentrating on nonlinear problems. There are some difficulties when using the finite difference method for solving nonlinear PDEs. For example, the nonlinearity may affect the expected accuracy from the discretisations which sometimes causes the expected accuracy from the chosen scheme to be only be observed in one-dimensional problems. Also the increasing complexity of the problem means that building the nonlinear solver is more time-consuming compared to solving linear cases.

In this thesis, robust and efficient finite difference solvers for handling the nonlinear Black-Scholes equation (1.3) are proposed with the implementation of parallel computing and extension to high-dimensional problems. The focus for the nonlinearity is on the Frey-Patie model of equation (1.9) which essentially gives a mechanism for understanding the market impact on the option price in illiquid markets, but the methodology developed in this thesis is by no means of restricted to this model. The structure of this thesis is depicted in Fig. 1.1 which illustrates all the chapters with the associated topics. To be more precise, three research objectives with achieved results are summarised below.

1.4.1 Newton-based Solvers for Nonlinear Black-Scholes Equations

Unlike solving the linear PDE, which keeps a constant matrix constructed from the finite difference spatial discretisation in a time-marching scheme, when handling the nonlinear PDE, iterative schemes are required and the coefficients of the matrix have to be evaluated at each time step. Therefore a core issue for making the finite difference solver robust is

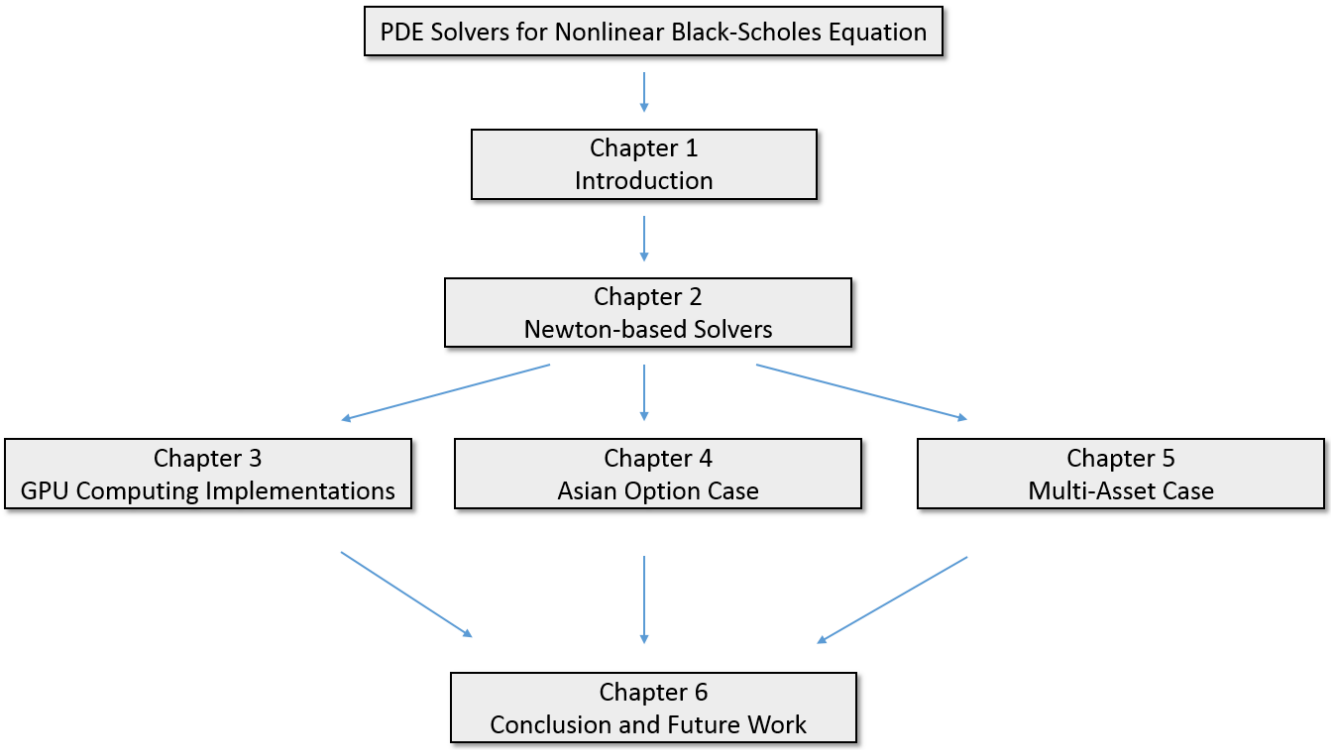

Fig. 1.1 The structure of the thesis.

to reduce the necessary number of iterations and guarantee that the iterative scheme works successfully. Several research questions are discussed as follows.

- How to construct nonlinear solvers to make the iterative scheme work robustly and efficiently?

- Whether the expected accuracy of the chosen scheme can be obtained and how can it be improved?

- How well do the constructed solvers work compared to other solvers?

In Chapter 2, detailed descriptions of constructing Newton-based solvers for nonlinear Black-Scholes equation are given. They are based upon using Newton-Raphson method for solving the nonlinear system generated from the finite difference spatial discretisaion. The benefit of using Newton-Raphson method is a rapidly convergent result for the iterative scheme, and the cost is the extra computations to obtain the Jacobian matrix. Different numerical experiments were designed and carried out to demonstrate that the Newton-based solvers can behave more robustly than elementary fixed point iteration. Also, comparisons with some other Newton-like methods show that the Newton-based solvers have better efficiency due to the simple tri-diagonal structure of the matrix.

Furthermore, possible adjustments for improving the accuracy are also proposed. It is commonly observed that some theoretical values from the convergent order of error are not easily achieved or are only obtained with very fine meshes. Some numerical oscillation problems may also destroy the higher order accuracy or lead to instability of the solver. In Chapter 2, a coordinate stretching technique introduced in [94] is applied which provides the adjustment of the grid points close to $S = K$, and the usual convergent behaviour of

the error can be observed without using a large number of grid points. The Rannacher timestepping method from [100] is also examined which remedies the oscillation problem and this approach combined with the coordinate stretching, eventually can help to recover the higher order accuracy. In addition, the technique of using Richardson extrapolation to cancel the solution error is described which results in improved answers by extrapolating using the answers from coarse meshes. These numerical adjustments indeed contribute to resolving the accuracy issue from the nonlinear problem supporting the applicability and usefulness of the constructed Newton-based solvers.

1.4.2 GPU Computing Implementations

In the financial industry, good quality and productive systems are always in high demand to deal with large-scale problems from the real market. The use of the nonlinear models addressed in Sec 1.1 potentially allows traders to obtain more realistic and accurate option prices. However one main drawback comes from the computational cost issue as calculating the price now takes much more time than using the standard linear model or approximate solutions. Parallel computing shows the possibility of bridging the gap between the accurate price and the efficient proxy. Questions linked to this subject are discussed as follows.

- What is the best framework of doing parallel computing on the constructed nonlinear PDE solvers for large-scale problems?

- What are the possible implementations using high-level libraries or software?

- What is the achieved performance compared with using sequential algorithm?

- How well does this approach work for nonlinear PDE solvers in terms of the performance and programming time?

To answer these questions, in Chapter 3 a framework for solving large-scale nonlinear PDEs with GPUs is proposed. The substantive tasks are decomposing the PDE solver into different steps of numerical linear algebra computations which are activated by batch operation. Two different implementations are examined for the constructed nonlinear batched solver for the given independent nonlinear PDEs. First the commercial software OpenACC which allows the user to do GPU computing easily without writing too much code. Second, fine tuning of the CUDA programming is explored with deeper control and manipulation of the threading and memory allocation to fit the specified problems. Furthermore, the idea of using multiple GPUs for the batched solver is proposed and implemented. The results achieved from both implementations show good speedups compared to the sequential computation and verify the effectiveness of the batch operation.

1.4.3 High-dimensional Problems

Most contemporary research on nonlinear option pricing focuses on the one-dimensional problem due to the difficulty of building the numerical solvers. There is very little literature on high-dimensional problems. Therefore the motivation for Chapter 4 and Chapter 5 is solving nonlinear models in high-dimensional cases by proposing efficient numerical PDE solvers. The questions discussed in these two chapters are as follows.

- What is the structure of nonlinear option pricing problems and the resulted nonlinear PDEs in high-dimensional cases?

- What are the methods to reduce the problems to lower dimension cases?

- How to combine the constructed Newton-based solvers with these methods to solve high-dimensional nonlinear PDEs?

In Chapter 4, the extension of the Frey-Patie model for Asian options is introduced to handle the market impact for the price of Asian option in illiquid markets. The generated mathematical problem becomes a two-dimensional nonlinear PDE and the focus is on constructing the numerical solvers. Using a semi-Lagrangian scheme reduces the complexity of the problem to lower dimensional cases, and the Newton-based solvers developed in Chapter 2 are immediately applicable. The results show the merged scheme is able to compute the Asian option pricing with nonlinear volatility effectively.

Chapter 5 discusses the multi-asset problem which is another kind of high-dimensional case considered in this thesis. A derivation of a generalised Frey-Patie model with multiple assets is given in the beginning of the chapter, and then a constructed ADI-Newton solver is explained which combines the advantages of using the ADI method to solve high-dimensional PDE and the developed robust Newton-based solver for dealing with the nonlinearity. The implementation of this scheme is explained with details and the convergence properties on several two-dimensional test problems are presented in the thesis.

1.5 Summary

Chapter 1 draws the contour of the whole thesis, and several models and existing numerical techniques are reviewed. Three main research objectives are addressed and answered in the following four chapters which contribute to different areas, such as numerical solvers, parallel computing in nonlinear option pricing. The next chapter starts with the fundamental finite difference method to construct the Newton-based solver for the nonlinear Black-Scholes equation.

Chapter 2

Newton-based Solvers

In chapter 2, implicit finite difference schemes for solving the fully nonlinear PDE stated in equation (1.3) are examined. The focus is on solving the corresponding nonlinear system resulting from the finite difference spatial discretisation, and different Newton linearisation techniques are applied and compared. Numerical results on the convergence of the numerical scheme are shown and explained, and possible remedies for achieving higher order accuracy are examined.

2.1 Introduction

The essential mathematical problem from nonlinear option pricing is to solve the fully nonlinear PDE

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0, \tag{2.1}$$

with typical nonlinear volatility functions $\sigma = \sigma(S,t,V_{SS})$. The finite difference method is a straightforward way to solve the PDE numerically and many schemes have appeared for handling different nonlinear volatility models in recent years.

Explicit schemes have been widely used due to a simple computation which only requires a linear combination of known values to update the solution at the next time step. A consistent monotone explicit finite difference scheme has been analysed by Company [17] for solving

the Frey-Patie model. Bučková [13] introduced the so-called Alternative Direction Explicit (ADE) scheme for dealing with different nonlinear models and this method is based on an explicit scheme but with better accuracy on temporal discretisation. These explicit schemes have to guarantee stability and consistency requiring some restrictions on the ratio of the sizes of the temporal and spatial discretisations and sometimes results in excessive computation. It is possible to combine other techniques for constructing unconditionally stable explicit schemes as shown in [36, 53], but with other limitations on parameters to ensure the positivity.

Implicit schemes have less restrictions on the step sizes but require paying the cost of solving a linear system at each time step. Pooley [99] studied the fully implicit and Crank-Nicolson schemes of solving the uncertain volatility model. Heider [57] examined these schemes for additional nonlinear volatility models. These implicit schemes generate a corresponding nonlinear system to be solved at each time step and Newton-Raphson method was applied in both papers [57, 99]. Ďuriš *et al.* [29] compared different linearisation techniques based on Newton-Raphson method to solve equation (2.1). Egorova [33] compared using explicit and implicit schemes to solve the transaction cost model proposed by Barles and Soner in equation (1.5) for the American option case. These approaches demonstrate some advantages of solving the nonlinear PDE by merging implicit schemes with Newton-Raphson method.

In the following sections, a detailed explanation of using implicit schemes to solve equation (2.1) for European option case is given. The selected nonlinear model in this thesis is the model introduced by Frey [43] which addresses the importance of the liquidity impacts. An overview and derivation of the model is given first, followed by a discussion of a finite difference discretisation to solve such nonlinear PDE. The discussions on numerical convergence to viscosity solution follow the same ideas from [57, 99], and different methods of solving the derived nonlinear system are compared. Moreover, variant improvements in terms of accuracy are examined such as using coordinate stretching and Rannacher timestepping,

which higher order of accuracy can be recovered. Comparisons of the complexity from different kinds of Newton-based solvers are shown at the end of this chapter.

2.2 Mathematical Modelling

Departing from the liquid market assumption in the Black-Scholes model, it is known that people can not trade as they expect. Therefore the hedging strategies from the traders could influenc prices as they may not be able to obtain sufficient assets to eliminate the risk. The result is the dynamics of asset price may be driven by the strategies from the investors to hold or sell the shares. Typical examples can be discovered in an illiquid market when there exists a large trader whose trades have a large impact, and in such cases it is necessary to include this strategy into the dynamics of the underlying asset price. In the following the ideas given by [42, 43, 105, 113] are followed in order to derive the Frey-Patie model and corresponding PDE of the option price.

Assuming that the price of an underlying asset S_t is influenced by some holding and hedging strategies from a large trader, or different small traders who apply the same strategies, in an illiquid market, and it satisfies the process

$$dS_t = \mu_0 S_t dt + \sigma_0 S_t dW_t + \rho S_t d\alpha_t, \tag{2.2}$$

where W_t is a standard Brownian motion, μ_0 is the drift term, σ_0 is the historical volatility, ρ is the liquidity of the market, and α_t is the strategy function which shows how many shares the investor should hold at a given time t. More precisely, let $\alpha_t = \phi(S_t, t)$, then by Itô's formula

$$d\alpha_t = \left(\frac{\partial \phi}{\partial t} + \frac{\sigma_0^2}{2}\frac{\partial^2 \phi}{\partial S^2}\right) dt + \frac{\partial \phi}{\partial S}, \tag{2.3}$$

and equation (2.2) can be replaced as

$$\left(1 - \rho S_t \frac{\partial \phi}{\partial S}\right) dS_t = \mu_0 S_t dt + \sigma_0 S_t dW_t + \rho S_t \left(\frac{\partial \phi}{\partial t} + \frac{\sigma_0^2}{2} S_t^2 \frac{\partial^2 \phi}{\partial S^2}\right) dt, \tag{2.4}$$

which can be simplified as

$$dS_t = \mu(S_t,t)S_t dt + \sigma(S_t,t)S_t dW_t, \tag{2.5}$$

with

$$\mu(S,t) = \frac{1}{1-\rho S\frac{\partial\phi}{\partial S}}\left(\mu_0 + \rho\left(\frac{\partial\phi}{\partial t} + \frac{\sigma_0^2}{2}S^2\frac{\partial^2\phi}{\partial S^2}\right)\right),$$

and

$$\sigma(S,t) = \frac{\sigma_0}{1-\rho S\frac{\partial\phi}{\partial S}}.$$

The strategy function $\phi(S_t,t)$ can be chosen to be

$$\phi(S_t,t) = \frac{\partial V}{\partial S}$$

as shown in [43]. By taking $\tilde{W}_t = W_t + \frac{\mu-r}{\sigma}t$ to change the equation to risk-neutral measure, and using Feymann-Kac theorem as stated in [107], the nonlinear Black-Scholes equation under this Frey-Patie model is

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2(S,t)S^2\frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \tag{2.6}$$

with

$$\sigma(S,t) = \frac{\sigma_0}{1-\rho S\frac{\partial^2 V}{\partial S^2}} = \sigma_0(1-\rho S V_{SS})^{-1}, \tag{2.7}$$

which allows the pricing of European option in an illiquid market.

Note that from the discussion in [43], the liquidity is more gererally controlled by a liquidity profile $\lambda(S)$ which gives

$$\sigma = \sigma_0(1-\rho\lambda(S)SV_{SS})^{-1}. \tag{2.8}$$

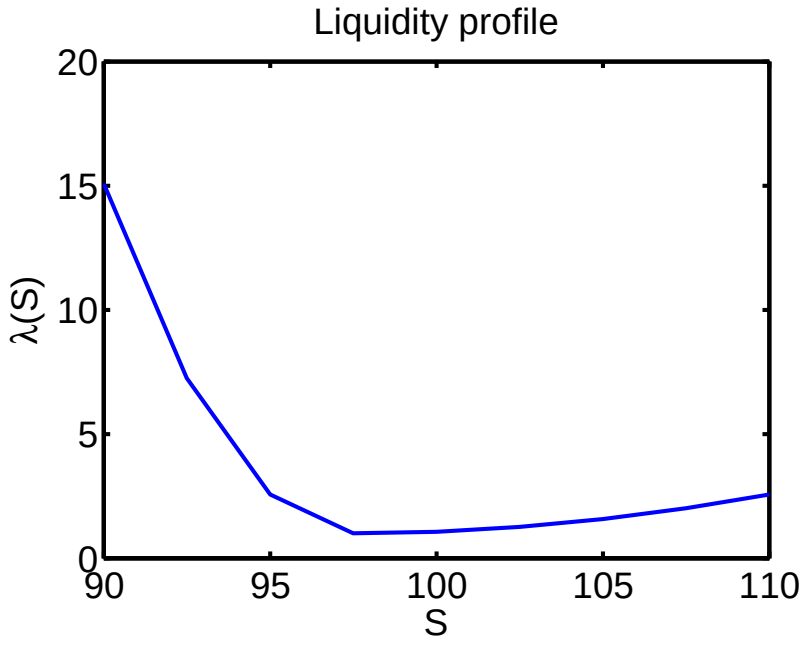

Fig. 2.1 The liquidity profile function from equation (2.9).

The liquidity profile can actually be parametrised specificity for fitting the volatility smile or skew. For example, taking the profile function as used in [43]

$$\lambda(S) = 1 + (S - S_0)^2 \left(a_1 \mathbf{1}_{S \leq S_0} + a_2 \mathbf{1}_{S > S_0}\right), \tag{2.9}$$

where $\mathbf{1}$ represents the indicator function. If the parameters are fixed to be $T = 0.25, \sigma_0 = 0.2, S_0 = 100, a_1 = 0.5, a_2 = 0.01, \rho = 0.01$, then Fig. 2.1 shows the liquidity profile and Fig. 2.2 presents the volatility from Frey-Patie model which can perform the behaviour of a volatility skew.

Fig. 2.3 also gives an example of calculating the option pricing with different liquidities ρ where the liquidity profile function is considered as $\lambda(S) \equiv 1$, and it can be observed that the price is getting larger when ρ increases.

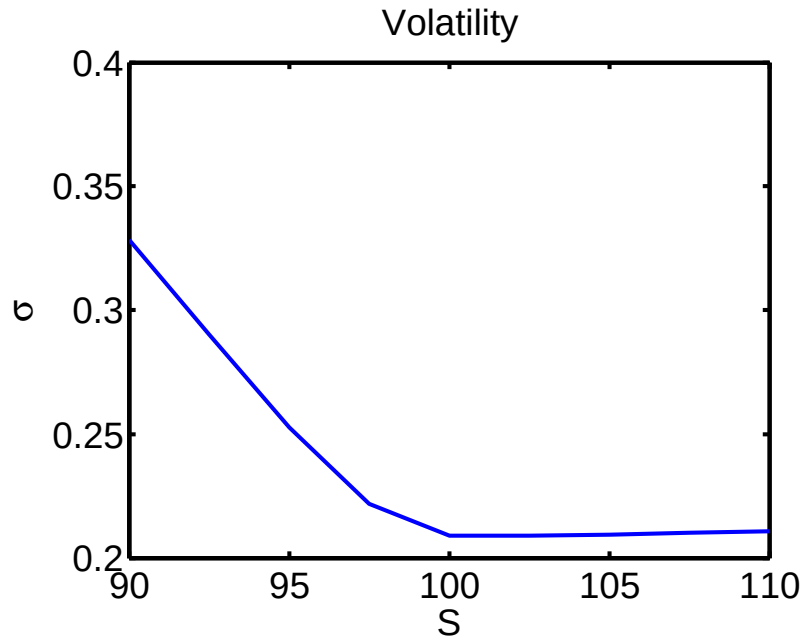

Fig. 2.2 Volatility obtained from equation (2.8) with the profile function shown in equation (2.9).

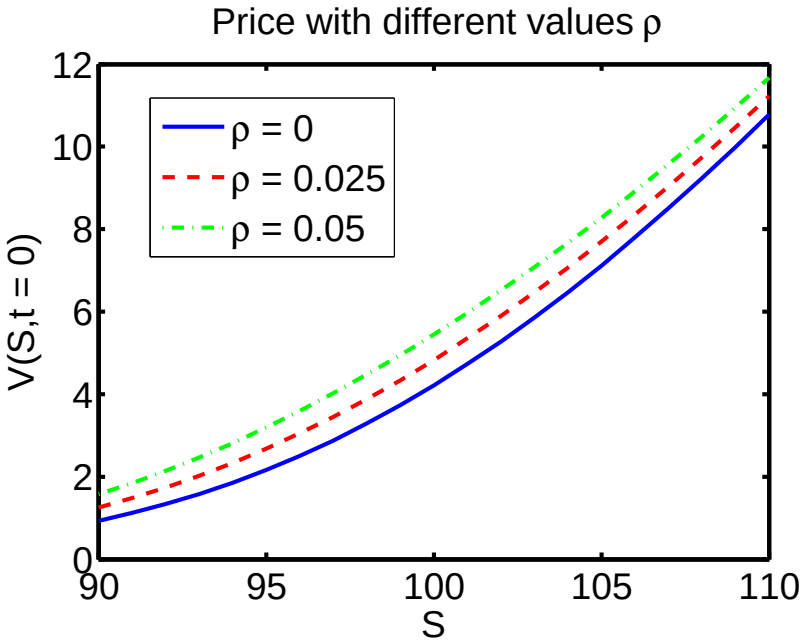

Fig. 2.3 Option prices with the volatility in equation (2.7) and different liquidities ρ.

2.3 Finite Difference Discretisation

Using standard finite difference notation, an implicit finite difference scheme based on θ-method replaces equation (2.6) reads as follows:

$$\frac{V_i^{n-1} - V_i^n}{\Delta t} = \theta \mathscr{L}(V_i^{n-1}) + (1-\theta)\mathscr{L}(V_i^n), \tag{2.10}$$

where

$$\mathscr{L}(V_i^n) = \frac{1}{2}(\sigma_i^n)^2 S_i^2 \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n}{(\Delta S)^2} + rS_i \frac{V_{i+1}^n - V_{i-1}^n}{2\Delta S} - rV_i^n.$$

Here the computational domain is truncated as $V(S,t) \in [0, S_{max}] \times [0,T)$, and $S_i = i\Delta S$, $i = 0, \cdots, N_S - 1$ also $n = 0, \cdots, N_t - 1$, where N_S and N_t are the numbers of grid points for spatial and temporal discretisations respectively and $\Delta S = S_{max}/(N_S - 1), \Delta t = T/(N_t - 1)$. V_i^n is the discretised replacement of $V(i\Delta S, n\Delta t)$. Note the calculation here is backward from $n = N_t - 1$ to $n = 0$. The volatility function σ_i^n is evaluated as

$$\sigma_i^n = \sigma_0 \left(1 - \rho S_i \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n}{(\Delta S)^2}\right)^{-1}.$$

When choosing $\theta = 1$, equation (2.10) becomes the fully implicit (or Backward Euler) scheme and may be written as the nonlinear system

$$H(V^{n-1})V^{n-1} = V^n, \tag{2.11}$$

where H is a $(N_S - 2) \times (N_S - 2)$ tri-diagonal matrix defined as $[a,b,c]$ and a,b,c are $(N_S - 2) \times 1$ column vectors representing the lower, main and upper diagonal entries of H which are

$$a_i = \frac{\Delta t}{2}\left(\frac{-rS_i}{\Delta S} + \frac{\left(\sigma_i^{n-1}\right)^2 S_i^2}{(\Delta S)^2}\right), \; b_i = 1 - \Delta t\left(\frac{\left(\sigma_i^{n-1}\right)^2 S_i^2}{(\Delta S)^2} + r\right), \; c_i = \frac{\Delta t}{2}\left(\frac{rS_i}{\Delta S} + \frac{\left(\sigma_i^{n-1}\right)^2 S_i^2}{(\Delta S)^2}\right),$$

where $i = 1, \cdots N_S - 2$.

Similarly when choosing $\theta = 0.5$, equation (2.10) is called the Crank-Nicolson scheme and the problem can be stated as the nonlinear system

$$H_1(V^{n-1})V^{n-1} = H_2(V^n)V^n, \tag{2.12}$$

where H_1 and H_2 are again $(N_S-2)\times(N_S-2)$ tri-diagonal matrices defined as $[\tilde{a},\tilde{b},\tilde{c}]$ and $[\tilde{d},\tilde{e},\tilde{f}]$, here again $\tilde{a},\tilde{b},\tilde{c},\tilde{d},\tilde{e},\tilde{f}$ are $(N_S-2)\times 1$ column vectors representing the lower, main and upper diagonal entries of H_1 and H_2 which are

$$\tilde{a}_i = \frac{\Delta t}{4}\left(\frac{-rS_i}{\Delta S} + \frac{\left(\sigma_i^{n-1}\right)^2 S_i^2}{(\Delta S)^2}\right),\ \tilde{b}_i = 1 - \frac{\Delta t}{2}\left(\frac{\left(\sigma_i^{n-1}\right)^2 S_i^2}{(\Delta S)^2} + r\right),\ \tilde{c}_i = \frac{\Delta t}{4}\left(\frac{rS_i}{\Delta S} + \frac{\left(\sigma_i^{n-1}\right)^2 S_i^2}{(\Delta S)^2}\right),$$

$$\tilde{d}_i = \frac{\Delta t}{4}\left(\frac{rS_i}{\Delta S} - \frac{\left(\sigma_i^{n}\right)^2 S_i^2}{(\Delta S)^2}\right),\ \tilde{r}_i = 1 + \frac{\Delta t}{2}\left(\frac{\left(\sigma_i^{n}\right)^2 S_i^2}{(\Delta S)^2} + r\right),\ \tilde{f}_i = \frac{\Delta t}{4}\left(\frac{-rS_i}{\Delta S} - \frac{\left(\sigma_i^{n}\right)^2 S_i^2}{(\Delta S)^2}\right),$$

where $i = 1,\cdots N_S-2$.

2.3.1 Numerical Convergence to the Viscosity Solution

As stated in Section 1.2, when discussing the solution of nonlinear equations from finance, the viscosity solution is the best choice. Barles [5, 6] proved that for any monotone, consistent and stable finite difference scheme converges to the unique viscosity solution. This provides clear assertion when one wants to show that the numerical scheme converges to the unique viscosity solution. The results of proving consistency is standard and can be found from [108]. Pooley [99] demonstrated the fully implicit and Crank-Nicolson schemes converge under certain conditions for the uncertain volatility model, and Heider [57] showed similar results for different nonlinear volatility models under a transformed coordinate. Following this approach the convergence to the viscosity solution for Frey-Patie model shown in equation (2.7) is as follows.

Let $\Gamma_i^n = \Gamma_i^n(V_{i-1}^n, V_i^n, V_{i+1}^n)$ be the evaluation of V_{SS} with $\{V_{i-1}^n, V_i^n, V_{i+1}^n\}$ by finite differences, namely

$$\Gamma_i^n(V_{i-1}^n, V_i^n, V_{i+1}^n) = \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n}{(\Delta S)^2}, \tag{2.13}$$

and $\sigma_i^n = \sigma_i^n(S_i, t^n, \Gamma_i^n(V_{i-1}^n, V_i^n, V_{i+1}^n))$ which in the Frey-Patie model is

$$\sigma_i^n = \sigma_0 \left(1 - \rho S_i \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n}{(\Delta S)^2}\right)^{-1} = \sigma_0 \left(1 - \rho S_i \Gamma_i^n\right)^{-1}, \tag{2.14}$$

then

$$\begin{aligned} \Gamma_i^n(V_{i-1}^n + \varepsilon, V_i^n, V_{i+1}^n) &= \Gamma_i^n(V_{i-1}^n, V_i^n, V_{i+1}^n + \varepsilon) = \Gamma_i^n + \frac{\varepsilon}{(\Delta S)^2}, \\ \Gamma_i^n(V_{i-1}^n, V_i^n + \varepsilon, V_{i+1}^n) &= \Gamma_i^n - \frac{2\varepsilon}{(\Delta S)^2}, \end{aligned} \tag{2.15}$$

and the relations below follow from equations (2.14) and (2.15):

$$\begin{aligned} &\sigma_i^n(S_i, t^n, \Gamma_i^n(V_{i-1}^n + \varepsilon, V_i^n, V_{i+1}^n)) = \sigma_i^n(S_i, t^n, \Gamma_i^n(V_{i-1}^n, V_i^n, V_{i+1}^n + \varepsilon)) \geq \sigma_i^n, \\ &\sigma_i^n(S_i, t^n, \Gamma_i^n(V_{i-1}^n, V_i^n + \varepsilon, V_{i+1}^n)) \leq \sigma_i^n, \end{aligned} \tag{2.16}$$

for all $\varepsilon > 0$. Note that the term $(1 - \rho S V_{SS})$ is assumed to be positive. With these relations it is sufficient to prove the following lemmas.

Lemma 1. *(**Monotonicity of the Fully Implicit Scheme**) When $\theta = 1$ in equation (2.10), the fully implicit scheme is monotone if $\frac{\sigma^2 S}{\Delta S} - r \geq 0$.*

Proof. Let $f_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n) = 0$ represent the discrete equation (2.10) for $\theta = 1$; then it is necessary to show that $\forall \varepsilon > 0$

(a) $f_i(V_{i-1}^{n-1} + \varepsilon, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n) \geq f_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n)$

(b) $f_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1} + \varepsilon, V_i^n) \geq f_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n)$

(c) $f_i(V_{i-1}^{n-1}, V_i^{n-1} + \varepsilon, V_{i+1}^{n-1}, V_i^n) \leq f_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n)$

(d) $f_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n + \varepsilon) \geq f_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n)$

The inequalities (b) and (c) hold immediately from equations (2.15) and (2.16). For the inequality (a),

$$\begin{aligned} f_i(V_{i-1}^{n-1}+\varepsilon, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n) \geq & f_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n) + \frac{\varepsilon S_i \left(\frac{(\sigma_i^{n-1})^2 S_i}{\Delta S} - r \right)}{2\Delta S} \\ \geq & f_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n), \end{aligned} \tag{2.17}$$

where the inequality holds from the initial assumption.

The inequality (d) holds also immediately since

$$\begin{aligned} f_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n+\varepsilon) = & f_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n) + \frac{\varepsilon}{\Delta t} \\ \geq & f_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_i^n), \end{aligned} \tag{2.18}$$

which concludes the proof. □

Lemma 2. *(**Stability of the Fully Implicit Scheme**) When $\theta = 1$ in equation (2.10), the fully implicit scheme is unconditionally stable.*

Proof. The proof of stability can be found in [99]. □

Theorem 1. *The fully implicit scheme converges to the viscosity solution of equation (1.3) when $\Delta S \to 0$ and $\Delta t \to 0$.*

Proof. The theorem is proved by using the facts of Lemma 1 and 2. □

In the following a similar analysis is shown for $\theta = 0.5$ in (2.10).

Lemma 3. *(**Monotonicity of the Crank-Nicolson Scheme**) When $\theta = 0.5$ in equation (2.10), the Crank-Nicolson scheme is monotone if $\frac{\sigma^2 S}{\Delta S} - r \geq 0$ and $\frac{1}{\Delta t} - r \geq 0$.*

Proof. Let $g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n) = 0$ represent the discrete equation (2.10) for $\theta = 0.5$; then again it is necessary to show that $\forall \varepsilon > 0$

(a) $g_i(V_{i-1}^{n-1}+\varepsilon, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n) \geq g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n)$

(b) $g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}+\varepsilon, V_{i-1}^n, V_i^n, V_{i+1}^n) \geq g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n)$

(c) $g_i(V_{i-1}^{n-1}, V_i^{n-1}+\varepsilon, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n) \leq g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n)$
(d) $g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n+\varepsilon, V_i^n, V_{i+1}^n) \geq g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n)$
(e) $g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n+\varepsilon, V_{i+1}^n) \geq g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n)$
(f) $g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n+\varepsilon) \geq g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n)$

The inequalities (b), (c) and (f) hold immediately again from equation (2.16). For inequality (a),

$$\begin{aligned} g_i(V_{i-1}^{n-1}+\varepsilon, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n) \geq & g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n) + \frac{\varepsilon S_i \left(\frac{(\sigma_i^{n-1})^2 S_i}{\Delta S} - r\right)}{4\Delta S} \\ \geq & g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n), \end{aligned} \tag{2.19}$$

which holds from the initial assumptions.

Similarly for the inequality (d),

$$\begin{aligned} g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n+\varepsilon, V_i^n, V_{i+1}^n) \geq & g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n) + \frac{\varepsilon S_i \left(\frac{(\sigma_i^{n})^2 S_i}{\Delta S} - r\right)}{4\Delta S} \\ \geq & g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n), \end{aligned} \tag{2.20}$$

which also follows from the assumptions.

Finally for the inequality (e),

$$\begin{aligned} g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n+\varepsilon) \geq & g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n) + \varepsilon\left(\frac{1}{\Delta t} - r\right) \\ \geq & g_i(V_{i-1}^{n-1}, V_i^{n-1}, V_{i+1}^{n-1}, V_{i-1}^n, V_i^n, V_{i+1}^n), \end{aligned} \tag{2.21}$$

which also holds from the assumptions and concludes the proof. □

Lemma 4. *(**Stability of the Crank-Nicolson Scheme**) When $\theta = 0.5$ in equation (2.10), the Crank-Nicolson scheme is unconditionally stable.*

Proof. The proof of stability can be found in [99]. □

Theorem 2. *The Crank-Nicolson scheme converges to the viscosity solution of equation (1.3) when $\Delta S \to 0$ and $\Delta t \to 0$.*

Proof. The theorem is proved by using the facts of Lemma 3 and 4. □

The conditions of Lemma 1 and Lemma 3 can sometimes be violated when using improper grid sizes or when the interest rate is large. Possible remedies could be applying upwinding schemes for the discretisation of the convection term which has been examined in [81], or restricting to the case of small interest rates followed in the numerical experiments in this thesis.

2.3.2 Numerical Issues

By using the finite difference scheme (2.10), the problem of solving equation (2.6) becomes solving the nonlinear system (2.11) or (2.12) depending on the chosen scheme. However, two main problematic issues may occur in the numerical computation especially when the time close to maturity.

The first issue arises from the non-smooth terminal condition, for examples, the European call and put options. It is known that the payoff functions are not smooth at $V(S = K, t = T)$, and the infinite behaviour of Gamma, which is equal to $\frac{\partial^2 V}{\partial S^2}$, is inevitable due to the discontinuity of Delta which is equal to $\frac{\partial V}{\partial S}$. Even if choosing suitable number of grid points to avoid calculating second derivative on $V(S = K, t = T)$, such issue still occurs when the grid size of spatial discretisation is too small. This infinite behaviour of Gamma can lead to instabilities in the numerical solvers because Gamma is actually an input of the volatility function which influences directly the entries of the nonlinear systems (2.11) and (2.12).

The second issue can be observed from equation (2.7), or equation (2.8) if considering a non-constant liquidity profile function that singularity may occur if the denominator is equal or very close to zero. This again could generate difficulties to construct a robust solver.

In [43], the authors proposed a smoothed version of the nonlinear Black-Scholes equation to solve numerically like

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma_0^2 S^2 \max\left(\alpha_0, \frac{1}{1-\min\left(\alpha_1, \lambda(S)S\frac{\partial^2 V}{\partial S^2}\right)}\frac{\partial^2 V}{\partial S^2}\right) = 0,$$

where $r = 0$ in this case and α_0, α_1 are positive constants. This setting can guarantee that the denominator is always greater than zero, and avoid problems when the volatilities tending to be too large or small which performs an easy adjustment for the second issue. The authors also suggested replacing the terminal condition with the prices from the standard Black-Scholes formula with constant volatility from $t = T$ to $t = T - \varepsilon$, and starting using the implicit finite difference scheme to solve the nonlinear equation from $t - T - \varepsilon$ to $t = 0$. ε is recommended to choose typically 8% of the lifetime of the contract. This shifting trick could potentially avoid the first issue since the terminal condition is no longer non-smooth. However, it could bring too many strong assumptions within that time period as the Black-Scholes formula is employed to obtain the option price.

Both issues were also investigated by Glover *et al* described in the full-feedback model in [48]. The strategy to handle the numerical difficulties provided by the authors is based on the Keller scheme (see [68]) which considers the nonlinear Black-Scholes equation as a system of two first-order equations in $V(S,t)$ and $V_S(S,t) = \frac{\partial V}{\partial S}(S,t)$. The domain for S is split into two regions, namely $S > K$ and $S < K$ and the grid points have to be chosen such that $S = K$ can coincide with one of the the grid points on S. Then at $S = K$, it is necessary to compute two values of the option price such as $V^+ = V(S = K^+, t), V^- = V(S = K^-, t)$ and

Delta $V_S^+ = V_S(S = K^+, t), V_S^- = V_S(S = K^-, t)$ such that

$$V^- = V^+,\ V_S^+ = V_S^- - 1,$$

for call option, and

$$V^- = V^+,\ V_S^+ = V_S^- + 1,$$

for put option. With such adjustment, the computations were shown highly robust and independent of grid which can remedy the numerical issues.

There are other problematic issues which were analysed from the asymptotic analysis close to expiry in [48] and can happen either with large λ or small σ. These kinds of numerical difficulties may be from the insufficient modelling of considering the nonlinear volatility in option pricing. However, the techniques provided in [48] enable to deal with basic problems with suitable parameters, and to incorporate with the numerical methods to solve the nonlinear systems (2.11) and (2.12) which are introduced in the next section.

2.4 Root-finding Problem

With the finite difference discretisation, equation (2.1) becomes a nonlinear system like those shown in equation (2.11) or (2.12). These nonlinear systems cannot be solved directly in a backward time-marching scheme as the entries of the tri-diagonal matrices depend on the solution. Therefore it is necessary to apply an iterative scheme to obtain the solution.

A standard way is to consider solving the nonlinear system as a root-finding problem, namely defining the root-finding function G as

$$G(V^{n-1}) = H(V^{n-1})V^{n-1} - V^n = 0, \tag{2.22}$$

when using the fully implicit scheme, and

$$G(V^{n-1}) = H_1(V^{n-1})V^{n-1} - H_2(V^n)V^n = 0, \tag{2.23}$$

when using the Crank-Nicolson scheme.

Note that V^{n-1} is the required solution at each time step in this backward time-marching process, and V^n is the known value at the current time step. In a given iterative scheme to solve equation (2.22) or (2.23), the nonlinear volatility depends on the unknown value V^{n-1} which is initially approximated by an initial guess V^*, i.e.

$$\sigma_i^{n-1} \approx \sigma_0 \left(1 - \rho S_i \frac{V_{i+1}^* - 2V_i^* + V_{i-1}^*}{(\Delta S)^2}\right)^{-1},$$

and in the following sections two different iterative schemes are introduced.

2.4.1 Frozen Coefficient Method

The frozen coefficient method is widely used when solving such nonlinear problem as the implementation simply uses the idea of fixed point iteration. In each iterative step, the stopping condition is to check whether the difference of the solved solution and the previous iteration is smaller than a given tolerance. Algorithm 1 describes the details and note the N_{ite} is a maximum tolerated number of iterations which judges whether the iterative scheme is convergent or not. The initial guess V^* at the first time step is picked up from the payoff function, and for the rest of the time steps it is chosen to be the solution from the previous time step.

2.4.2 Newton-Raphson Method

Algorithm 1 is an easy way to solve equation (2.1). However, a reliably convergent result is normally only obtained when the temporal discretisation size Δt is small. This is a drawback of this method as it may force the use of a large number of grid points along the temporal

Algorithm 1: Frozen Coefficient Method

Input: terminal condition $V^{N_t-1} = V(S,t=T)$, initial guess V^*, tol
Output: $V^0 = V(S,t=0)$
for $n = N_t - 1 : 1$ **do**
  $V^* = V^n$;
  for $k = 1 : N_{ite}$ **do**
    1. Solve equation (2.22) or (2.23) to get V^{n-1} ;
    2. $err = V^* - V^{n-1}$;
    if $\|err\| < tol$ **then**
      $V^{n-1} = V^*$, break;
      else
        $V^* = V^{n-1}$, go back to 1.;
      end
    end
  end
end

axis with little gain compared to using fully explicit scheme.

Newton-Raphson method in fact offers a better adaptive root search in the iterative scheme. The main idea is to have a corrective direction to help updating the root-finding algorithm which requires the Jacobian matrix $Jac(G)$ of the root-finding function G. Algorithm 2 describes the details of solving the problem by Newton-Raphson method. Again the initial guess at the first time step is often picked up as the same value from payoff function to ensure it is within the convergent region, and for the rest of time steps it is chosen to be the solution from the previous time step. $Jac(G(V^*))$ means the Jacobian matrix evaluated at V^*. The stopping condition shown in Algorithm 2 is designed to check whether the root-finding function is close enough to 0 as

$$\frac{\|G(V^*)\|}{\max(1,\|V^*\|)} < tol,$$

here tol is a given tolerance which usually is chosen to be a small number, and $\|V^*\|$ represents the norm of the vector V^*. In fact some alternative stopping conditions can be

considered like checking the updating term *res* such as

$$\frac{\|res\|}{\max(1,\|V^*\|)} < tol,$$

where

$$res = -\left[Jac(G(V^*))\right]^{-1} G(V^*).$$

Fig. 2.4 shows the same behaviour of using these two different stopping conditions on the number of iterations for the first ten time steps when solving the Frey-Patie model.

Algorithm 2: Newton-Raphson Method

Input: terminal condition $V^{N_t-1} = V(S, t = T)$, initial guess V^*, tol
Output: $V^0 = V(S, t = 0)$
for $n = N_t - 1 : 1$ **do**
  $V^* = V^n$;
  for $k = 1 : N_{ite}$ **do**
    1. Calculate $G(V^*)$ by equation (2.22) or (2.23) ;
    2. Calculate res = $-\left[Jac(G(V^*))\right]^{-1} G(V^*)$;
    3. **if** $\frac{\|G(V^*)\|}{\max(1,\|V^*\|)} < tol$ **then**
      $V^{n-1} = V^*$, break;
      else
        $V^* = V^* + res$, go back to 1.;
      end
    end
  end
end

Jacobian Matrix

One difficulty of implementing a root-finding solver with Newton-Raphson method is to compute the Jacobian matrix efficiently. It is time-consuming using a small perturbation to calculate this because it requires the evaluation of the root-finding function twice. Instead, a derived formula can be achieved by applying a decomposition

$$H(V^{n-1}) = \Sigma^{n-1} H_a + H_b, \text{ where } \Sigma^{n-1} = \mathrm{Diag}((\sigma_i^{n-1})^2).$$

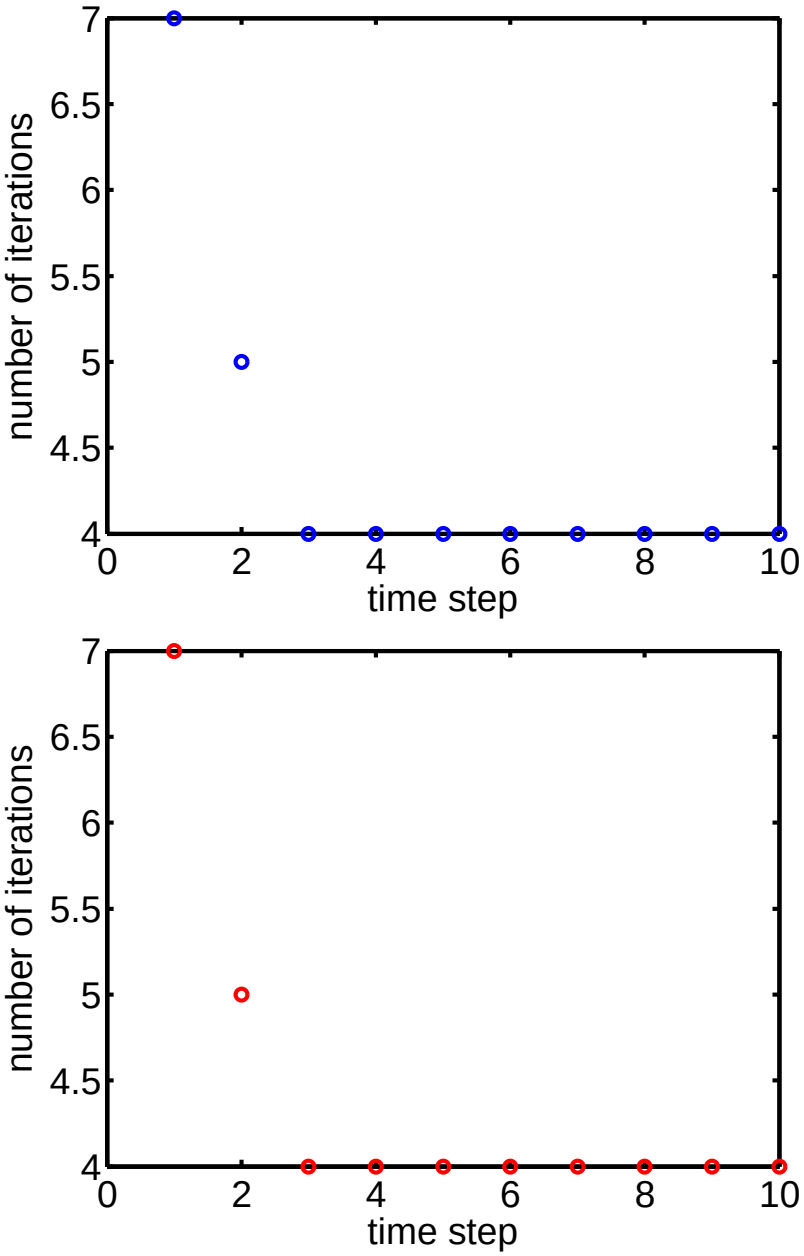

Fig. 2.4 Number of iterations during the first ten time steps for solving the Frey-Patie model with $N_S = 121, N_t = 61$. The upper plot uses the root-finding function as a stopping condition and the lower plot uses the residue as a stopping condition.

Note that H_a and H_b are constant tridiagonal matrices. By using this decomposition, the Jacobian matrix of G becomes

$$Jac(G(V^{n-1})) = \frac{\partial[H(V^{n-1})V^{n-1}]}{\partial V^{n-1}} = H(V^{n-1}) + \text{Diag}(H_a V^{n-1})\nabla(\Sigma^{n-1}),$$

where

$$\nabla(\Sigma^{n-1}) = ((\nabla(\sigma_1^{n-1})^2)^T, (\nabla(\sigma_2^{n-1})^2)^T, ..., (\nabla(\sigma_{N_S-2}^{n-1})^2)^T)^T,$$

here $\nabla(\sigma_i^{n-1})^2$ is treated as a row vector. Then for the Frey-Patie model in equation (2.7), the formulas are obtained as

$$H_a = \frac{\theta \Delta t S^2}{2(\Delta S)^2} \times [1, -2, 1],$$

$$(\nabla(\sigma_i^{n-1})^2)^T = \frac{2\sigma_i^{n-1}\sigma_0 \rho S_i}{\left(\Delta S\left(1 - \rho S_i \frac{V_{i+1}^{n-1} - 2V_i^{n-1} + V_{i-1}^{n-1}}{(\Delta S)^2}\right)\right)^2}.$$

Here again the matrix form $[1, -2, 1]$ means a tri-diagonal matrix with lower diagonal entries equal to 1, main diagonal entries equal to -2 and upper diagonal entries equal to 1.

Damped Updating

Newton-Raphson method provides an efficient way to find the root. Sometimes the updating direction may become too large resulting an unnecessarily large change to the current approximate solution leading to a loss in accuracy. Equally this large change may lead to oscillations of the root-finding function affecting the convergence of the solution. In order to avoid this situation, a damping factor η is usually inserted in order to provide a smaller and safer correction in the updating process. This leads to a change in the updating formula listed in Algorithm 2, to

$$V^* = V^* + \eta \times res,$$

where $res = -\left[Jac(G(V^*))\right]^{-1} G(V^*)$. η may be chosen as 2^{-m} as suggested in [69] such that m is the smallest integer satisfying

$$\left\|G(V^* + 2^{-m} \times res)\right\| < (1 - \alpha 2^{-m}) \left\|G(V^*)\right\|,$$

where α is usually chosen as a small number. When taking $\alpha = 0$, this setting can guarantee that G would be monotone decreasing in the iterative loop and avoid any oscillation.

Fig 2.5 shows an example of using a damped factor in Newton-Raphson method to achieve an adaptive updating. It can be observed that using damped Newton's method avoids the oscillations of the root-finding function, although it may require evaluation of the root-finding function at least twice within a Newton-Raphson iteration. However this extra cost can guarantee a more robust result when applying Newton-based solvers.

2.4.3 Numerical Experiments

The numerical experiment described here was to solve a European call option with Frey-Patie model where the terminal and boundary conditions were taken as

$$\begin{cases} V(S,T) = (S-K)^+, & \text{for} \quad 0 \le S < S_{max} \\ V(0,t) = 0, & \text{for} \quad 0 \le t \le T \\ V(S,t) = S - Ke^{-r(T-t)}, & \text{when} \quad S = S_{max} \end{cases}$$

and the model parameters were chosen to be: $K = 100, S_{min} = 0, S_{max} = 300, T = 1$. The liquidity parameter in the Frey-Patie model in equation (2.7) was chosen to be $\rho = 0.01$. The tolerance for Newton's iterations was set to be $tol = 10^{-10}$. The experiments were designed to examine different σ_0 and r.

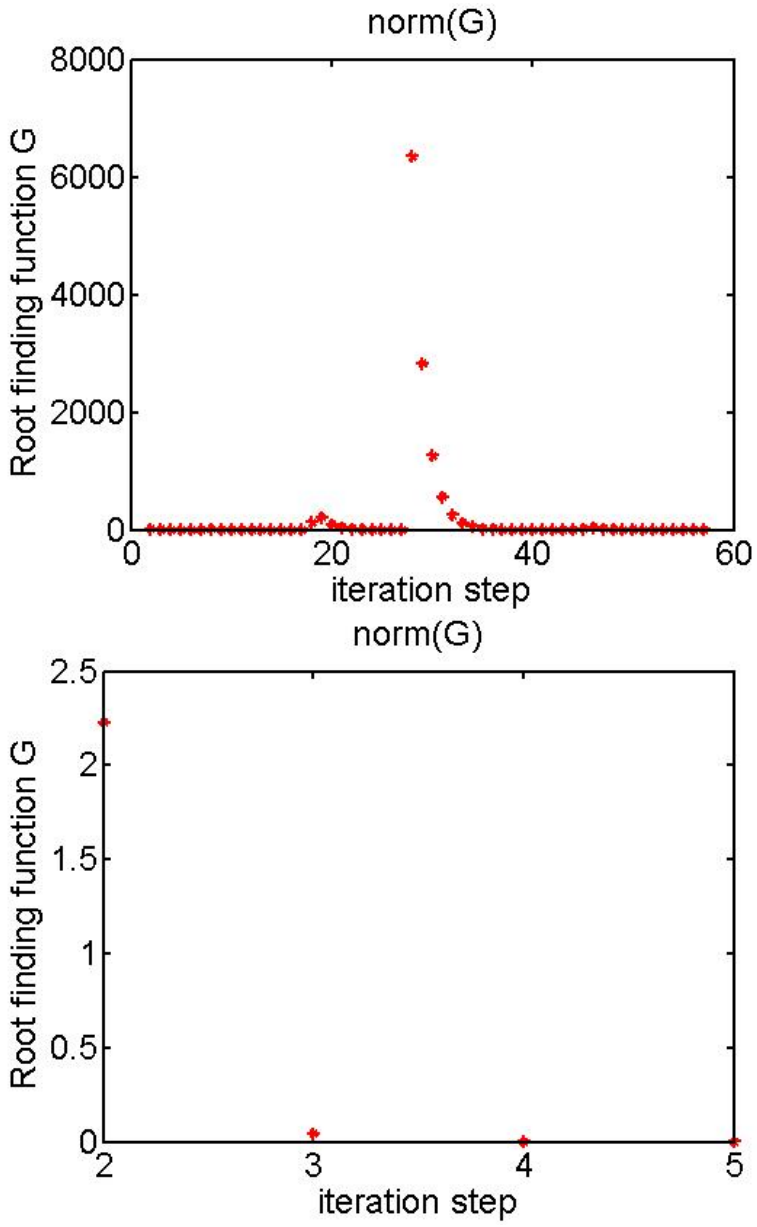

Fig. 2.5 Results of solving the Frey-Patie model by Newton-Raphson method without damped updating (the upper plot) and with damped updating (the lower plot).

Table 2.1 Fully implicit scheme with the l_2 norm for the case $r = 0.01$ and $\sigma = 0.4$.

N_S	N_t	$V(Newton)$	$V(Frozen)$	$Diff$	Err	R	eoc	$\hat{Err}$	$\hat{R}$	$e\hat{o}c$
61	31	16.4039	16.4039	4.2e-09	0.058	2.35	1.23	0.105	2.26	1.18
121	61	16.4627	16.4627	1.0e-09	0.025	2.10	1.07	0.046	2.17	1.12
241	121	16.4877	16.4877	1.7e-08	0.011	1.95	0.96	0.021	2.27	1.18
481	241	16.4997	16.4997	2.4e-08	0.006	1.87	0.90	0.009	2.86	1.51
961	481	16.5058	16.5058	1.3e-08	0.003			0.003		
1921	961	16.5091	16.5091	6.9e-09						

Table 2.2 Fully implicit scheme with the l_∞ norm for the case $r = 0.01$ and $\sigma = 0.4$.

N_S	N_t	$V(Newton)$	$V(Frozen)$	$Diff$	Err	R	eoc	$\hat{Err}$	$\hat{R}$	$e\hat{o}c$
61	31	16.4039	16.4039	2.8e-09	0.058	2.35	3.13	0.105	2.26	1.18
121	61	16.4627	16.4627	2.2e-09	0.025	2.10	1.07	0.046	2.17	1.12
241	121	16.4877	16.4877	8.2e-08	0.011	1.95	0.96	0.021	2.27	1.18
481	241	16.4997	16.4997	5.2e-08	0.006	1.87	0.90	0.009	2.86	1.51
961	481	16.5058	16.5058	3.0e-08	0.003			0.003		
1921	961	16.5091	16.5091	2.0e-08						

The aim of the numerical experiment was to examine the experimental order of convergence (eoc) at the point $V(S = K, t = 0)$ constructed from the convergence rate of the error. Two possible choices of defining the ratio were considered. Let V_h be the solution evaluated with mesh sizes $h\Delta S$ for the spatial discretisation, and $h\Delta t$ for temporal discretisation; then the ratio of error with asymptotic behaviour was constructed as follows

$$R = \frac{\|V_h - V_{h/2}\|}{\|V_{h/2} - V_{h/4}\|}, \tag{2.24}$$

where the eoc is equal to $\log_2(R)$. The other choice is to take the solution evaluated with a small mesh size to be the true solution and then the ratio of error can be defined as:

$$\hat{R} = \frac{\|V_h - \hat{V}\|}{\|V_{h/2} - \hat{V}\|}, \tag{2.25}$$

where $\hat{V}$ is the solution evaluated with the most refined grid and the $e\hat{o}c$ is equal to $\log_2(\hat{R})$.

Table 2.1 and 2.2 show the results of using the fully implicit scheme. The value $Diff$ is the difference of the solutions from Newton-Raphson method and the frozen coefficient

Table 2.3 Average number of iterations in the first 5 time steps for the case $r = 0.01$ and $\sigma = 0.4$.

N_S	N_t	$V(Newton)$ with l_2	$V(Frozen)$ with l_2	$V(Newton)$ with l_∞	$V(Frozen)$ with l_∞
61	31	4.6	7.4	4.6	7.4
121	61	4.8	8.4	4.8	8.4
241	121	5.2	9.8	5.4	9.6
481	241	5.6	12	6.2	11.6
961	481	6.6	17.2	7	16.6
1921	961	10.8	64	11.6	60.2

method which demonstrates both converge to the the same solution. The behaviour of the experimental order of convergence under the l_2 norm and l_∞ is the same and both definitions of R and $\hat{R}$ approximately match the theoretical value 1 for the fully implicit scheme. From Table 2.3 the average number of iterations for the first five time steps when using Newton-Raphson method and the frozen coefficient method are presented. The number of iterations when using the frozen coefficient method is much higher when the grid size is smaller and sometimes exceeds the maximum number of tolerated iterations causing the iterative process to fail. Therefore Newton-Raphson method is a more robust way of solving this nonlinear system.

It is worthwhile noting that for *eôc*, which uses the solution with fine meshes to represent the true solution $\hat{V}$, it is sometimes not easy to obtain an asymptotic behaviour if $\hat{V}$ is not accurate enough. Therefore another example presented here is to take the solution from the approximate formula in equation (1.14), to be $\hat{V}$ and then calculate *eôc*. Fig. 2.6 shows the results of using the fully implicit scheme with the cases $\rho = 0.01, 0.005, 0.001$. It can be observed that when ρ is getting smaller, the values *eôc* reach the theoretical value. The results also demonstrate that the approximate formula in equation (1.14) works well when the parameter ρ is small as discussed in [29].

Table 2.4 presents the same example but using the Crank-Nicolson scheme. The solution becomes unstable when the spatial grid point reaches 121, due to ΔS becoming too small and the oscillations occur around the region $S = K$ (as shown in Fig 2.7 when using central

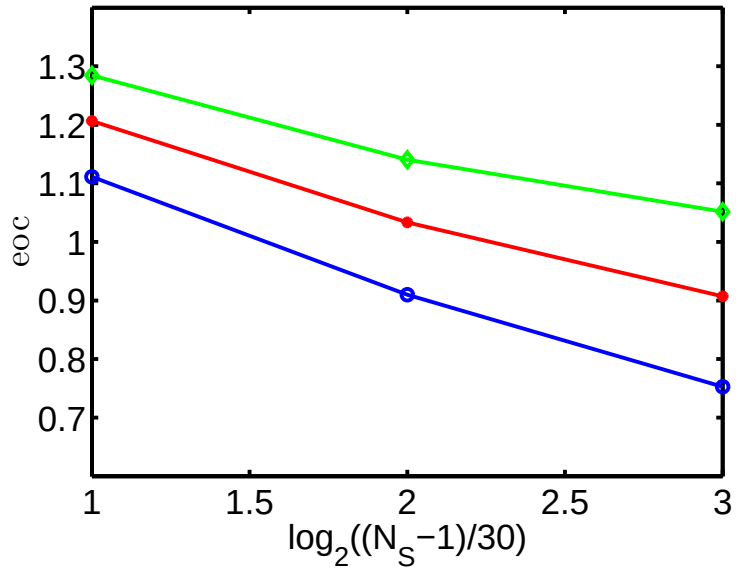

Fig. 2.6 *eôc* with different liquidities. The line with blue circle is $\rho = 0.01$, the line with red star is $\rho = 0.005$ and the line with green diamond is $\rho = 0.001$.

Table 2.4 Crank-Nicolson scheme with the l_2 norm for the case $r = 0.01$ and $\sigma = 0.4$.

M	N	$V(Newton)$	$V(Frozen)$	$Diff$	Err	R	eoc
61	31	16.5483	16.5483	8.8e-10	0.208	1.03	0.04
121	61	16.7566	16.9196	0.16	0.201		
241	121	16.9585	fail				

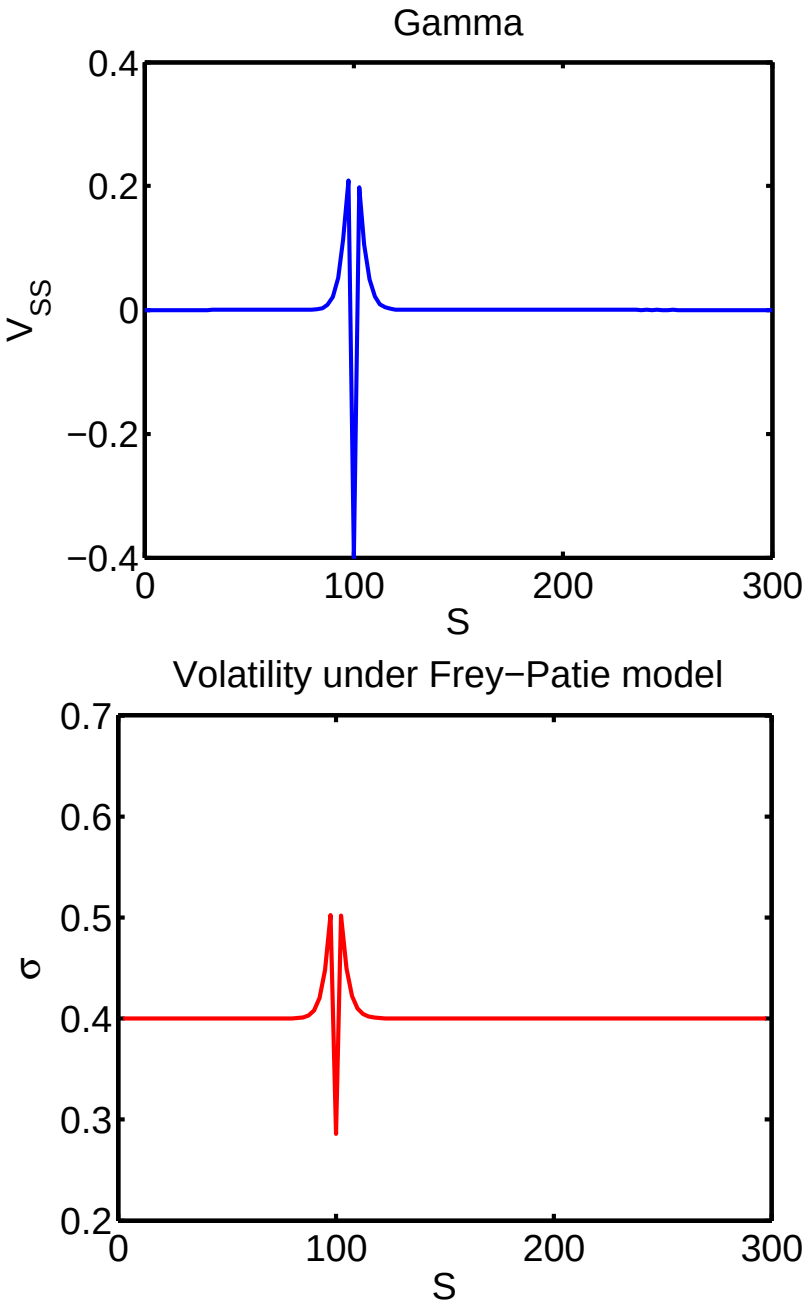

Fig. 2.7 Oscillation occurs when using a small grid size in the Crank-Nicolson scheme.

difference method to evaluate V_{SS}). The oscillation influences strongly the order of accuracy and some improvements can be applied to recover it which will be introduced in the next section.

2.5 Accuracy Improvement

As mentioned in the previous section, the numerical scheme sometimes cannot achieve the theoretical order of accuracy, as the nonlinearity may affect some of the properties of the numerical solutions, e.g. like continuity or smoothness, or oscillations which can occur when using smaller grid sizes. There are some strategies which potentially can recover the expected accuracy of a scheme without requiring highly refined grids.

2.5.1 Coordinate Stretching

A coordinate stretching technique used in [94] provides an idea to adjust the grid points around the most important region, normally close to $S = K$. The stretched coordinate x is defined by

$$S = \frac{K}{\lambda}\sinh(x - L_S) + K, \tag{2.26}$$

or inversely as

$$x = \sinh^{-1}\left(\frac{\lambda}{K}(S-K)\right) + L_S, \tag{2.27}$$

where L_S is the parameter to control the amount of stretching, and $\lambda = \sinh(L_S)$. Let $S^* = S/K$, then from equation (2.27),

$$S^* = \frac{(\sinh(x - \sinh^{-1}(\lambda))}{\lambda} + 1, \tag{2.28}$$

which leads the adjustment of taking small values around $S^* = 1$ which is equivalent to $S = K$ as shown in Fig 2.8.

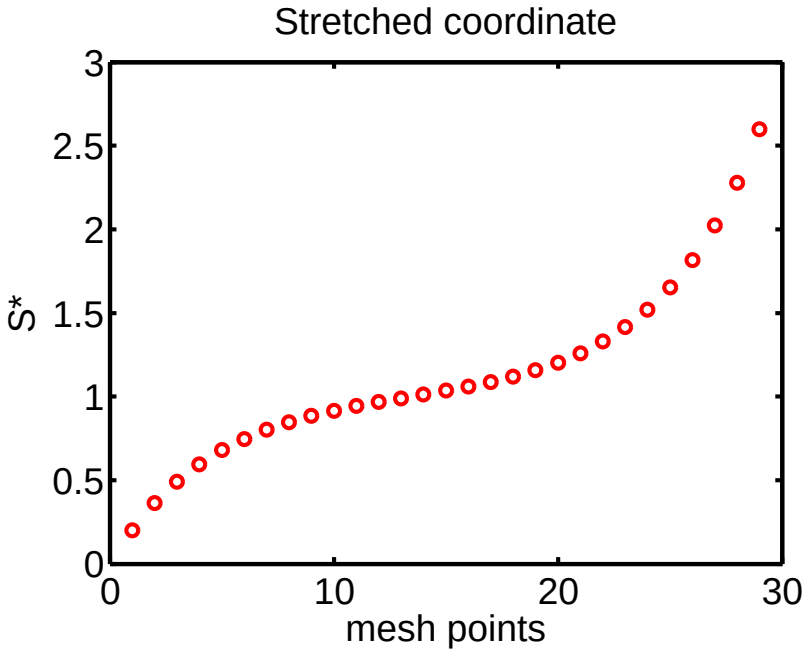

Fig. 2.8 New coordinate points on the non-stretched mesh which shows more points are employed close to the region $S = K$.

Applying this coordinate stretching, equation (2.1) becomes

$$\frac{\partial V}{\partial t} + \frac{1}{2}(\sigma T_\lambda(x))^2 \frac{\partial^2 V}{\partial x^2} + \tilde{r} T_\lambda(x) \frac{\partial V}{\partial x} - rV = 0, \tag{2.29}$$

where

$$T_\lambda(x) = \frac{\sinh(x - L_S) + \lambda}{\cosh(x - L_S)},$$

$$\tilde{r} = r - \frac{1}{2}\sigma^2 T_\lambda(x) \tanh(x - L_S),$$

and the developed Newton-based solver described in Sec 2.4.2 can be applied as before to solve equation (2.29). Table 2.5 shows the results using the fully implicit scheme with coordinate stretching with the control size $L_S = 3$ for the same numerical experiment provided in Sec. 2.4.3 and the convergent behaviour of the ratio can be achieved without using refined grid points.

Table 2.5 Fully implicit scheme with the l_2 norm for the case $r = 0.01$ and $\sigma = 0.4$ under stretched coordinate.

N_S	N_t	Err	R	eoc	$\hat{Err}$	$\hat{R}$	$\hat{eoc}$
31	16	0.063	1.78	0.83	0.125	2.02	1.01
61	31	0.035	2.03	1.02	0.061	2.35	1.23
121	61	0.017	2.01	1.01	0.026	3.00	1.58
241	121	0.008			0.008		
481	241						

2.5.2 Rannacher Timestepping

As observed from Fig 2.7, using the Crank-Nicolson scheme can have trouble with small sizes of mesh which harms the order of convergence and the solution can converge to a wrong one. Rannacher [100] proposed a methodology to remedy it. The idea is to replace the first few Crank-Nicolson timesteps by fully implicit timesteps which less likely to oscillate. Giles [47] also suggested some other techniques based on manipulating the timesteps of a fully implicit scheme which can avoid this type of numerical oscillation.

Table 2.6 shows the result of replacing the first two timesteps by using the fully implicit scheme, and for the remainder of the timesteps using the Crank-Nicolson scheme. Compared with Table 2.4 it demonstrates that the oscillation problem can be cured by using Rannacher timestepping even though second order accuracy is not achieved. Fig. 2.9 also presents the results of manipulating the timesteps which are replaced by the fully implicit scheme and it can be found that replacing more timesteps may harm the accuracy. The choice of the number of timesteps with Rannacher timestepping in fact depends on the case and needs empirical study.

Finally Table 2.7 gives the results of combining coordinated stretching with the control size $L_S = 3$ and Rannacher timestepping which replacing the first two time steps by fully implicit scheme. It can be observed that without using large number of grid points the second order accuracy can be recovered.

Table 2.6 Crank-Nicolson scheme with the l_2 norm for the case $r = 0.01$ and $\sigma = 0.4$. Rannacher timestepping is applied to replace the first two time steps by using fully implicit scheme.

N_S	N_t	Err	R	eoc	$\hat{Err}$	$\hat{R}$	$\hat{eoc}$
61	31	0.0263	3.29	1.72	0.0398	2.95	1.56
121	61	0.0080	2.61	1.38	0.0134	2.46	1.30
241	121	0.0030	2.03	1.02	0.0054	2.28	1.18
481	241	0.0015	1.71	0.77	0.0023	2.71	1.43
961	481	0.0008			0.0008		
1921	961						

Table 2.7 Crank-Nicolson scheme with the l_2 norm for the case $r = 0.01$ and $\sigma = 0.4$ under stretched coordinate. Rannacher timestepping is applied to replace the first two time steps by using fully implicit scheme.

N_S	N_t	Err	R	eoc	$\hat{Err}$	$\hat{R}$	$\hat{eoc}$
31	16	0.0067	1.59	0.67	0.0123	2.21	1.14
61	31	0.0042	4.02	2.01	0.0055	4.16	2.05
121	61	0.0010	3.74	1.90	0.0013	4.73	2.24
241	121	0.0002			0.0002		
481	241						

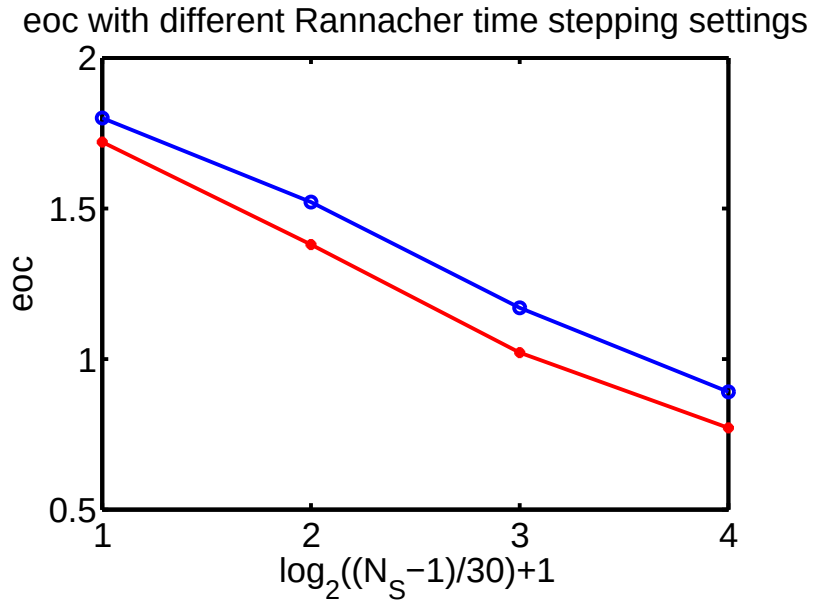

Fig. 2.9 eoc of using Rannacher timestepping for replacing the first one time step (blue circle) and first two time steps (red star) by fully implicit scheme. Here N_S presents the grid points for spatial discretisation.

2.5.3 Richardson Extrapolation

A technique of using Richardson extrapolation stated in [47, 94] is also worth trying. The idea is to eliminate the lower order term from the truncation error by taking some linear combination of the existing solutions. To be clear, suppose V is the exact solution of the option price, and $V(\Delta S, \Delta t)$ is the solution calculated by finite difference method with temporal mesh size equal to Δt and spatial mesh size equal to ΔS. Then for the fully implicit scheme it is known that the leading truncation error term should satisfy $O((\Delta S)^2 + \Delta t)$ which implies:

$$V(\Delta S, \Delta t) = V + c_1(\Delta S)^2 + c_2(\Delta t) + h.o.t, \tag{2.30}$$

here c_1, c_2 are constants and $h.o.t$ represents other higher order term. Similarly

$$V\left(\frac{\Delta S}{2}, \frac{\Delta t}{2}\right) = V + c_1\left(\frac{\Delta S}{2}\right)^2 + c_2\left(\frac{\Delta t}{2}\right) + h.o.t. \tag{2.31}$$

Now observe that

$$V(\Delta S, \Delta t) - V\left(\frac{\Delta S}{2}, \frac{\Delta t}{2}\right) = \frac{3}{4}c_1(\Delta S)^2 + \frac{1}{2}c_2\Delta t + h.o.t,$$

which shows

$$2V\left(\frac{\Delta S}{2}, \frac{\Delta t}{2}\right) - V(\Delta S, \Delta t) = V + O((\Delta S)^2 + (\Delta t)^2),$$

and it indicates the formula for achieving the second order accuracy on time by extrapolating the solutions from the fully implicit scheme as

$$V_{R(k,k/2)} = 2V\left(\frac{\Delta S}{2k}, \frac{\Delta t}{2k}\right) - V\left(\frac{\Delta S}{k}, \frac{\Delta t}{k}\right), \;\; k = 2^{i-1}\ \forall i \in \mathbb{N}.$$

Following the same idea, for the case of using Crank-Nicolson scheme, the leading truncation error term satisfies $O((\Delta S)^2 + (\Delta t)^2)$ which implies:

$$V(\Delta S, \Delta t) = V + d_1(\Delta S)^2 + d_2(\Delta t)^2 + h.o.t, \tag{2.32}$$

Table 2.8 Fully implicit scheme with the l_2 norm for the case $r = 0.01$ and $\sigma = 0.4$.

N_S	N_t	$V(Newton)$	Err	R	eoc	$V(Richardson)$	Err	R	eoc
31	16	16.2437	0.160	2.72	1.45				
61	31	16.4039	0.058	2.35	1.23	16.5641	0.0425	4.92	2.30
121	61	16.4627	0.025	2.10	1.07	16.5215	0.0086	7.20	2.85
241	121	16.4877	0.011	1.95	0.96	16.5128	0.0012	4.29	2.10
481	241	16.4997	0.006	1.87	0.90	16.5116	0.0002		
961	481	16.5058	0.003			16.5119			
1921	961	16.5091							

here d_1, d_2 are constants and $h.o.t$ represents other higher order term. Similarly

$$V\left(\frac{\Delta S}{2}, \frac{\Delta t}{2}\right) = V + d_1\left(\frac{\Delta S}{2}\right)^2 + d_2\left(\frac{\Delta t}{2}\right)^2 + h.o.t, \tag{2.33}$$

and subtracting equation (2.33) by equation (2.32) gives

$$V(\Delta S, \Delta t) - V\left(\frac{\Delta S}{2}, \frac{\Delta t}{2}\right) = \frac{3}{4}\left(d_1(\Delta S)^2 + d_2(\Delta t)^2\right) + h.o.t,$$

which leads to

$$\frac{4}{3}V\left(\frac{\Delta S}{2}, \frac{\Delta t}{2}\right) - \frac{1}{3}V(\Delta S, \Delta t) = V + O((\Delta S)^4 + (\Delta t)^4),$$

and it suggests the formula that the fourth order accuracy could be obtained by extrapolating the solutions from the Crank-Nicolson scheme as

$$V_{R(k,k/2)} = \frac{4}{3}V\left(\frac{\Delta S}{2k}, \frac{\Delta t}{2k}\right) - \frac{1}{3}V\left(\frac{\Delta S}{k}, \frac{\Delta t}{k}\right), \ k = 2^{i-1}\ \forall i \in \mathbb{N}.$$

Note that the extrapolation technique works well mostly for the case with the stable behaviour of the experimental order of convergent. An example is given in Table 2.8 using the results from Table 2.1 to do the extrapolation, which shows the high order accuracy solution can be extrapolated from the solutions on coarse meshes.

2.6 Other Newton-based Solvers

Solving the root-finding problem with Algorithm 2 leads to the numerical solution of the original nonlinear partial differential equation (2.1). In this approach the inversion of the Jacobian matrix was done using a tri-diagonal direct solver such as the Thomas algorithm. This is sometimes called an exact Newton's method as a direct solver is employed. In fact some alternatives approaches may also be considered and introduced next.

2.6.1 Newton-like Methods

Different techniques that approximate the Jacobian matrix or solve the linear system with an iterative solver can be applied. In general these approaches are called Newton-like methods. Essentially these methods are suitable for some difficult situations, e.g. when the cost of evaluating the Jacobian matrix is expensive or when analytic formula doesn't exist, or the matrix structure of generated nonlinear system is complicated and direct solvers are too slow. Two standard techniques are described and compared here.

To simplify the notation, the root-finding problem listed in equation (2.22) and (2.23) are rewritten in the form

$$G(v) = 0,$$

with the updating formula

$$v^{k+1} = v^k + \delta v^k, \;\; \delta v^k = -[Jac(G(v^k))]^{-1} G(v^k),$$

where k represents the iteration step.

Inexact Newton Methods

A major computational cost when applying Newton-Raphson method is the expense of calculating the updating direction δv^k, which requires the inversion of the Jacobian matrix

$[Jac(G)]^{-1}$. Therefore reducing the number of iterations would be a way to shorten the overall computation time. The idea is to perform a Newton iteration approximately, as shown in [23] by finding an update direction which satisfies the inexact Newton condition

$$\left\| G(v^k) + Jac\left(G(v^k)\right)\delta v^k \right\| < \beta \left\| G(v^k + \delta v^k) \right\|,$$

where β is known as the forcing term.

In terms of implementation there are two loops, one being the inner loop for finding the updating direction and the other being the outer Newton iterative loop. The choice of β is important. Small values of β reduce the iteration to simply Newton's method. Other choices of β may not improve the result, but rather lead to a poorer one. Discussions of suitable choices can be found from [69, 72].

Broyden-type Method

Another approach to avoid the excessive computation of the Jacobian matrix is Broyden's method. From experiences gained through numerical experiments, a highly accurate Jacobian matrix in each iteration is often unnecessary to achieve fast convergence. Broyden's method relies on the concept of a generalised secant method which leads to an iterative scheme for approximating the Jacobian matrix,

$$Jac(G)_k = Jac(G)_{k-1} + \frac{\Delta G - Jac(G)_{k-1}\Delta v}{\|\Delta v\|^2}\Delta v^T. \tag{2.34}$$

here again k represents the iteration step. The first few Jacobian matrices in the iteration still need to be obtained numerically by using a finite difference method. Broyden's method then simplifies the evaluation of Jacobian matrix to some additions and multiplications of matrices and vectors. Note the formula in equation (2.34) in most of the cases does change the structure of the original Jacobian matrix especially if it is sparse matrix.

There are several modifications of Broyden's method as shown in [84]. A typical example is to preserve the matrix structure by using the sparse Broyden method introduced by Schubert [104]. However in some cases the implementation of this sparsity preservation method does not save much time. Therefore a simple trick, is to pick up only the tri-diagonal entries after performing each iteration in Broyden's method, and this can achieve convergence behaviour similar to the original matrix with cheaper computation on the numerical linear algebra. Further discussion can be found in [33] which Egorova compared using different Broyden-type methods to solve the Barles-Soner model with nonlinear function shown in equation (1.5).

2.6.2 Deferred Correction Problem

Another Newton-based solver introduced here is based on the deferred correction method which was used in solving initial boundary value problem of ordinary differential equations (see [75, 79]). A similar idea can be applied to solve equation (2.1) with Newton's linearisation. First consider a smooth function F representing the nonlinear Black-Scholes equation (2.1), i.e.

$$F(V_t, V_S, V_{SS}, V) \equiv V_t + \frac{1}{2}\sigma^2 S^2 V_{SS} + rSV_S - rV = 0.$$

Here the abbreviations V_t, V_S, V_{SS} are for partial derivatives of V with respect to t, S and the second derivative with respect to S. Then a linearisation on an approximate value $(V_t^*, V_S^*, V_{SS}^*, V^*)$ of the function F reads as follows:

$$\begin{aligned} &F(V_t^* + e_t, V_S^* + e_S, V_{SS}^* + e_{SS}, V^* + e) \\ &\approx F(V_t^*, V_S^*, V_{SS}^*, V^*) + \frac{\partial F}{\partial V_t} e_t + \frac{\partial F}{\partial V_S} e_S + \frac{\partial F}{\partial V_{SS}} e_{SS} + \frac{\partial F}{\partial V} e, \end{aligned} \tag{2.35}$$

where e is correction term and the partial derivatives are evaluated at $(V_t^*, V_S^*, V_{SS}^*, V^*)$.

Equation (2.35) transforms equation (2.1) into a linear partial differential equation corresponding to a correction term e with zero boundary and initial condition. This equation can

be solved easily if all the coefficients of equation (2.35) are determined. The coefficient can be evaluated either by derived formula or the finite difference method. Eventually after finite difference discretisation, the problem becomes

$$\frac{\partial F}{\partial V_t^*}e^n - \Delta t F(V_t^*, V_S^*, V_{SS}^*, V^*) = H_e^*(V^*)e^{n+1}. \tag{2.36}$$

where H_e is a tri-diagonal matrix and the algorithm of solving equation (2.36) is described as Algorithm 3. Again, an initial guess can be chosen to be the solution from the previous time step. Figure 2.10 shows a difference between the solution solved from root-finding approach (denoted NM1) and deferred correction problem (denoted NM2) under l_2 and l_∞ norm using the same parameters proposed in Sec. 2.4.3 with the tolerance equal to $tol = 10^{-8}$, and both methods converge to almost the same solution. However, the drawback of the deferred correction method is that in the new transformed equation (2.35), the four coefficients are not always guaranteed to be positive values which could make the finite difference scheme become quite unstable or even converge to a wrong solution. Forcing the coefficients to be zero when getting negative values is an option but can change the updating direction which can harm the efficiency of this method.

Algorithm 3: Deferred Correction Problem

Input: terminal condition $V^{N_t-1} = V(S, t = T)$, initial guess V^*, tol
Output: $V^0 = V(S, t = 0)$
for $n = N_t - 1 : 1$ **do**
  for $k = 1 : N_{ite}$ **do**
    1. Calculate $\frac{\partial F}{\partial V_t^*}, \frac{\partial F}{\partial V_S^*}, \frac{\partial F}{\partial V_{SS}^*}, \frac{\partial F}{\partial V^*}$;
    2. Solve equation (2.35) to get e^n ;
    if $\|e^n\| < tol$ **then**
      $V^n = V^*$, break;
      else
        $V^* = V^* + e^n$, go back to 1.;
      end
    end
  end
end

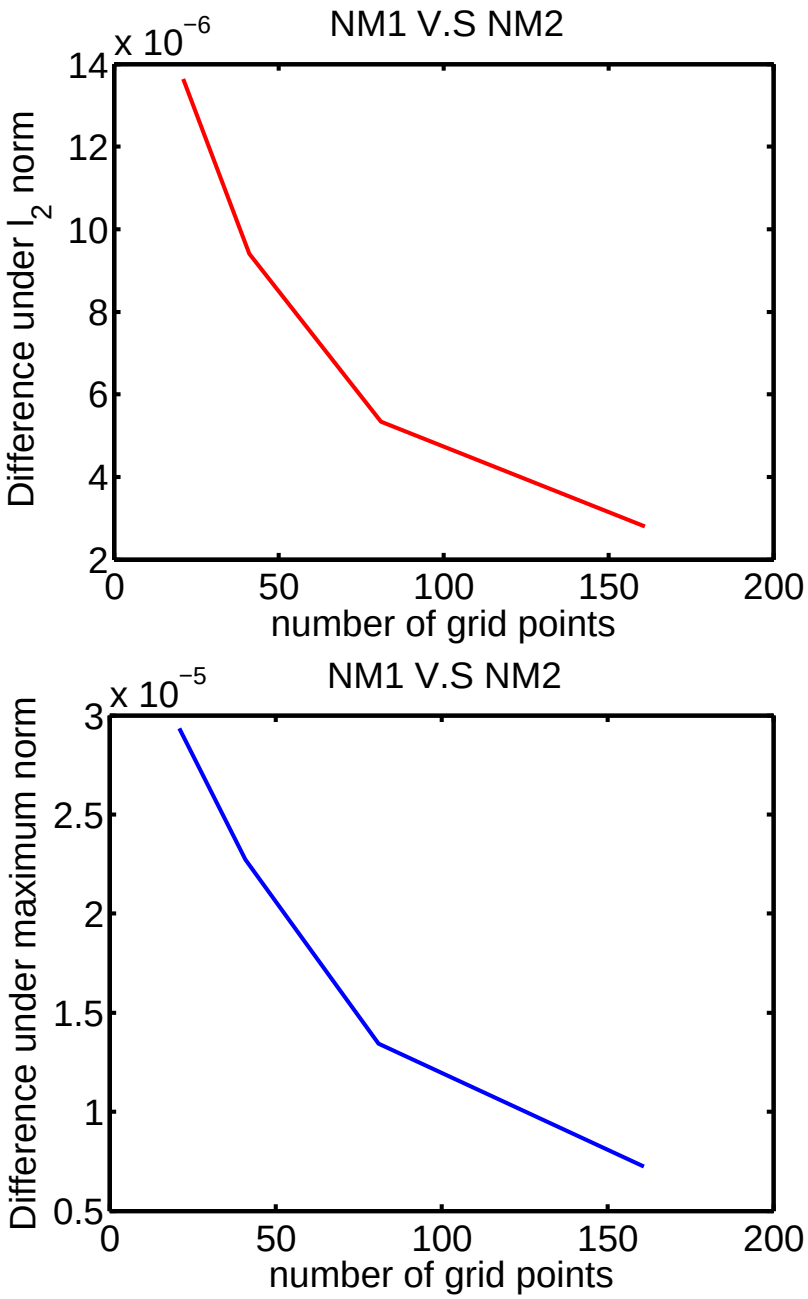

Fig. 2.10 Difference of the solution of solving the Frey-Patie model between the root-finding approach (NM1) and the deferred correction method (NM2) with respect to the number of grid points under the l_2 norm (left) and l_∞ norm (right) when $tol = 10^{-8}$.

2.6.3 Numerical Experiments

The numerical experiment provided here is designed as in Sec. 2.4.3. Tne aim is to compare the complexity of different methods, and the approach taken is to perform the comparisons of using the so-called experimental order of time complexity $eotc$ introduced in [29] as defined below

$$Time = \tilde{c} \times \Delta t^{eotc}$$

and can be expressed as

$$eotc_h = -\frac{\log_2((Time)_h/(Time)_{h/2})}{\log_2((h\Delta t)/(h\Delta t/2))}.$$

here $Time$ means the total CPU computation time and h means evaluation with mesh sizes $h\Delta S, h\Delta t$ for spatial and temporal discretisations, respectively.

Table 2.9 shows the values of $eotc$ for Newton-Raphson method (denoted NM1), Deferred correction method (denoted NM2) and Newton-like method (denoted NM3). The Jacobian matrix of Newton-Raphson method is calculating by the derived formula, and the same for the coefficients for Deferred correction method, and both methods are using Thomas algorithm to solve the tri-diagonal system. The Newton-like method is using inexact Newton's method where the iterative solver is using Jacobi method. The result shows Newton-like method performs slowly and the computation time grows faster than other two methods. The reason is for the concerning equation, the matrix structure is tri-diagonal which is a simple structure, and iterative solver does not have benefit compared with using a simple and fast tri-diagonal direct solver.

Another comparison in Fig. 2.11 shows using different Newton-like methods to solve Frey-Patie Model. The aim is to compare using exact and inexact Newton's methods, combined with finite difference method and Broyden-type method to approximate the Jacobian matrix. The grid points of temporal discretisation is fixed to be $N_t = 1000$ and of spatial discretisation were chosen as $N_S = 50, 100, 150, 200, 250$. Damped updating was also used to guarantee an

Table 2.9 *eotc* for the Frey-Patie model, here NM1 means using the Newton-Raphson method, NM2 means using the deferred correction approach and NM3 means using the inexact Newton's method.

N_S	N_t	$eotc_{NM1}$	$eotc_{NM2}$	$eotc_{NM3}$
40	40	—	—	—
80	80	1.190	1.300	1.410
160	160	1.642	1.531	2.124
240	240	2.479	2.402	2.842
480	480	3.888	3.465	4.241

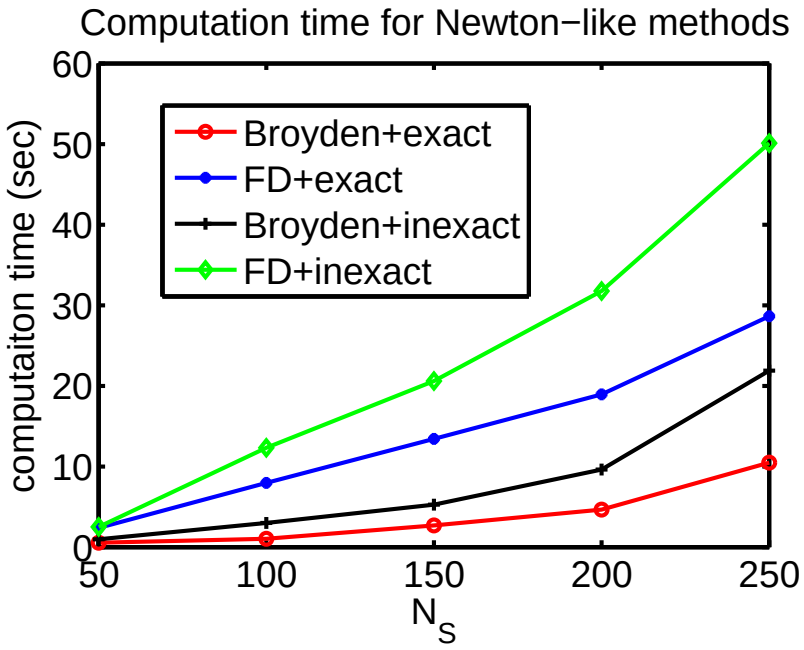

Fig. 2.11 Different strategies to implement Newton-like method.

adaptive updating. It is observed that Broyden-type method can replace the finite difference method to evaluate the Jacobian matrix and reduce the computation time. It shows that if there is no exact formula for Jacobian matrix, then a better strategy in one-dimensional case is to choose exact Newton's method with some approximation of Jacobian matrix like Broyden-type methods.

2.7 Summary

In this chapter, Newton-based solvers are constructed for solving the one-dimensional nonlinear Black-Scholes equation. Several adjustments are applied to make the solver efficient and robust, and some strategies of improving the order of accuracy are also examined. Different numerical experiments were designed that solved the European option case under the Frey-Patie model, demonstrating that these methods and improvements work well. A comparison with variants of Newton-like methods also concludes that using a direct Newton's method is better for the one-dimensional case in terms of the efficiency.

The developed Newton-based solver is able to solve nonlinear option pricing properly for the single option case. In order to impact on the finance industry, it is necessary to have a framework for demonstrating solvers that are able to handle large-scale problems efficiently and productively. A detailed discussion focusing on parallel computing implementations is provided in the next chapter.

Chapter 3

Large-scale Problems with GPU Computing

In this chapter, the aim is to construct a parallel computation framework for the Newton-based solvers developed in Chapter 2 to deal with large-scale nonlinear option pricing problems. The concepts of batch operation and implementations using GPUs are explained. Eventually a batched solver for large-scale nonlinear PDEs was constructed and numerical experiments of achieved performance were examined.

3.1 Introduction

Solving PDE problems with parallel computing has been studied for decades in different areas as shown in [86, 92]. Unlike Monte Carlo simulation which has a large potential being done in parallel, PDE solvers have certain restrictions. The calculations are contained within a time-marching scheme where initial conditions rely on the solution from the previous time step. Nevertheless, at each time step, the core parts, the numerical linear algebra computations, are suitable for parallel algorithms (see [24, 52]) either for explicit or implicit schemes. With proper implementation good parallel performance is still achievable for PDE solvers.

There are many different platforms for implementing parallel algorithms. Using multi-core processors is one of the popular choices as it is able to speed up expensive computations. The hardware accelerator Intel Xeon Phi launched around 2012 enables computing with high throughput and some advantages have been demonstrated for building a triangular solver in [114] and for the multigrid method in [111]. Another popular approach is the use of graphics processing units (GPUs) due to their massively parallel architecture and high bandwidth. Several recent studies concentrated on exploiting the parallelism in the direct solver for the linear system solution which is required when using an implicit scheme. For examples, Egloff [31] and Giles [45] constructed a parallel tri-diagonal solver for solving the linear Black-Scholes equation by implementing the parallel cyclic reduction algorithm, and Zhang [116] used the benefits of a recursive doubling algorithm to design some hyprid algorithms which achieved better performance than the linear solver from LAPACK library [2]. Chang [15] described another implementation using the SPIKE algorithm (see [97, 98]) to solve linear systems with tri-diagonal structure. In [102] a comparison of the benchmark results from different platforms for constructing the multigrid preconditioners is given. László *et al.* [78] also examined the performance of building the linear Black-Scholes PDE solver by using CPU, GPU and FPGA architectures. Essentially it is necessary to consider the target problem when choosing the platform on which to implement high performance computing, with some further considerations on the cost of energy of the machines and software development time as suggested in [46].

Another challenging issue that often occurs in the quantitative finance is the need to solve large-scale problems. These problems arise from needing prices for many different contracts or choices of parameters which result in large-scale PDE problems. A discussion in [30] examined the performance of using GPUs to build massively parallel PDE solvers, and an idea of batch operation was also described in [45] and can be applied to solve different independent PDEs.

In the following sections the emphasis is on two main tasks. The first task is to construct a batched Newton-based solver for handling large-scale nonlinear PDEs. The developed nonlinear solver is decomposed into the different tasks of numerical linear algebra computations in order to easily apply the batch operation. The second task is to set up a framework for doing GPU computing with the constructed batched Newton-based solver. Two possible implementations, using OpenACC and CUDA programming, are explained in detail and compared in terms of the efficiency. Numerical results are shown in the end of this chapter.

3.2 Large-scale Nonlinear Option Pricing Problems

In Chapter 2 it is shown the solution of the nonlinear Black-Scholes equation

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0, \tag{3.1}$$

can be transformed into a repeated root-finding problem in a discretised nonlinear system, namely

$$G(V^*) = H(V^*)V^* - f = 0, \tag{3.2}$$

where V^* is the pursued solution, f represents the right hand side value which can be evaluated from known values at the current time step, and H is a tri-diagonal matrix for which the entries depend on the solution V^*. Equation (3.2) can be solved by applying the Newton-based solver.

As commonly seen in the financial markets, option contracts vary because of different maturities, strike prices or types of option. The problem of solving equation (3.1) or (3.2) will change by inputting different parameters. These factors also affect the terminal and boundary conditions as well, more precisely,

- Different types of option change the terminal condition from payoff function,
- Different maturities T_i change the boundary condition and the grid size Δt,

- Different strike prices K_i change the terminal condition from payoff function,
- Different volatilities $(\sigma_0)_i$ change the entries of the tri-diagonal matrix H,
- Different interest rates r_i change the entries of the tri-diagonal matrix H.

3.2.1 A Batched Newton-based Solver

If a computer system receives a lot of requirements taken from the possible choices above, for example, calculating option prices at different maturities, then it will be necessary to solve a set of independent nonlinear Black-Scholes equations leading to the solution of a set of nonlinear systems, i.e.

$$G_l(V_l^*) = H_l(V_l^*)V_l^* - f_l = 0,\ l = 1,...,L, \tag{3.3}$$

where L represents the problem size. These nonlinear systems are independent and can be solved in parallel. The idea of batch operation is to assemble all the nonlinear systems into one big one, namely

$$\mathbf{G}(\mathbf{V}^*) = \mathbf{H}(\mathbf{V}^*)\mathbf{V}^* - \mathbf{F} = 0, \tag{3.4}$$

where $\mathbf{G} = (G_1,...,G_L)^T$, $\mathbf{F} = (f_1,...,f_L)^T$, $\mathbf{V}^* = (V_1^*,...,V_L^*)^T$ and $\mathbf{H}$ is constructed from $H_1,...,H_L$. Fig 3.1 shows an example of constructing $\mathbf{H}$ with $L = 3$. Some entries are filled with zeros to preserve the structure of a tri-diagonal matrix.

Note that for different nonlinear systems of $\mathbf{G}(\mathbf{V}^*)$, the stopping condition of the iterative scheme depends on $d_l,\ l = 1,...,L$, where

$$d_l = \frac{\|G_l\|}{\max(\|V_l^*\|, 1)},$$

is calculated from the root-finding function, or alternatively

$$d_l = \frac{\|res_l\|}{\max(\|V_l^*\|, 1)},$$

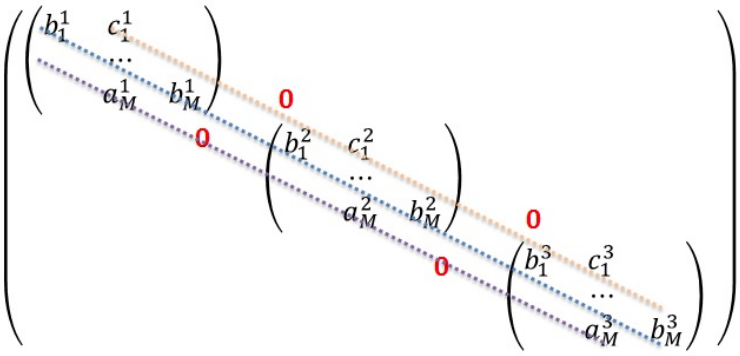

Fig. 3.1 Example of assembling three matrices.

is chosen from the updating residual, here

$$res_l = -\left[Jac(G_l(V_l^*))\right]^{-1} G_l(V_l^*).$$

To guarantee all the nonlinear systems converge with respect to a given tolerance tol, it is necessary to check

$$d_l < tol,\ l = 1, ..., L.$$

Checking these different conditions one by one may be cumbersome and it is worthwhile to consider a total norm from the batched system like

$$d = \sqrt{d_1^2 + ... + d_L^2}, \tag{3.5}$$

which implies immediately that

$$d^2 > d_l^2,\ l = 1, ..., L,$$

and therefore by checking the condition

$$d < tol, \tag{3.6}$$

it is sufficient to guarantee that all the $d_l, l = 1, ..., L$, satisfy the stopping condition and ensure the convergent of the iterative scheme for solving the batched nonlinear system (3.4).

Algorithm 4 presents the algorithm for constructing a batched Newton-based solver with problem size L. It can be viewed as assembling different numerical linear algebra computations from independent equations together and then doing the calculation once with a much larger size. This setting provides enough work for keeping the GPUs diligent when doing the parallel computing, which is introduced in the next section. It is important to notice that the batched matrix $\mathbf{H}$ has to be tri-diagonal, and so it is necessary to fill in zero values to some entries.

Algorithm 4: Batched Newton-based Solver

Input: terminal conditions $V_l^{N_t-1} = V_l(S, t = T)$, initial guesses V_l^*, for $l = 1, ..., L$, tol,
Output: $V_l^0 = V_l(S, t = 0)$ for $l = 1, ..., L$
for $n = N_t - 1 : 1$ **do**
  Initialise the batched vector $\mathbf{V}^* = (V_1^*, ..., V_L^*)^T$;
  for $k = 1 : N_{ite}$ **do**
    1. Get the batched matrix $\mathbf{H}$ and vector $\mathbf{F}$ by $\mathbf{V}^*$;
    2. Calculate $\mathbf{G}$ by equation (3.2) ;
    3. Calculate d ;
    if $d < tol$ **then**
      $\mathbf{V}^{n-1} = \mathbf{V}^*$, break;
      else
        1. Calculate **res** $= -[Jac(\mathbf{G}(\mathbf{V}^*))]^{-1}\mathbf{G}(\mathbf{V}^*)$;
        2. $\mathbf{V}^* = \mathbf{V}^* +$ **res**, go back to 1.;
      end
    end
  end
end

3.3 GPU Computing Implementations

3.3.1 The GPU Architecture

Graphics processing units (GPUs) are devices incorporated into many computers or laptops for creating images efficiently and displaying them. The rapid development and evolution of GPUs for general purpose programming has allowed researchers to accelerate computations in different areas. In the following a brief introduction to the GPU architecture is given and further details of the architecture from the latest graphic card in NVIDIA can be found in [90].

Generally speaking, a GPU consists of a large number of stream multiprocessors (SM), and each of them with a variable number of stream processors (SP) that can execute computation in parallel. This can be of benefit to computational problems that can be expressed as data parallel computing due to its massively parallel architecture (see Figure 3.3). It also contains a hierarchical memory called global memory which is writable from both the GPU and its CPU, plus a hierarchy of memories such as shared memory, constant memory, texture memory and registers.

There are three metrics which can be considered to measure performance when doing GPU computing. The first one is to compute the elapsed time from the application using the parallel algorithm with CPU and GPU, and then compare with using the sequential algorithm with only CPU to get the speedups. This is a standard comparison to have an idea of how much time can be saved when doing GPU computing.

The second one is the memory bandwidth, which usually is expressed in GB/s and shows the rate of writing and reading data. The memory bandwidth can be evaluated under the theoretical sense which usually called peak bandwidth, or with the sense of effective memory bandwidth which is derived from the computation. Note that measuring the memory bandwidth relies on the computation time, for example, the effective memory bandwidth can

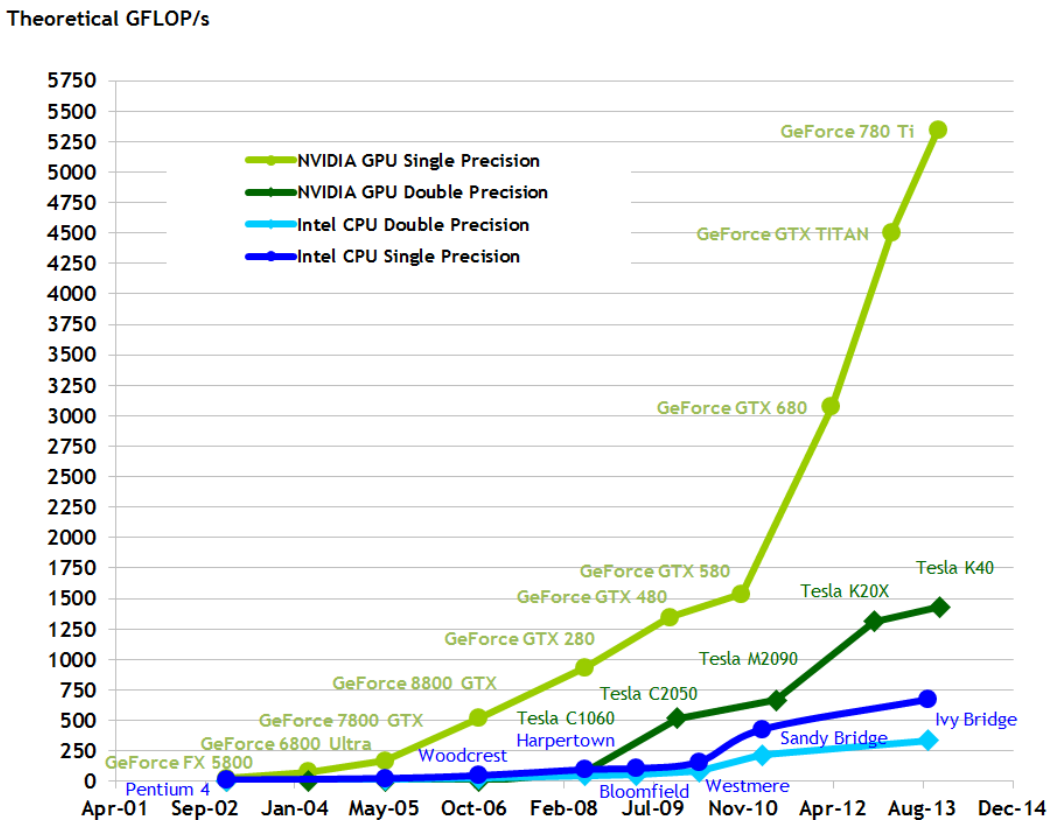

Fig. 3.2 Floating-Point Operations per Second for the CPU and GPU. The figure is from [88].

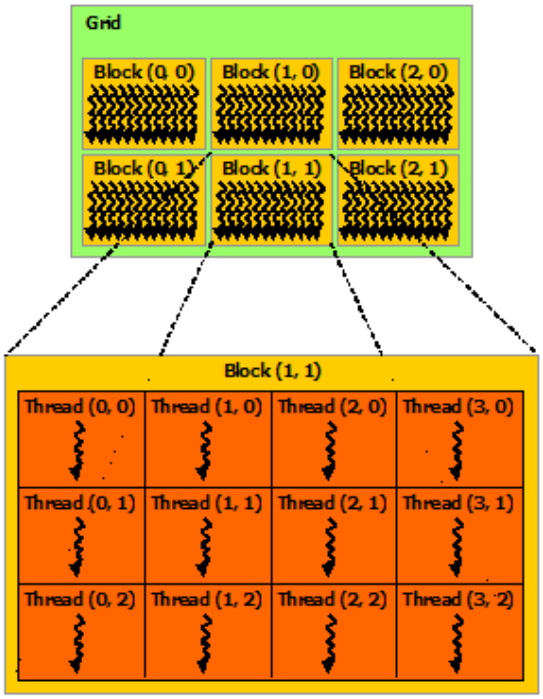

Fig. 3.3 Grid of Thread Blocks. The figure is from [88].

Fig. 3.4 Nvidia Kepler GPU hardware structure. The figure is from [91]

be calculated as

$$(R_{byte} + W_{byte})/(time \times 10^9),$$

where R_{byte} means the number of bytes with reading application, and W_{byte} represents the number of bytes with writing application.

The third one is the computational throughput which is measured with GFLOP/s (Giga-FLoating-point OPerations per second). It indicates how many operations from additions and multiplications are done per second in a given computation. Figure 3.2 shows that the GPU has a much higher theoretical throughput than CPU. It also depends on getting the computation time first in order to evaluate the throughput as following

$$N_{operation}/(time \times 10^9).$$

Table 3.1 An example of using matrix transposition to test the effective memory bandwidth of using single GPU.

N	effective bandwidth	theoretical bandwidth	percentage
4096	145.18 GB/s	240 GB/s	60.4%
8192	145.13 GB/s	240 GB/s	60.4%
16384	139.21 GB/s	240 GB/s	58.0%

.

The graphic card used in this thesis is a NVIDIA Tesla K80 which is popular for accelerating scientific computations. It uses the Kepler architecture (see Figure 3.4) having much greater throughput, higher compute capability, and larger number of registers than the previous Fermi architecture and also many additional innovations as explained in [91]. One other feature is the K80 can be seen - there are two separate GPUs on the same board and thus doing multi-GPU computing is possible. All numerical experiments are done with the workstation from Amazon EC2 with the P2 Instance. The example provided here is to examine the effective memory bandwidth by doing a matrix transposition with a matrix size $N \times N$, and the computational throughput by doing a matrix-multiplication with two $N \times N$ matrices in double precision. In such case it is clear that

$$(R_{byte} + W_{byte}) = 2 \times N \times N \times S_{double},$$

where S_{double} means the size of the data in double precision, and also

$$N_{operation} = N^3.$$

Table 3.1 and Table 3.2 show the results run on the Tesla K80 machine using a single GPU and it can be observed that it is possible to reach 60% of the theoretical memory bandwidth and 70% of the throughput.

Table 3.2 An example of using matrix mutiplication to test the computational throughput of using single GPU.

N	computational throughput	theoretical throughput	percentage
4096	947.449 Gflop/s	1455 Gflop/s	65.1%
8192	1053.57 Gflop/s	1455 Gflop/s	72.4%
16384	1056.47 Gflop/s	1455 Gflop/s	72.6%

The memory bandwidth and computational throughput metrics give ideas about whether the calculation are either memory bound or computation bound. This is quite important for doing performance optimisations. For complicated algorithms this may require tools to help evaluate these metrics; these are introduced in the next sections.

3.3.2 CUDA Programming

To employ the GPUs by programming, one must cope with multiple threads which can be executed at the same time. The threads are the basic units of the assigned executions which are grouped into blocks, and the blocks are grouped into grids as shown in Figure 3.3. One standard way of performing actions with data on these threads and blocks is to use the Compute Unified Device Architecture (CUDA), which is an extension of the C language.

When running a computer program to do scientific computation, normally some specific functions are executed to give instructions on the threads and blocks in parallel. These functions are called kernel functions which are programmed in CUDA. Kernel functions are essentially manipulating the computations on device (or GPUs) and have to be allocated memory suitable to the purpose. For example, the constant and texture memory are only read-only and limited. The registers contain the smallest but the fastest memory. Global memory is accessible by all threads but its performance is the slowest. It may happen that if the memory allocation is not managed well, then the results can only obtain low speedups even if there is enough parallelism in the computation.

CUDA also contains several useful APIs for doing data transfer or measuring performance. For examples:

- *cudaMemcpy*
- *checkCudaErrors*
- *cudaEvent_t*
- *nvprof*, *nvvp*

cudaMemcpy permits synchronous data transfers between host and device which means the transfers start when CUDA has finished all other executions called previously. Also, other executions from CUDA will not start until the transfers have finished. It is crucial to have the fewest operations for transferring data and minimise the transferred size which can help to optimise the performance. *checkCudaErrors* can report different fundamental issues from programming and is usually applied with *cudaMemcpy* to secure a safe data transfer.

As mentioned computation time is an important metric in which to measure performance. Normally it can be done by using the standard *clock_t* in C code. CUDA provides another specific timer *cudaEvent_t* which records the start and stop time of the relevant execution and returns the elapsed time between them in milliseconds. The timer is also able to record different events in one execution, useful for the complexity analysis of the algorithm. The profiler *nvprof* enables a deeper understanding of the data movements when running the code. It also contains many useful functions to count the number of operations which are important for measuring the bandwidth and throughputs. The visual profiler *nvvp* has a graphical interface and can display a timeline of the works from CPUs and GPUs which can help to realise and optimise the performance. All these mentioned APIs are used in the numerical experiments and in the next sections some further specific CUDA libraries are introduced and applied as well.

3.3.3 A Parallel Batched Newton-based Solver

The main purpose in this chapter is to solve large-scale nonlinear option pricing problems, in which the batched Newton-based solver constructed in Sec. 3.2.1 assembles all the problems together. As explained in Algorithm 4, the solver is decomposed into different basic linear algebra computations where the parallelism is well known and many existing tools are able to applied. More precisely, the computations below are parallelised

- Batch operation on the matrix $\mathbf{H}$ and evaluation of the root-finding function $\mathbf{G}$,
- Tri-diagonal solver for calculating the inversion of Jacobian matrix $[Jac(\mathbf{G})]^{-1}$,
- Calculating the l_2 norm for checking the stopping condition,
- Vector addition for updating $\mathbf{V}^* = \mathbf{V}^* + \mathbf{res}$.

This decomposition gives an insight on how to calculate the memory bandwidth and throughput for which the number of operations can be estimated by computations mentioned above. For example, when evaluating the volatility function, Gamma is required, namely

$$V_{SS} = \frac{V^*_{i-1} - 2V^*_i + V^*_{i+1}}{(\Delta S)^2},$$

which needs two additions and three multiplications for each i. Similarly when evaluating

$$\sigma_i = \sigma_0 \left(1 - \rho S_i V_{SS}\right)^{-1},$$

there are one addition and four multiplications. Some coefficients, which are repeatedly used, can be simplified as

$$C_1 = \frac{\sigma_i^2 S_i^2}{(\Delta S)^2},\ C_2 = \frac{r S_i}{\Delta S},$$

where for C_1 it needs five multiplications, and for C_2 two multiplications. The entries of tri-diagonal matrix H can then be obtained by

$$a_i = \frac{-\Delta t}{2}\left(C_1 - C_2\right),$$

$$b_i = 1 + \Delta t\,(C_1 + r)\,,$$

$$c_i = \frac{-\Delta t}{2}\,(C_1 + C_2)\,,$$

when using the fully implicit scheme and a_i, b_i, c_i all contain three operations for each i.

Now consider the batch size is equal to L, N_S, N_t are the grid points employed for the spatial and temporal discretisations, and N_{ite} is the an average number of iterations for making the Newton-based solver converge. The number of operations for the tri-diagonal solver is approximated by the operation count of the Thomas algorithm which is $O(N_S)$. Then, summing up all the operations needed in Algorithm 4 with the formulas provided in Sec. 2.4.2, the computational throughput, is evaluated by the formula

$$N_{operation}/(time \times 10^9),$$

where

$$N_{operation} = 68 \times (N_S - 2) \times N_{ite} \times (N_t - 1) \times L.$$

Following a similar idea, in Algorithm 4, the data storage involved the known value V^n, initial guess V^*, the entries of the tri-diagonal matrix H, the entries of the tri-diagonal Jacobian matrix Jac, the right hand side value f, the root-finding function G, and also the residue vector *res* for doing the updating. Hence the effective memory bandwidth is

$$2 \times 11 \times (N_S - 2) \times N_{ite} \times (N_t - 1) \times L \times S_{double}/(time \times 10^9),$$

again here S_{double} represents the size of the data in double precision.

The calculations above show the metrics for measuring the performance of the computation. As mentioned the paralleled parts are common linear algebra calculations, and there exist sophisticated GPU-accelerated libraries from CUDA Toolkit [88] that can obtain reasonable performance as the codes are highly-optimised and can be directly used by just

calling the functions. For examples, the CUDA sparse matrix library (cuSPARSE) [89] and the CUDA Basic Linear Algebra Subroutine library (cuBLAS) [87] contain many useful functions for parallel numerical linear algebra computation. A brief explanation of the specific functions used in this chapter is given as follows (for more details or examples see [18, 103]).

gtsv,gtsv2

An useful parallel tri-diagonal solver called ***gtsv*** exists in the cuSPARSE library. This solver uses the SPIKE algorithm [15]. The inputs are the three arrays of the tri-diagonal matrix, and one array from the right hand side vector, all of which are necessarily allocated on device for executing the function. Note that the output is the solution as an array on device. The ***gtsv*** solver supports both single and double precision. Figure 3.5 shows an example of solving a given tri-diagonal linear system by using the ***gtsv*** solver and using the Thomas algorithm implemented with CPU. It can be observed that using the library solver performs faster as the size of matrix grows. A drawback appears when using the ***gtsv*** solver is that when calling the solver, it automatically does an initialisation each time to allocate suitable memory which can ruin the performance. Because in the constructed Newton-based solver, when the number of grid points are given, the size of matrix is fixed which means it is only necessary to initialise the matrix once. The new ***gtsv2*** solver released in CUDA9.0 remedies this issue since it is possible to give a buffer size as an input to avoid the repeated initialisations and it can perform much faster.

nrm2

As addressed in Sec. 3.2.1, this is able to guarantee the batched Newton-based solver convergence by checking the stopping condition in equation (3.5). The term d can be obtained by calculating the norm of the batched vector **G** or **res**, using the function ***nrm2*** from the cuBLAS library. The input is the array on device and the output is the l_2 norm on host which

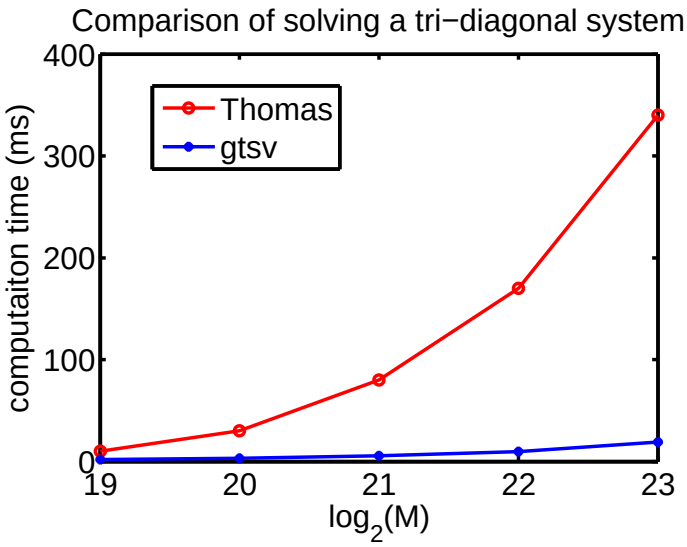

Fig. 3.5 Comparison of solving a $(M \times M)$ tri-diagonal linear system by using the gtsv solver from the cuSparse library and the Thomas algorithm implemented on the CPU with double precision.

can be verified directly to see whether it is smaller than a given tolerance. The function ***nrm2*** calculates the norm very fast even with a large size of array, and it shows the benefits of efficiency rather than calculating different norms d_l, $l = 1, ..., L$, and checking the stopping conditions many times.

axpy

The function ***axpy*** from the cuBLAS library is helpful when updating the residual. Typically the updating is a simple vector addition even in the batched Newton-based solver, as the initial guess $\mathbf{V}^*$ is decomposed to V_l^*, $l = 1, ..., L$. The function ***axpy*** presents the addition of the form

$$a \times x + y,$$

where a is scalar which is chosen to be 1 when doing the updating. x and y are both vectors which are replaced by the initial guess $\mathbf{V}^*$ and the calculated residue **res**.

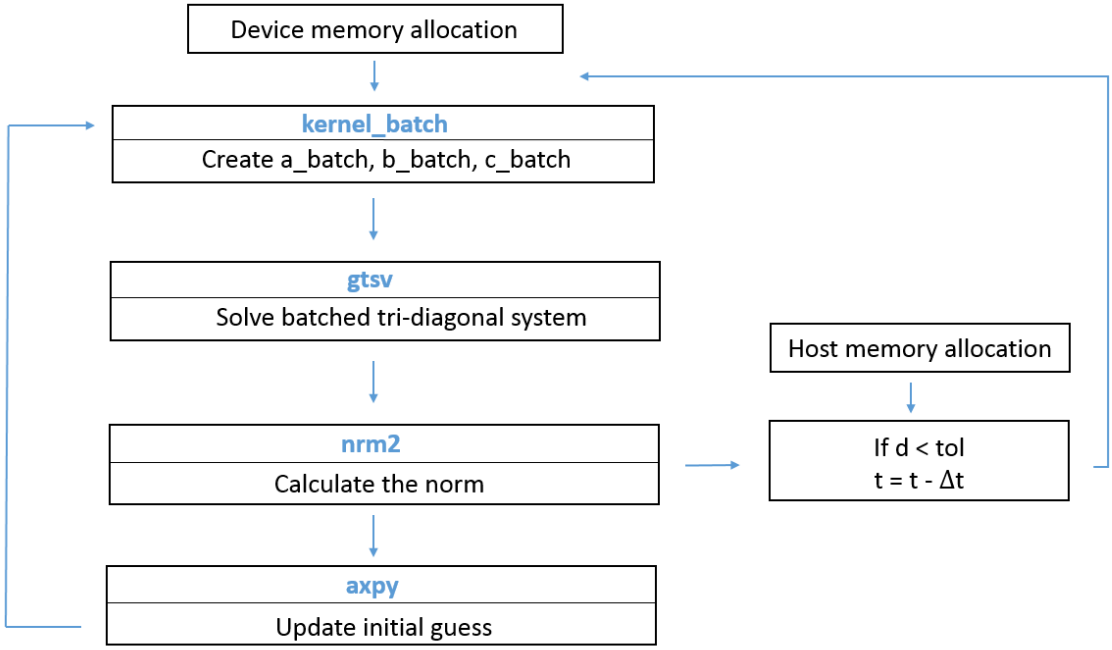

Fig. 3.6 A framework of the GPU implementation.

The final task is to construct the batched matrix consisting of evaluations of the entries with given explicit formula. These evaluations are all independent as the entries may differ from the chosen discretised point and can be done in parallel with global memory. It is important to note that CUDA organises execution on device using groups of 32 threads, which is normally called a warp. The threads in the same warp are run on the same stream multiprocessor. To improve performance, the allocation of the global memory when loading the data needs to be coalesced which means the threads have to access consecutive global memory within a half-warp (16 threads). Fig. 3.6 shows the framework for implementing the parallel batched Newton-based solver by using all the introduced functions.

3.3.4 OpenACC

As an alternative to CUDA programming, there exists an easier platform called OpenACC (Open Accelerators) which is able to implement GPU computing and becoming increasingly popular. OpenACC is commercial software which relies on a specific PGI compiler. The threshold for beginners is lower as it only requires users to specify the core part of

loops that need to be parallelised. The embedded compiler will analyse and construct a parallel equivalent in the executable code for execution. It can be applied to complicated algorithms, e.g. solving problems in computational fluid dynamics, as shown in [74]. This platform may not achieve as good a performance as CUDA programming, however, it can obtain computation or simulation results without spending too much time on coding and optimising the code which can allow the users more time for the scientific part of the research.

The most important task for using OpenACC is to give sufficient instructions for the compiler to examine and work. For example, the main task is to identify the core parts to be parallelised using the following commands:

```
#pragma acc parallel loop
{
Code to be parallelised ;
}
```

and this is applied to most of the for loops in Algorithm 4 except the loop for time-marching scheme as this loop has to be implemented sequentially.

If the type of computation is known precisely inside the loop, for example, like reduction for calculating the norm, then one can give more specific commands to increase the speedup as:

```
#pragma acc loop reduction
{
Reduction code to be parallelised ;
}
```

A challenge of using OpenACC is when doing the GPU computing, the compiler examines the loops and decides how to allocate the memory and transmit the data automatically which may incur some unnecessary data transmission. In order to manage the data movement well, it is important to add commands to address which data is transmitted from host to device, or from device to host, or where the data has been kept on device to use as follows:

```
#pragma acc data copyin(a[]) copyout(b[]) present(c[])
```

```
{
Code to be parallelised ;
}
```

which is quite important in the time-marching scheme as some arrays created on device need not necessarily copied to host.

In general OpenACC is useful when a parallel computing implementation is required with a tight deadline. In the next section several numerical experiments are described which were designed to examine the performance.

3.4 Numerical Experiments

The numerical experiment provided here was the solution of the European call option where the terminal and boundary conditions were taken as

$$\begin{cases} V(S,T) = (S-K)^+, \quad \text{for} \quad 0 \leq S < S_{max} \\ V(0,t) = 0, \quad \text{for} \quad 0 \leq t \leq T \\ V(S,t) = S - Ke^{-r(T-t)}, \quad \text{when} \quad S = S_{max} \end{cases}$$

and some basic parameters were fixed as $\sigma_0 = 0.4, K = 100, r = 0.03, S_{min} = 0, S_{max} = 300, T = 1/12$. The tolerance for Newton's iteration was chosen to be 10^{-4} for single precision and 10^{-8} for double precision. The grid points were fixed to be $N_S = N_t = 1024$. The large-scale problems were chosen to be with different σ_0 which are

$$(\sigma_0)_i = \sigma_0 - 0.0001 \times i, \; i = 0, ..., L-1,$$

here L represents the size of the problem.

3.4.1 Comparisons of CUDA and OpenACC

In the first experiment, the aim is to compare the performance of using CUDA and OpenACC. For a fair comparison the implementations were done on the same machine and the graphic card used is Quadro K2100M which has a compute capability equal to 3.0. For the hardware information, the CPU processor is : Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz with 4096 MB. The compilers for C code is gcc 4.7, for CUDA is CUDA7.5 and for OpenACC is PGI 16.4.

Table 3.3 Computation time (s) for single precision.

#Options	CPU	OpenACC	CUDA
$L = 64$	16.0	5.91	7.66
$L = 128$	32.5	11.6	8.77
$L = 256$	64.8	23.0	11.6

Table 3.4 Speedup for single precision.

#Options	CPU	OpenACC	CUDA
$L = 64$	1.0x	2.7x	2.0x
$L = 128$	1.0x	2.8x	3.7x
$L = 256$	1.0x	2.8x	5.5x

Table 3.5 Computation time (s) for double precision.

#Options	CPU	OpenACC	CUDA
$L = 64$	24.8	12.2	10.9
$L = 128$	49.7	24.1	13.0
$L = 256$	122	65.5	17.0

Table 3.6 Speedup for double precision.

#Options	CPU	OpenACC	CUDA
$L = 64$	1.0x	2.0x	2.2x
$L = 128$	1.0x	2.1x	3.8x
$L = 256$	1.0x	1.9x	7.2x

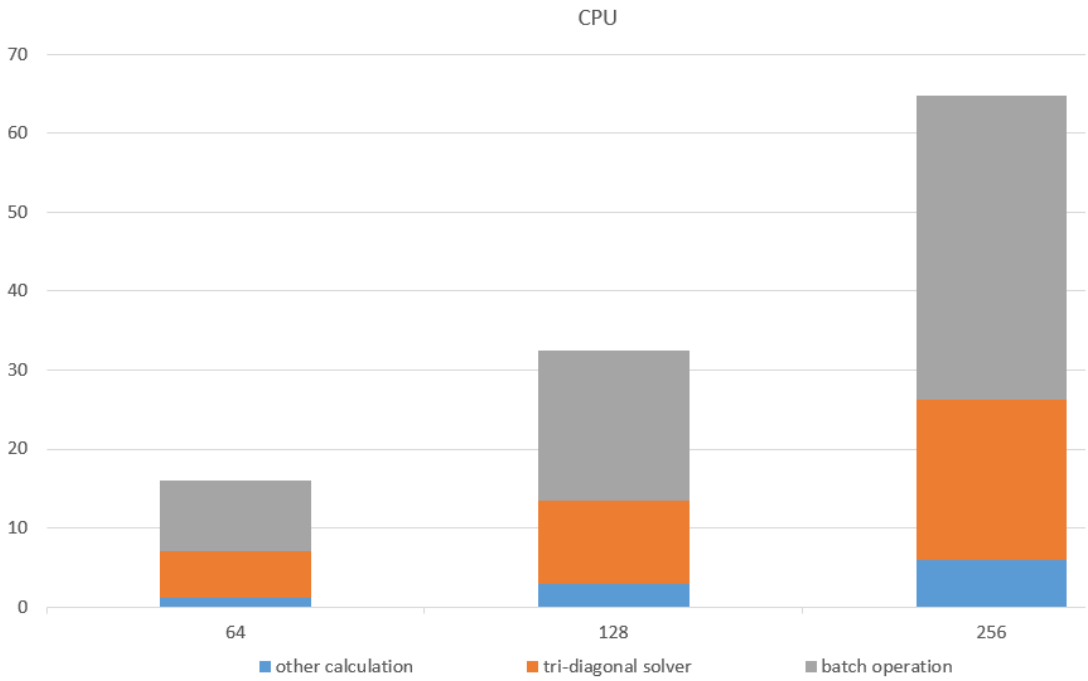

Fig. 3.7 Complexity analysis for single precision calculating by CPU.

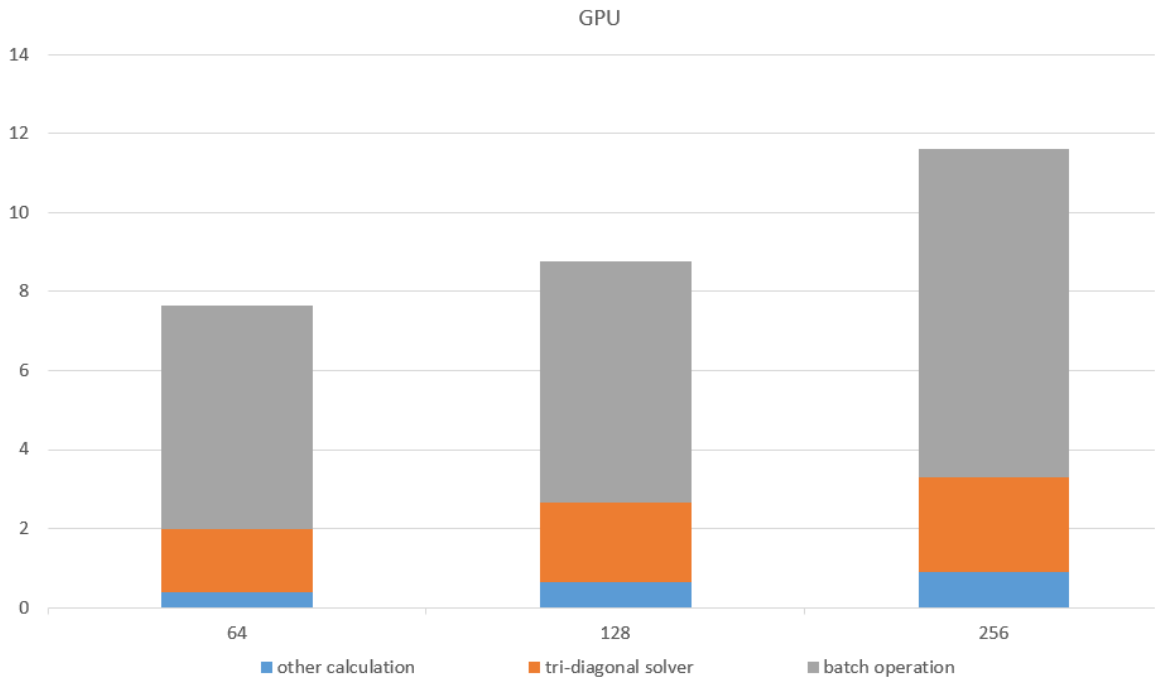

Fig. 3.8 Complexity analysis for single precision calculating by GPU.

Table 3.3 and 3.4 show the total computation time and speedups with different implementations for single precision. Similarly Table 3.5 and 3.6 show the results for double precision. It can be observed that by using OpenACC, it is possible to obtain a speed up of 2 by adding only few lines of codes. However the speedup doesn't increase when the size of problem grows due to the data movement becoming more complicated and requiring more sophisticated techniques to control it.

Using CUDA library and kernel functions shows better performance and the speedup increases when the size of problem is getting larger. The implementation takes more time in double precision than single because the main computations are memory bound and calculation in double precision takes more time to read and write the data. Therefore avoiding too much data transmission or usage is important when handling such problems in double precision.

Fig. 3.7 and 3.8 also show the individual computation times of each large-scale problem. For both using CPU and GPU, constructing the necessary tri-diagonal matrix is the time-consuming part where using GPU computing takes the advantage of performing these calculations with a large number of threads instead of evaluating them sequentially. The parallel tri-diagonal solver also shows benefits when the size of batched matrix is large. For calculating the norm and doing the updating, using GPU gives also better results, however this time is less important than other parts so the results are combined into the other calculation part.

3.4.2 Numerical Results with K80

The second numerical experiment was designed to solve the same problem stated in Sec 3.4 using a Tesla K80 GPU which has a compute capability 3.7. Note that in this experiment only one GPU was used for the calculations. The CPU hardware is as Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz. The compiler for the C code is gcc 4.7, for CUDA is CUDA9.0. In

Table 3.7 Results of using CPU and GPU for double precision.

#Options	CPU time	GPU time	speedup	effective bandwidth	computational throughput
$L = 512$	178.9	8.780	20.37x	53.63 GB/s	41.44 Gflops/s
$L = 1024$	381.9	12.45	30.67x	75.67 GB/s	58.46 Gflops/s
$L = 2048$	801.3	25.21	31.78x	74.73 GB/s	53.28 Gflops/s

Table 3.8 Profiling results of using GPU for double precision.

#Options	GPU time	batch	gtsv2	others
$L = 512$	8.780	4.94 (56,2%)	2.67(30.4%)	1.17 (13.3%)
$L = 1024$	12.45	7.68 (60.6%)	3.12 (25.0%)	1.65 (13.2%)
$L = 2048$	25.21	14.63 (58.0%)	6.88 (27.2%)	3.7 (14.6%)

this experiment the parallel tri-diagonal solver *gtsv2* was chosen to use.

Table 3.7 shows the results for speedups, effective memory bandwidth and the computational throughput when doing the computation in double precision. Table 3.8 presents the profiling results on the individual task. It can be seen that the results achieve around 20% of the theoretical memory bandwidth but very low throughput, as most of the computation, including using the libraries, is memory bound. One reason for this is in the batched Newton-based solver, the iterative scheme waits until all the stopping conditions for the assembled nonlinear systems are satisfied which results in storing more data. This can also be observed from the profiling results as the batch operation contributes the majority of the computation time. However, around 20 times speedup is still achievable when doing the calculation in double precision as the size of the batched matrix is large and the benefits of using GPU-based libraries can be seen.

3.5 Multi-GPU Computing

As mentioned before the NVIDIA Tesla K80 has two GPUs which can both be used. Therefore some further improvements to the performance from using multiple GPUs are considered. The idea is to split the tasks for different GPUs to work for at the same time. For the large-scale problems, since all the nonlinear PDEs are independent, an easier strategy is to do

the splitting for the number of problems divided by the number of GPUs. Fig 3.9 shows an example of using two GPUs where in each GPU, the implementation is exactly the same as the framework explained from Fig 3.6.

Different implementations of assigning each device to do the work in parallel are possible, and one easy way relies on using OpenMP where the compiler can allocate the tasks to the assigned device automatically by adding the commands

```
#pragma omp parallel for
for (int i=0; i<Num_device; i++){
    cudaSetDevice(i);
    //  main code to be run in each device
}
```

and in such case, device(0) is the first GPU running the assigned tasks and similarly for other devices.

Table 3.9 shows the computation times for a large-scale problem where $L = 4096$ using the CPU, one GPU and multiple GPUs on the Amazon EC2 workstation with P2 Instance (p2.8xlarge) which it allows to use a maximum 8 GPUs for doing computation. For the multi-GPU computing the problems are split by the number of GPUs, for example, if using two GPUs then for the first 2048 problems which are assigned to run on device 0 and the remaining 2048 problems on device 1. It shows that by using a single GPU, about 27 times speedup can be achieved. When adding the number of GPUs, a further speedup can be obtained although it seems not achieving the theoretical value. The reason why the performance is lower than expected may be from spending time on the process of splitting the work and also collecting the necessary data from different devices. These influences could be minimised possibly by controlling the data transmission to avoid too many transfers between different devices.

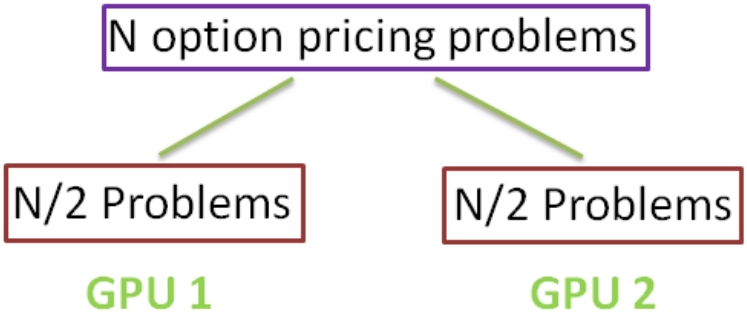

Fig. 3.9 An example of splitting the works for two GPUs.

Table 3.9 Results of using multiple GPUs for $L = 4096$.

	CPU	1 GPU	2 GPUs	4 GPUs	8 GPUs
time (s)	1562	57.983	36.467	22.79	16.27
speedup	1x	26.939x	45.31x	68.53x	96.00x

3.6 Summary

In this chapter, a batched Newton-based solver is constructed for dealing with large-scale nonlinear option pricing problems. The solver consists of different numerical linear algebra computations which are suitable for parallel computation. GPUs were chosen to be the implementation, and two possible platforms of doing CUDA programming or using OpenACC, are introduced. It can be observed that if the application doesn't require an huge speedup, but needs quick implementation, then using OpenACC is a good option as it can avoid spending much time on coding. If it is necessary to obtain higher levels of speedup, then doing CUDA programming and using CUDA libraries can achieve this goal. Further improvement using multiple GPUs is also introduced and demonstrated through an example showing using two GPUs.

Chapter 4

Asian Option Pricing

In this chapter, an extension of the Frey-Patie model from Vanilla type options to Asian option pricing is illustrated. The mathematical problem becomes that of solving a high-dimensional nonlinear PDE, using a merged numerical method consisting of a Newton-based solver and a semi-Lagrangian scheme. A range of numerical results are compared and some improvements of the developed solver are also discussed.

4.1 Introduction

Asian options are commonly seen in insurance markets where the price depends on an average of the underlying asset over time. It is usually cheaper than European or American option due to the average effect and more difficult to be manipulated as explained in [59]. In general there exists no analytic or closed form solution so numerical methods have to be employed to calculate the price. Kemna [70] used the Monte Carlo method for pricing such options, and a further study by Boyle [12] with different variance reduction techniques used the Quasi-Monte Carlo method. Rogers [101] considered an approach of reducing the problem to the solution of a parabolic PDE and provided a lower bound for the solution.

In addition to these techniques, an alternative way of pricing Asian option, as explained and derived in [107, 112] (also in Sec.4.2), is to solve the following PDE for $V(S,A,t)$

$$\frac{\partial V}{\partial t}+\frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}+rS\frac{\partial V}{\partial S}+\frac{(S-A)}{t}\frac{\partial V}{\partial A}-rV=0, \tag{4.1}$$

where S is the underlying asset, $r>0$ is the risk-free interest rate, σ is volatility of the underlying asset price process, and A is the asset price average with definition

$$A_t=\frac{1}{t}\int_0^t S_\tau d\tau. \tag{4.2}$$

Different volatility models have been studied in equation (4.1) as well as constant volatility, for examples, Parrott [95] used a stochastic volatility models with such PDE, and jump models were discussed by D'Halluin [25]. Tangman [109] has included further models such as Merton and Carr-German-Madan-Yor models. The uncertain volatility model was looked at by Fan [39] which results in the PDE becoming nonlinear.

As it can be observed that the PDE in equation (4.1) is two-dimensional and a variety of methods such as operator splitting or dimensional reduction have been explored. One modern technique is applying the semi-Lagrangian scheme, which was introduced by Parrott [94, 95] and further examined with different examples by D'Halluin in [25]. This technique has good potential for parallel computing as the problem becomes that of solving a large-scale set of one-dimensional PDEs, and Castillo *et al* [14] proposed an elaborate implementation of a parallel solver using GPUs which included a fast technique of doing interpolation using the GPU texture memory.

In the following sections, the focus is on constructing the semi-Lagrangian solver for Asian option pricing. To begin with the extension of the Frey-Patie model to Asian option pricing and the derivation of the fully nonlinear PDE is given. A semi-Lagrangian scheme is then merged with the Newton-based solver from Chapter 2 as the optimal numerical method to solve the high-dimensional fully nonlinear PDE. Different numerical experiments are

compared for examining the feasibility of using semi-Lagrangian scheme also some possible improvements are discussed.

4.2 Mathematical Modelling

Following the same idea given by [43] and also the derivation shown in [105], assume that the price of an underlying asset S_t is influenced by strategies of holding the asset from a large trader or amount of small traders using same strategy in an illiquid market, and it satisfies the process

$$dS_t = \mu_0 S_t dt + \sigma_0 S_t dW_t + \rho S_t d\alpha_t, \tag{4.3}$$

where W_t is the standard Brownian motion, μ_0 is the drift term, σ_0 is the volatility, ρ is the liquidity of the market, and α_t is the strategy function. Let $\alpha_t = \phi(S_t,t)$ by Itô's formula

$$d\alpha_t = \left(\frac{\partial \phi}{\partial t} + \frac{\sigma_0^2}{2}\frac{\partial^2 \phi}{\partial S^2}\right) dt + \frac{\partial \phi}{\partial S}, \tag{4.4}$$

so equation (4.3) can be replaced by

$$\left(1 - \rho S_t \frac{\partial \phi}{\partial S}\right) dS_t = \mu_0 S_t dt + \sigma_0 S_t dW_t + \rho S_t \left(\frac{\partial \phi}{\partial t} + \frac{\sigma_0^2}{2} S_t^2 \frac{\partial^2 \phi}{\partial S^2}\right) dt \tag{4.5}$$

which can be simplified to

$$dS_t = \mu(S_t,t) S_t dt + \sigma(S_t,t) S_t dW_t, \tag{4.6}$$

with

$$\mu(S,t) = \frac{1}{1 - \rho S \frac{\partial \phi}{\partial S}} \left(\mu_0 + \rho \left(\frac{\partial \phi}{\partial t} + \frac{\sigma_0^2}{2} S^2 \frac{\partial^2 \phi}{\partial S^2}\right)\right),$$

$$\sigma(S,t) = \frac{\sigma_0}{1 - \rho S \frac{\partial \phi}{\partial S}},$$

where the strategy function $\phi(S_t,t)$ will be chosen to be

$$\phi(S_t,t) = \frac{\partial V}{\partial S}$$

as explained in [43].

Now consider a portfolio

$$\Pi_t = V_t - \Delta_t^* S_t,$$

so that the total change in value is

$$d\Pi_t = dV_t - \Delta_t^* dS_t,$$

which is assumed to be hedgeable without any uncertainty and thus equal to the rate of return on riskless portfolio, namely

$$d\Pi_t = dV_t - \Delta_t^* dS_t = r(V_t - \Delta_t^* S_t)dt, \tag{4.7}$$

where r is the risk-free interest rate. Then by Itô's formula [107],

$$dV = \frac{\partial V}{\partial t}dt + \frac{\partial V}{\partial S}dS + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}(dS)^2 + \frac{\partial V}{\partial A}dA,$$

so that equation (4.7) becomes

$$\begin{aligned} & r(V_t - \Delta_t^* S_t)dt \\ = \; & \left(\frac{\partial V}{\partial t} + \mu(S,t)S_t\frac{\partial V}{\partial S} + \frac{\sigma^2(S,t)S_t^2}{2}\frac{\partial^2 V}{\partial S^2}\right)dt + \frac{\partial V}{\partial A}\frac{(S_t - A_t)}{t}dt + \frac{\partial V}{\partial S}S_t\sigma(S,t)dW_t \\ - \; & \Delta_t^*\left(\mu(S,t)S_t dt + \sigma(S,t)S_t dW_t\right). \end{aligned} \tag{4.8}$$

Equation (4.8) shows that Δ_t^* has to be chosen as

$$\Delta_t^* = \frac{\partial V}{\partial S}$$

to eliminate the term dW_t. Hence the following equation is obtained

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

which can be simplified to

$$\frac{\partial V}{\partial t}+\frac{1}{2}\sigma^2(S,t)S^2\frac{\partial^2 V}{\partial S^2}+rS\frac{\partial V}{\partial S}+\frac{(S-A)}{t}\frac{\partial V}{\partial A}-rV=0, \tag{4.9}$$

with

$$\sigma(S,t)=\frac{\sigma_0}{1-\rho S\frac{\partial^2 V}{\partial S^2}}=\sigma_0(1-\rho S V_{SS})^{-1}. \tag{4.10}$$

Note that when ρ equal to 0, equation (4.10) becomes the standard linear partial differential equation as stated in equation (4.1) which the solution is the price of the Asian option with constant volatility as discussed in [107, 112].

4.3 Semi-Lagrangian Scheme

To construct a nonlinear solver for equation (4.10), the aim is to merge the Newton-Raphson method, as shown in Chapter 2, with the semi-Lagrangian scheme. This method is denoted as the SL-Newton (Semi-Lagrangian Newton) solver and described as follows.

First, apply the semi-Lagrangian scheme to equation (4.9), the equation becomes

$$\frac{DV}{Dt}+\frac{1}{2}\sigma^2S^2\frac{\partial^2 V}{\partial S^2}+rS\frac{\partial V}{\partial S}-rV=0, \tag{4.11}$$

where

$$\frac{DA}{Dt}=\frac{1}{t}(S-A). \tag{4.12}$$

In order to use a finite difference discretisation, assume that the computational domain is truncated as $V(S,A,t)\in[0,S_{max}]\times[0,A_{max}]\times[0,T)$. The discretisation notation is defined

as

$$V_{i,j}^n = V(S_i, A_j, t^n) = V(i\Delta S, j\Delta A, n\Delta t),$$

where $i = 0, ..., N_S - 1$, $j = 0, ..., N_A - 1$ and $n = 0, ..., N_t - 1$. Here N_S, N_A, N_t are the numbers of grid points for discretisation on S, A, t respectively, and $\Delta S = S_{max}/(N_S - 1)$, $\Delta A = A_{max}/(N_A - 1)$, $\Delta t = T/(N_t - 1)$.

Applying the standard finite difference scheme of θ-method with backward timestepping, equation (4.11) can be discretised as

$$\frac{V_{i,j}^{n-1} - \tilde{V}_{i,j}^n}{\Delta t} = \theta \mathscr{L}(V^{n-1}) + (1-\theta)\mathscr{L}(\tilde{V}^n), \tag{4.13}$$

where

$$\mathscr{L}(V^{n-1}) = \frac{1}{2}\sigma^2 S_i^2 \frac{V_{i+1,j}^{n-1} - 2V_{i,j}^{n-1} + V_{i-1,j}^{n-1}}{(\Delta S)^2} + rS_i \frac{V_{i+1,j}^{n-1} - V_{i-1,j}^{n-1}}{2\Delta S} - rV_{i,j}^{n-1},$$

and the time integral path is along $(\tilde{A}_j, t^n)$ to (A_j, t^{n-1}) for $n = N_t - 1, ..., 1$. Here

$$\tilde{V}_{i,j}^n = V(S_i, \tilde{A}_j, t^n)$$

and $\tilde{A}_j$ can be formulated as stated in [94] by simply doing an integration of the equation (4.12)

$$\int_{A_j}^{\tilde{A}_j} \frac{1}{S_i - A}\,\mathrm{d}A = \int_{t^{n-1}}^{t^n} \frac{1}{t}\,\mathrm{d}t$$

which results the formula

$$\tilde{A}_j = S_i - \left(\frac{t^{n-1}}{t^n}\right)(S_i - A_j). \tag{4.14}$$

V^{n-1} is the required solution in the backward timestep and therefore equation (4.13) needs the solution of a nonlinear system. This can be formulated as a root-finding problem following the same idea as in Chapter 2 as:

$$G(V^{n-1}) = H(V^{n-1})V^{n-1} - \tilde{f} = 0, \tag{4.15}$$

where G is the root-finding function, H is the constructed tri-diagonal matrix and $\tilde{f}$ is the residual term put on the right hand side which can be evaluated from the known value V^n at the current time step t^n. The root-finding problem of equation (4.15) can be solved either by frozen coefficient method in Algorithm 5 or by Newton-Raphson method, shown in Algorithm 6. The Jacobian matrix can be evaluated using the same derived formula from Sec 2.4.2.

Algorithm 5: Frozen Coefficient Method

Input: Terminal condition $V^{N_t-1} = V(S,A,t=T)$, initial guess $V^* = V^{N_t-1}$, tolerance tol
Output: $V^0 = V(S,A,t=0)$
for $n = N_t - 1 : 1$ **do**
  for $j = 1 : N_A$ **do**
    for $k = 1 : N_{ite}$ **do**
      1. Calculate σ with V^* and construct tri-diagonal matrix H;
      2. Calculate $\tilde{A}_j$ by equation (4.14);
      3. Calculate $\tilde{V}^n_{i,j} = V(S_i, \tilde{A}_j, t^n)$ by doing interpolation for all $i = 1,...,N_S$;
      4. Calculate $\tilde{f}$ of equation (4.13) by $V(S_i, \tilde{A}_j, t^n)$ for all $i = 1,...,N_S$;
      5. Solve $V_{temp} = H \backslash \tilde{f}$;
      if $\|V_{temp} - V^*\| \leq tol$ **then**
        $V(S,A_j,t^{n-1}) = V^*$, break;
      end
      else
        $V^* = V_{temp}$ and go back to 1.;
      end
    end
  end
end

4.3.1 Terminal and Boundary Conditions

The solution $V(S,A,t)$ of the equation (4.9) is defined in the domain $D = [0,+\infty) \times [0,+\infty) \times [0,T)$ with the following possible terminal conditions

- $V(S,A,T) = \Phi(S,A,K) = \max(A-K,0)$ (fixed strike call),

Algorithm 6: Newton-Raphson Method

Input: Terminal condition $V^{N_t-1} = V(S,A,t=T)$, initial guess $V^* = V^{N_t-1}$, tolerance tol

Output: $V^0 = V(S,A,t=0)$

for $n = N_t - 1 : 1$ **do**
  for $j = 1 : N_A$ **do**
    for $k = 1 : N_{ite}$ **do**
      1. Calculate σ with V^* and construct tri-diagonal matrix H;
      2. Calculate $\tilde{A}_j$ by equation (4.14);
      3. Calculate $\tilde{V}^n_{i,j} = V(S_i,\tilde{A}_j,t^n)$ by doing interpolation for all $i = 1,...,N_S$;
      4. Calculate $\tilde{f}$ of equation (4.13) by $V(S_i,\tilde{A}_j,t^n)$ for all $i = 1,...,N_S$;
      5. Define the root-finding function $G(V^*) = H(V^*)V^* - \tilde{f} = 0$;
      6. Calculate the Jacobian matrix $Jac(G)$;
      7. Calculate the residue $res = -(Jac(G))^{-1}G$;
      if $\frac{\|res\|}{\max(\|V^*\|,1)} \leq tol$ **then**
        $V(S,A_j,t^{n-1}) = V^*$, break;
      end
      else
        $V^* = V^* + res$ and go back to **1.**;
      end
    end
  end
end

- $V(S,A,T) = \Phi(S,A,K) = \max(K-A,0)$ (fixed strike put),
- $V(S,A,T) = \Phi(S,A,K) = \max(S-A,0)$ (floating strike call),
- $V(S,A,T) = \Phi(S,A,K) = \max(A-S,0)$ (floating strike put),

where K is the strike price, and with the Neumann boundary condition

- $\lim_{S\to+\infty} \frac{\partial^2 V}{\partial S^2}(S,A,t) = 0.$

In fact the boundary condition of $S \to +\infty$ can be further analysed by following the same idea shown in [14, 25] which replaces the boundary condition in equation (4.9) and obtains

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{(S-A)}{t}\frac{\partial V}{\partial A} - rV = 0, \;\; S \to +\infty. \tag{4.16}$$

A linear solution can be obtained for equation (4.16) as

$$V(S,A,t) = C_1(\tau)S + C_2(\tau)A + C_3(\tau), \tag{4.17}$$

where $\tau = T - t$ means the time to maturity. Substituting V in equation (4.16) by (4.17) generates the nonhomogeneous system of differential equations as

$$\begin{aligned} C_1'(\tau) + \frac{1}{T-\tau}C_2(\tau) &= 0, \\ C_2'(\tau) + \left(r + \frac{1}{T-\tau}\right)C_2(\tau) &= 0, \\ C_3'(\tau) + rC_3(\tau) &= 0, \end{aligned}$$

and the solutions can be found explicitly as

$$\begin{aligned} C_1(\tau) &= k_1 - \frac{k_2}{r}e^{-r\tau}, \\ C_2(\tau) &= k_2(T-\tau)e^{-r\tau}, \\ C_3(\tau) &= k_3 e^{-r\tau}, \end{aligned}$$

where k_1, k_2, k_3 are determined from the terminal conditions. For example, for the fixed strike call case $V(S,A,t) = \max(A-K,0)$, the coefficients k_1, k_2, k_3 are obtained as

$$k_1 = 1, \;\; k_2 = r, \;\; k_3 = A - K - rTA.$$

Therefore, the solution for $A - K > 0$ and $S \to +\infty$ is given by

$$V_1(S,A,t) = \left(k_1 - \frac{k_2}{r}\right) e^{-r(T-t)} S + k_2 t e^{-r(T-t)} A + k_3 e^{-r(T-t)}, \tag{4.18}$$

and the asymptotic behaviour of the solution

$$V(S,A,t) = \max(V_1(S,A,t), 0), \;\; S \to +\infty \tag{4.19}$$

is taken as the boundary condition instead of using the second order condition.

4.3.2 Coordinate Stretching

The coordinate stretching technique used in 2.5 can also be considered in the present context when solving equation (4.11) which has been used in [94]. Following a similar idea as discussed in Sec. 2.5, the refinement of the grid points is expected to be acted around $S = K$ and $A = K$, then the stretched coordinates x_1, x_2 are defined as

$$S = \frac{K}{\lambda} \sinh(x_1 - L_S) + K,$$

$$A = \frac{K}{\lambda} \sinh(x_2 - L_S) + K,$$

or conversely

$$x_1 = \sinh^{-1}\left(\frac{\lambda}{K}(S-K)\right) + L_S,$$

$$x_2 = \sinh^{-1}\left(\frac{\lambda}{K}(A-K)\right) + L_S,$$

where L_S is the parameter controlling the stretching size, and $\lambda = \sinh(L_S)$.

Applying this coordinate stretching, equation (4.11) becomes

$$\frac{DV}{Dt} + \frac{1}{2}\sigma^2 (T_\lambda(x_1))^2 \frac{\partial^2 V}{\partial x_1^2} - \frac{1}{2}\sigma^2 \tanh(x_1 + L_S) T_\lambda(x_1) \frac{\partial V}{\partial x_1} + r T_\lambda(x_1) \frac{\partial V}{\partial x_1} - rV = 0, \quad (4.20)$$

where

$$T_\lambda(x_1) = \frac{\sinh(x_1 - L_S) + \lambda}{\cosh(x_1 - L_S)}.$$

4.4 Numerical Experiments

Several numerical experiments are described that illustrate the experimental order of convergence (*eoc*) under different circumstances, with the same definitions used as in Chapter 2. Let V_h be the solution evaluated with mesh sizes $h\Delta S, h\Delta A$ for the spatial discretisations in the S and A directions, and $h\Delta t$ for the temporal discretisation. Then the ratio of error is defined as:

$$R = \frac{\|V_h - V_{h/2}\|}{\|V_{h/2} - V_{h/4}\|}, \quad (4.21)$$

where the $eoc = \log_2(R)$. As explained in Chapter 2, an alternative choice is taking the solution evaluated with small mesh size to be the exact solution and then the ratio of error can be defined as:

$$\hat{R} = \frac{\|V_h - \hat{V}\|}{\|V_{h/2} - \hat{V}\|}, \quad (4.22)$$

where $\hat{V}$ is the solution evaluated with fine meshes and the $e\hat{o}c = \log_2(\hat{R})$.

The aim of the numerical experiments is to examine the *eoc* at the point $V(S = K, A = K, t = 0)$ with the above definitions. The interpolation method uses the cubic spline function in MATLAB. Certain parameters were fixed as $K = 100, S_{max} = A_{max} = 3K, T = 1, r =$

Table 4.1 Fully implicit scheme with $r = 0.01$, $\sigma_0 = 0.4$ and $\rho = 0$.

$N_S = N_A$	N_t	V	Err	R	eoc	$\hat{Err}$	$\hat{R}$	$\hat{eoc}$
61	31	9.4146	0.0391	1.42	0.50	0.0907	1.75	0.81
121	61	9.3754	0.0275	1.75	0.81	0.0516	2.14	1.10
241	121	9.3479	0.0157	1.88	0.91	0.0240	2.88	1.52
481	241	9.3322	0.0083			0.0083		
961	481	9.3238						

Table 4.2 Crank-Nicolson scheme with $r = 0.01$, $\sigma_0 = 0.4$ and $\rho = 0$.

$N_S = N_A$	N_t	V	Err	R	eoc	$\hat{Err}$	$\hat{R}$	$\hat{eoc}$
61	31	9.3876	0.0330	1.71	0.77	0.0646	2.04	1.03
121	61	9.3545	0.0193	2.10	1.07	0.0316	2.57	1.36
241	121	9.3352	0.0092	3.03	1.60	0.0122	4.03	2.01
481	241	9.3260	0.0030			0.0030		
961	481	9.3229						

$0.01, \sigma = 0.4$. The payoff function was chosen to be the fixed strike call, i.e. $V(S,A,T) = \Phi(S,A,K) = \max(A-K,0)$.

Table 4.1 and 4.2 show the results of using backward Euler scheme and Crank-Nicolson scheme for cauculating the option price $V(S = K, A = K, t = 0)$ when $\rho = 0$. Here N_S, N_A denote the number of grid points using on the spatial discretisations of S and A, and N_t represents the number of grid points on the temporal discretisation. For the fully implicit scheme the results tend to the expected value of *eoc* as the number of grid points become larger. For the Crank-Nicolson scheme, Rannacher timestepping method was applied to replace the first two time steps by the fully implicit scheme. The convergence behaviour does not behave well bacause of the non smooth terminal condition; similar results were presented in [25].

Table 4.3 shows the results of using Crank-Nicolson scheme with coordinate stretching as described in Sec. 4.3.2. Again the first two time steps of the Crank-Nicolson scheme were replaced by the fully implicit scheme. The improvements can be observed as the *eoc* values are close to 2 without using fine meshes.

Table 4.3 Crank-Nicolson scheme with $r = 0.01$, $\sigma_0 = 0.4$ and $\rho = 0$. The coordinate stretching technique is applied.

$N_S = N_A$	N_t	Err	R	eoc
31	16	0.0178	3.65	1.87
61	31	0.0048	4.04	2.01
121	61	0.0012		
241	121			

Table 4.4 Fully implicit scheme with $r = 0.01$, $\sigma_0 = 0.4$ and $\rho = 0.01$.

$N_S = N_A$	N_t	$V(S = K, A = K, t = 0)$	Err	R	eoc
50	10	1.2402	3.34e-3	0.35	-1.50
100	20	1.2369	9.45e-3	1.43	0.52
200	40	1.2274	6.60e-3	1.76	0.81
400	80	1.2208	3.75e-3	2.10	0.93
800	160	1.2171	1.78e-3	–	–
1600	320	1.2153	–	–	–

For the nonlinear volatility case, the liquidity factor was fixed at $\rho = 0.01$. Table 4.4 and 4.5 show the results of using the fully implicit scheme and the Crank-Nicolson scheme. The nonlinear system was solved by Newton-Raphson method since the solution doesn't converge well for the frozen coefficient method. For the backward Euler scheme the *eoc* approaches 1. For the Crank-Nicolson the Rannacher timestepping technique was again applied to replace the first two time steps with the fully implicit scheme. Second order accuracy is not seen; possible reasons are errors accumulated from the interpolation or in the evaluation of the nonlinearity.

Table 4.5 Crank-Nicolson scheme with $r = 0.01$, $\sigma_0 = 0.4$ and $\rho = 0.01$.

$N_S = N_A$	N_t	$V(S = K, A = K, t = 0)$	Err	R	eoc
50	10	1.2365	9.53e-3	2.15	1.11
100	20	1.2270	4.42e-3	2.39	1.26
200	40	1.2226	1.84e-3	8.49	-0.23
400	80	1.2207	2.17e-3	2.63	1.4
800	160	1.2186	8.27e-7	–	–
1600	320	1.2177	–	–	–

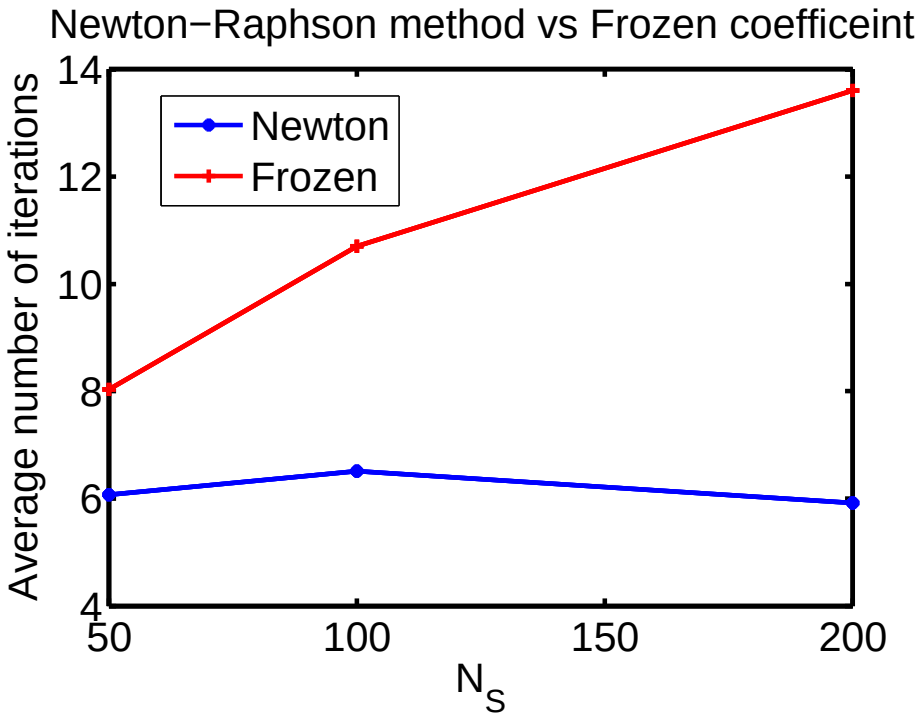

Fig. 4.1 The average number of iterations for Newton's method and Frozen coefficient for $N_S = 50, 100, 200$. The average is taken over all the discretisations on A-direction and all the time steps t^n.

Figure 4.1 gives a comparison of the number of iterations averaged over each discretise A-coordinate and all the time steps t^n. For Newton-Raphson method it requires around 6 iterations to converge, compared to the frozen coefficient method which needs more than double this number or even diverges. Figure 4.2 also emphasises the average number of iterations on each time step for Newton-Raphson method. It was found in practice that the most demanding iteration is for the time step t^1 which is the last step of the backward calculation (from V^1 to V^0) and therefore sometimes the damped updating for Newton-Raphson method has to be employed to guarantee the root-finding function G in equation (4.15) is monotone decreasing in order to avoid oscillation.

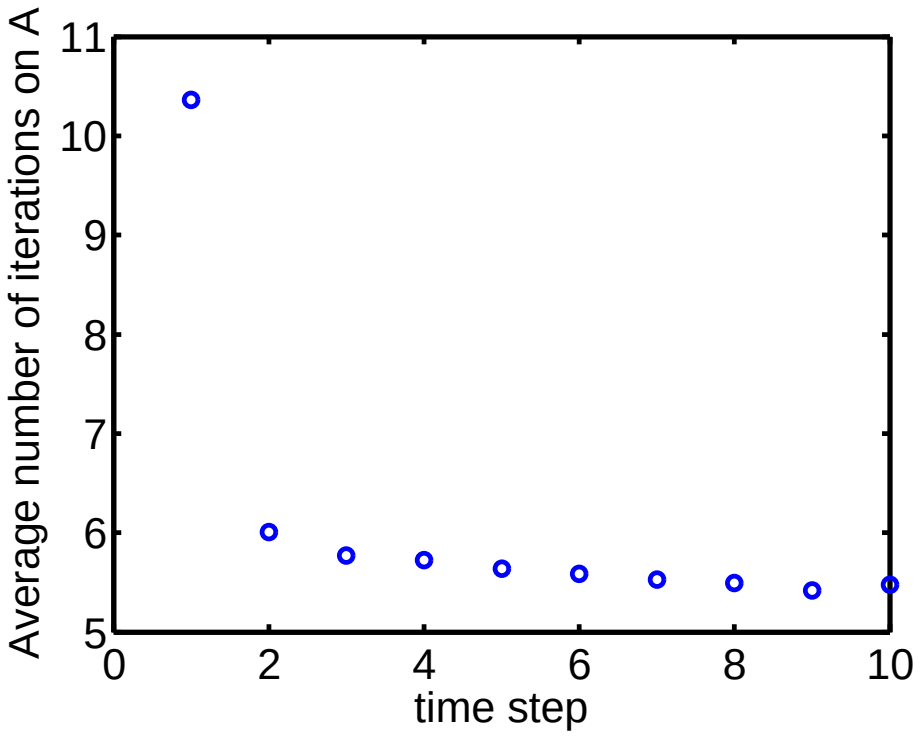

Fig. 4.2 The average number of iterations for Newton-Raphson method. The average is taken over all the discretisations on A-direction at each time step.

4.5 Summary

This chapter discusses the case of pricing Asian options in illiquid market with the Frey-Patie model. This requires the solution of a two-dimensional fully nonlinear PDE. The methodology of using a semi-Lagrangian scheme is followed leading to a SL-Newton solver. Essentially the SL-Newton solver provides the benefits of obtaining the Asian option price by solving a set of one-dimensional nonlinear PDEs. The cost optimal since after finite difference spatial discretisation, these one-dimentional nonlinear systems which emerge have matrices that are all tri-diagonal. Numerical results show that the second order accuracy is not easily obtained since it is affected by different sorts of errors in the SL-Newton solver when using the Crank-Nicolson scheme. However applying the fully implicit scheme in the SL-Newton solver does have the expected accuracy and may be a better choice when applying the solver.

Chapter 5

Multi-Asset Problems

In this chapter, the aim is to explore the multi-asset option pricing problem for illiquid markets which results in a multi-dimensional nonlinear Black-Scholes equation. A generalisation of the Frey-Patie model to the multi-asset case is proposed and the nonlinear PDEs are derived. Numerical solution using an Alternative Direction Implicit method merged with the Newton-based solver is constructed and explained. Several numerical experiments are discussed and compared to examine accuracy and efficiency.

5.1 Introduction

Multi-asset problems occur when the financial product depends on more than one underlying asset. Examples can be found from S&P 500, FTSE 100, options amongst others, which can be seen as a basket of stocks in a portfolio. These kinds of products protect investors against the market losses especially when the constructed portfolio is not sufficient diversified. An option that depends on multiple assets is called a multi-asset option and has different structures such as basket options, spread options, rainbow options, and so on.

These multi-asset options allow various payoff functions, for example, the spread call option which is widely used in commodity markets as discussed in [16] has the payoff

function for the two-asset case

$$V(S_1,S_2,T) = \Phi(S_1,S_2,K) = \max\left(S_1 - S_2 - K, 0\right), \tag{5.1}$$

where S_1, S_2 are the two underlying assets, T is the maturity and K is the strike price of the option. Another example is the arithmetic sum basket option, where the payoff function can be expressed as

$$V(S_1,S_2,T) = \Phi(S_1,S_2,K) = \max\left(Ave - K, 0\right), \tag{5.2}$$

where $Ave = a_1S_1 + a_2S_2$ and a_1, a_2 are the assigned weights for assets S_1, S_2, respectively. It is worthwhile noting that for some examples with simple payoff functions, explicit solutions or approximate formulae can be derived. For instance, the spread option with payoff function in equation (5.1) is normally approximated by using Kirk's formula shown in [71], and for the basket option with payoff function in equation (5.2) it is possible to have a proxy using the inverse gamma distribution as introduced in [85]. If the payoff function is

$$\Phi(S_1,S_2,K) = \max(S_1 - S_2, 0),$$

then the option price can be obtained by using Margrabe's formula proposed in [83]. However in general if the dynamics of underlying assets or the payoff functions are complicated, it is difficult to get exact solutions or approximate formulas and numerical solutions using approximations to PDE or Monte Carlo simulation are necessary.

In this thesis, the focus is on using a numerical PDE approach to solve the option pricing problems with multiple assets, which typically leads to a multi-dimensional Black-Scholes equation as discussed in [58], namely

$$\frac{\partial V}{\partial t} + \sum_{i=1}^{n} rS_i\frac{\partial V}{\partial S_i} + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\rho_{ij}\sigma_i\sigma_jS_iS_j\frac{\partial^2 V}{\partial S_i\partial S_j} - rV = 0, \tag{5.3}$$

where n represents the number of assets. When n is greater than 3, the PDE approach is not appropriate as it comes the curse of dimensionality. Typically equation (5.3) is solved when

$n = 2$ or $n = 3$ with the built numerical solvers.

Using a fully implicit scheme to solve equation (5.3) somehow is expensive especially when the number of grid points are becoming large. This is because the generated system to be solved at each time step is of an enormous size, and it is costly to store such matrices when doing computation either with dense or sparse structure. The Alternative Direction Implicit (ADI) method provides a technical way to reduce the dimensional cost when solving high-dimensional PDE. The idea was proposed many decades ago but is still commonly used or modified today. The first study was given by Peaceman and Rachford [96] which provided a natural way of splitting the high-dimeional PDE into different intermediate steps in which each step involves lower dimensional PDE. Further schemes which manipulate the splitting steps or involve special designs for different types of equations can be found in [19, 26–28, 60, 61, 110] . These schemes provide a good approximation of the original high-dimensional PDE but stability conditions may be required if cross-derivative term exists. In't Hout examined different schemes in [62, 63, 65, 66, 115] for the convection-diffusion equation with cross-derivative, and with applications especially to Heston type models in [55, 56, 64].

The advantage of applying an ADI scheme is solving reasonable matrix sizes. The other benefit is the set of lower dimensional PDEs can be solved in parallel as they can be generated independently and does not depend on each other. Dang [22] provided a framework for solving three-dimensional PDEs with cases like Rainbow and basket options by using GPUs. Egloff [32] gave also an example of doing GPU computing for solving a stochastic volatility model. These demonstrations highlight a promising direction of using GPUs to implement ADI solvers.

In the following sections, the focus is on pricing option on multiple assets in illiquid markets for which very few studies can be found in the literature. The idea of the Frey-Patie model is reviewed and a more generalised model for multi-asset is derived with the

corresponding multi-dimensional nonlinear Black-Scholes equation. A two-dimensional case is studied using the standard Peaceman-Rachford scheme with Newton-Raphason method and different numerical experiments are examined.

5.2 An Extension of the Frey-Patie Model

In Chapter 2, derivation of the nonlinear PDE for Frey-Patie model is given for the one-dimensional case. It explains how an option price is affected by the holding or hedging strategy on the single underlying asset taken by a large trader or groups of small traders in an illiquid market. When handling multi-asset problems, it is necessary to consider a more general model with different dynamics of the underlying assets which may influence each other. In the following the details of extending the Frey-Patie model to the multi-asset case is explained

Assume now the option price depends on multiple underlying asset $S_i,\ i = 1,...,n$. Then following the same idea as the one-dimensional case, assume the dynamic of S_i satisfies the stochastic process

$$dS_i = \tilde{\mu}_i S_i dt + \sum_{j=1}^{n} \tilde{\sigma}_{ij} S_j dW_j + \sum_{j=1}^{n} \tilde{\rho}_{ij} S_j d\alpha_j,\ i,j = 1,...,n, \tag{5.4}$$

where α_j are strategies for a large trader or a group of small traders holding stocks S_j and $\tilde{\rho}_{ij}$ are the liquidities for the underlying asset S_i with holding strategies α_j. Note when $n = 1$, the model is exactly the same one as discussed in Chapter 2 for the one-dimensional case.

To simplify the equation, consider $\mu_i = \tilde{\mu}_i S_i,\ \sigma_{ij} = \tilde{\sigma}_{ij} S_j,\ \rho_{ij} = \tilde{\rho}_{ij} S_j$ to replace the equation (5.4) to

$$dS_i = \mu_i dt + \sum_{j=1}^{n} \sigma_{ij} dW_j + \sum_{j=1}^{n} \rho_{ij} d\alpha_j,\ i,j = 1,...,n, \tag{5.5}$$

and the aim is to write equation (5.5) in the form

$$dS_i = b_i dt + \sum_{j=1}^{n} v_{ij} dW_j. \tag{5.6}$$

To start first consider the strategy of holding or hedging stocks is described by the functions

$$\alpha_i = \phi_i(S_1, S_2, ..., S_n, t).$$

Applying multi-dimensional Itô's lemma mentioned in [107],

$$d\alpha_i = \frac{\partial \phi_i}{\partial t} dt + \sum_{j=1}^{n} \frac{\partial \phi_i}{\partial S_j} dS_j + \frac{1}{2} \sum_{i,j=1}^{n} \frac{\partial^2 \phi_i}{\partial S_i \partial S_j} \sum_{k=1}^{n} \sigma_{ik} \sigma_{jk} dt, \tag{5.7}$$

here the fact $< dW_i, dW_j >= \delta_{ij} dt$ is applied, and combining equation (5.5) and (5.7) the following equation is obtained

$$dS_i = H_i dt + \sum_{j=1}^{n} \sigma_{ij} dW_j + \sum_{j=1}^{n} \rho_{ij} \sum_{k=1}^{n} \frac{\partial \phi_j}{\partial S_k} dS_k, \tag{5.8}$$

where

$$H_i = \mu_i + \sum_{j=1}^{n} \rho_{ij} \frac{\partial \phi_j}{\partial t} + \frac{1}{2} \sum_{p,q=1}^{n} \frac{\partial^2 \phi_j}{\partial S_p \partial S_q} \sum_{l=1}^{n} \sigma_{pl} \sigma_{ql}. \tag{5.9}$$

Equation (5.8) can be rewritten as

$$dS_i = H_i dt + \sum_{j=1}^{n} \sigma_{ij} dW_j + \sum_{k=1}^{n} \sum_{j=1}^{n} \rho_{ij} \frac{\partial \phi_j}{\partial S_k} dS_k, \tag{5.10}$$

which gives the following vector form

$$I d\vec{S} = \Sigma d\vec{W} + A d\vec{S} + \vec{H} dt. \tag{5.11}$$

Here the notations $d\vec{S} = (dS_1, dS_2, ..., dS_n)^T$, $d\vec{W} = (dW_1, dW_2, ..., dW_n)^T$, $\vec{H} = (H_1, H_2, ..., H_n)^T$ from (5.9), and $\Sigma = (\sigma_{ij})_{n \times n}$. I is an identity matrix, and A is a $(n \times n)$ matrix defined as

$$A = \begin{pmatrix} \sum_{j=1}^{n} \rho_{1j} \frac{\partial \phi_j}{\partial S_1} & \sum_{j=1}^{n} \rho_{1j} \frac{\partial \phi_j}{\partial S_2} & \cdots & \sum_{j=1}^{n} \rho_{1j} \frac{\partial \phi_j}{\partial S_n} \\ \sum_{j=1}^{n} \rho_{2j} \frac{\partial \phi_j}{\partial S_1} & \sum_{j=1}^{n} \rho_{2j} \frac{\partial \phi_j}{\partial S_2} & \cdots & \sum_{j=1}^{n} \rho_{2j} \frac{\partial \phi_j}{\partial S_n} \\ & \cdots & & \\ \sum_{j=1}^{n} \rho_{nj} \frac{\partial \phi_j}{\partial S_1} & \sum_{j=1}^{n} \rho_{nj} \frac{\partial \phi_j}{\partial S_2} & \cdots & \sum_{j=1}^{n} \rho_{nj} \frac{\partial \phi_j}{\partial S_n} \end{pmatrix},$$

which can be simplified to

$$A = \begin{pmatrix} \rho_{11} & \rho_{12} & \cdots & \rho_{1n} \\ \rho_{21} & \rho_{22} & \cdots & \rho_{2n} \\ & \cdots & & \\ \rho_{n1} & \rho_{n2} & \cdots & \rho_{nn} \end{pmatrix} \begin{pmatrix} \frac{\partial \phi_1}{\partial S_1} & \frac{\partial \phi_1}{\partial S_2} & \cdots & \frac{\partial \phi_1}{\partial S_n} \\ \frac{\partial \phi_2}{\partial S_1} & \frac{\partial \phi_2}{\partial S_2} & \cdots & \frac{\partial \phi_2}{\partial S_n} \\ & \cdots & & \\ \frac{\partial \phi_n}{\partial S_1} & \frac{\partial \phi_n}{\partial S_2} & \cdots & \frac{\partial \phi_n}{\partial S_n} \end{pmatrix} = \rho \times Jac(\phi),$$

where *Jac* is the Jacobian matrix.

Note equation (5.10) now becomes

$$(I - A)d\vec{S} = \Omega d\vec{W} + \vec{H}dt, \tag{5.12}$$

and after changing the measure to the risk-neutral one, the terms in equation (5.6) can be found to be

$$b_i = r,\ \Omega = (v_{ij})_{n\times n} = (I - A)^{-1}\Sigma.$$

Hence applying the multi-dimensional Feymann-Kac theorem from [107], a multi-dimensional nonlinear Black-Scholes equation under the Frey-Patie model can be derived

$$\frac{\partial V}{\partial t} + \sum_{i=1}^{n} rS_i \frac{\partial V}{\partial S_i} + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\sum_{k=1}^{n} v_{ik}v_{jk}\right) S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} - rV = 0, \tag{5.13}$$

and the option price can be obtained by solving this equation. An example for $n = 2$ is given as follows.

Example 5.2.1. *Suppose S_1, S_2 satisfy the stochastic processes*

$$dS_1 = \tilde{\mu_1} S_1 dt + \tilde{\sigma}_{11} S_1 dW_1 + \tilde{\sigma}_{12} S_2 dW_2 + \tilde{\rho}_{11} S_1 d\alpha_1 + \tilde{\rho}_{12} S_2 d\alpha_2, \tag{5.14}$$

$$dS_2 = \tilde{\mu_2} S_2 dt + \tilde{\sigma}_{21} S_1 dW_1 + \tilde{\sigma}_{22} S_2 dW_2 + \tilde{\rho}_{21} S_1 d\alpha_1 + \tilde{\rho}_{22} S_2 d\alpha_2, \tag{5.15}$$

and assume $\alpha_i = \phi_i(S_1, S_2, t)$. Again to simplify the notation let $\mu_i = \tilde{\mu}_i S_i$, $\sigma_{ij} = \tilde{\sigma}_{ij} S_j$, $\rho_{ij} = \tilde{\rho}_{ij} S_j$, i.e. replace (5.14) by

$$dS_1 = \mu_1 dt + \sigma_{11} dW_1 + \sigma_{12} dW_2 + \rho_{11} d\alpha_1 + \rho_{12} d\alpha_2, \tag{5.16}$$

$$dS_2 = \mu_2 dt + \sigma_{21} dW_1 + \sigma_{22} dW_2 + \rho_{21} d\alpha_1 + \rho_{22} d\alpha_2, \tag{5.17}$$

Applying Itô's lemma to α_1 and α_2,

$$d\alpha_1 = \frac{\partial \phi_1}{\partial t} + \left(\frac{\partial \phi_1}{\partial S_1} dS_1 + \frac{\partial \phi_1}{\partial S_2} dS_2 \right) + \frac{1}{2} \left(\sum_{i,j=1}^{2} \frac{\partial^2 \phi_1}{\partial S_i \partial S_j} \sum_{k=1}^{2} \sigma_{ik} \sigma_{jk} \right) dt, \tag{5.18}$$

$$d\alpha_2 = \frac{\partial \phi_2}{\partial t} + \left(\frac{\partial \phi_2}{\partial S_1} dS_1 + \frac{\partial \phi_2}{\partial S_2} dS_2 \right) + \frac{1}{2} \left(\sum_{i,j=1}^{2} \frac{\partial^2 \phi_2}{\partial S_i \partial S_j} \sum_{k=1}^{2} \sigma_{ik} \sigma_{jk} \right) dt. \tag{5.19}$$

Combining these equations,

$$dS_1 = h_1 dt + \sigma_{11} dW_1 + \sigma_{12} dW_2 + \rho_{11} \left(\frac{\partial \phi_1}{\partial S_1} dS_1 + \frac{\partial \phi_1}{\partial S_2} dS_2 \right) + \rho_{12} \left(\frac{\partial \phi_2}{\partial S_1} dS_1 + \frac{\partial \phi_2}{\partial S_2} \right) dS_2,$$

$$dS_2 = h_2 dt + \sigma_{21} dW_1 + \sigma_{22} dW_2 + \rho_{21} \left(\frac{\partial \phi_1}{\partial S_1} dS_1 + \frac{\partial \phi_1}{\partial S_2} dS_2 \right) + \rho_{22} \left(\frac{\partial \phi_2}{\partial S_1} dS_1 + \frac{\partial \phi_2}{\partial S_2} \right) dS_2,$$

and rewrite into vector form as

$$I d\vec{S} = \Sigma d\vec{W} + A d\vec{S} + \vec{H} dt, \tag{5.20}$$

where

$$\Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix}, I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \vec{H} = \begin{pmatrix} H_1 \\ H_2 \end{pmatrix},$$

and

$$A = \begin{pmatrix} \rho_{11}\frac{\partial \phi_1}{\partial S_1} + \rho_{12}\frac{\partial \phi_2}{\partial S_1} & \rho_{11}\frac{\partial \phi_1}{\partial S_2} + \rho_{12}\frac{\partial \phi_2}{\partial S_2} \\ \rho_{21}\frac{\partial \phi_1}{\partial S_1} + \rho_{22}\frac{\partial \phi_2}{\partial S_1} & \rho_{21}\frac{\partial \phi_1}{\partial S_2} + \rho_{22}\frac{\partial \phi_2}{\partial S_2} \end{pmatrix} = \rho \times Jac(\phi),$$

where

$$\rho = \begin{pmatrix} \rho_{11} & \rho_{12} \\ \rho_{21} & \rho_{22} \end{pmatrix}$$

represents the liquidities.

Hence, by changing the measure to the risk-neutral one and choosing the functions to be

$$\phi_1 = \frac{\partial V}{\partial S_1},\ \phi_2 = \frac{\partial V}{\partial S_2},$$

eventually the two-dimensional nonlinear Black-Scholes equation can be derived as

$$\frac{\partial V}{\partial t} + r\left(S_1\frac{\partial V}{\partial S_1} + S_2\frac{\partial V}{\partial S_2}\right) + \frac{1}{2}\left(a\frac{\partial^2 V}{\partial S_1^2} + b\frac{\partial^2 V}{\partial S_1 \partial S_2} + c\frac{\partial^2 V}{\partial S_2 \partial S_1} + d\frac{\partial^2 V}{\partial S_2^2}\right) - rV = 0, \tag{5.21}$$

where

$$a = v_{11}^2 + v_{12}^2,\ b = v_{11}v_{21} + v_{12}v_{22},\ c = v_{21}v_{11} + v_{22}v_{12},\ d = v_{21}^2 + v_{22}^2,$$

and

$$\begin{pmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{pmatrix} = (I - A)^{-1}\Sigma,$$

where

$$v_{11} = \frac{\left(1 - \left(\rho_{21}\frac{\partial V}{\partial S_1 \partial S_2} + \rho_{22}\frac{\partial V}{\partial S_2^2}\right)\right)\sigma_{11} + \left(\rho_{11}\frac{\partial V}{\partial S_1 \partial S_2} + \rho_{12}\frac{\partial V}{\partial S_2^2}\right)\sigma_{21}}{det(I - A)},$$

$$v_{12} = \frac{\left(1 - \left(\rho_{21}\frac{\partial V}{\partial S_1 \partial S_2} + \rho_{22}\frac{\partial V}{\partial S_2^2}\right)\right)\sigma_{12} + \left(\rho_{11}\frac{\partial V}{\partial S_1 \partial S_2} + \rho_{12}\frac{\partial V}{\partial S_2^2}\right)\sigma_{22}}{det(I - A)},$$

$$v_{21} = \frac{\left(\rho_{21}\frac{\partial V}{\partial S_1^2} + \rho_{22}\frac{\partial V}{\partial S_1 \partial S_2}\right)\sigma_{11} + \left(1 - \left(\rho_{11}\frac{\partial V}{\partial S_1^2} + \rho_{12}\frac{\partial V}{\partial S_1 \partial S_2}\right)\right)\sigma_{21}}{det(I - A)},$$

$$v_{22} = \frac{\left(\rho_{21}\frac{\partial V}{\partial S_1^2} + \rho_{22}\frac{\partial V}{\partial S_1 \partial S_2}\right)\sigma_{12} + \left(1 - \left(\rho_{11}\frac{\partial V}{\partial S_1^2} + \rho_{12}\frac{\partial V}{\partial S_1 \partial S_2}\right)\right)\sigma_{22}}{det(I-A)}.$$

The determinant can be expanded as

$$det(I-A) = 1 - \left(\rho_{21}\frac{\partial V}{\partial S_1 \partial S_2} + \rho_{22}\frac{\partial V}{\partial S_2^2}\right) - \left(\rho_{11}\frac{\partial V}{\partial S_1^2} + \rho_{12}\frac{\partial V}{\partial S_1 \partial S_2}\right)$$
$$+ \left(\rho_{12}\rho_{21}\left(\frac{\partial^2 V}{\partial S_1 \partial S_2}\right)^2 + \rho_{11}\rho_{22}\frac{\partial^2 V}{\partial S_1^2}\frac{\partial^2 V}{\partial S_2^2} - \rho_{12}\rho_{21}\frac{\partial^2 V}{\partial S_1^2}\frac{\partial^2 V}{\partial S_2^2} - \rho_{11}\rho_{22}\left(\frac{\partial^2 V}{\partial S_1 \partial S_2}\right)^2\right),$$

which can be simplified to

$$det(I-A) = 1 - \rho \times Hess(V) - det(\rho)\left(\frac{\partial^2 V}{\partial S_1 \partial S_2}\right)^2 + det(\rho)\frac{\partial^2 V}{\partial S_1^2}\frac{\partial^2 V}{\partial S_2^2},$$

where $Hess$ means the Hessian matrix.

Note that choosing the parameters to be

$$\Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix} = \begin{pmatrix} \sigma_{11} & 0 \\ 0 & \sigma_{22} \end{pmatrix},$$

$$\rho = \begin{pmatrix} \rho_{11} & \rho_{12} \\ \rho_{21} & \rho_{22} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix},$$

simplifies equation (5.21) to become the standard multi-dimensional linear Black-Scholes equation.

5.3 ADI-Newton Solver

Solving the equation (5.13) is not an easy task as it is necessary to construct a nonlinear solver for high-dimensional PDEs. Using a fully implicit scheme then requires the solution of a huge linear systems within the iterative scheme where the matrix structure normally would have a banded structure from the spatial discretisations. The whole computation is

becoming expensive. Therefore rather than a fully implicit scheme or a Crank-Nicolson scheme, an ADI method is chosen considered as the methodology.

The idea of using an ADI method is to create several fractional time steps in the time-marching scheme, and at each fractional time step, the spatial discretisation is employed implicitly in a chosen direction where as the other terms are discretised explicitly using known values at current fractional time step. To be more precise, consider the two-dimensional nonlinear Black-Scholes equation (5.21) which can be rewritten as

$$\frac{\partial V}{\partial t}+r\left(S_1\frac{\partial V}{\partial S_1}+S_2\frac{\partial V}{\partial S_2}\right)+\frac{1}{2}\left(\alpha S_1^2\frac{\partial^2 V}{\partial S_1^2}+2\beta S_1S_2\frac{\partial^2 V}{\partial S_1\partial S_2}+\gamma S_2^2\frac{\partial^2 V}{\partial S_2^2}\right)-rV=0. \tag{5.22}$$

where $\alpha=\alpha(S_1,S_2,t,V_{S_1S_1},V_{S_1S_2},V_{S_2S_2})$, $\beta=\beta(S_1,S_2,t,V_{S_1S_1},V_{S_1S_2},V_{S_2S_2})$ and $\gamma=\gamma(S_1,S_2,t,V_{S_1S_1},V_{S_1S_2},V_{S_2S_2})$. The solution $V(S_1,S_2,t)$ depends on two different spatial dimensions. The Peaceman-Rachford scheme is followed since it gives a good approximation in the two-dimensional case. Essentially in the backward time-marching scheme, when calculating the value V^{n-1} from V^n, the Peaceman-Rachford scheme suggests splitting it into the the following two fractional steps:

$$\frac{V_{ij}^{n-\frac{1}{2}}-V_{ij}^{n}}{\Delta t/2}=\mathscr{L}_1(V^{n-\frac{1}{2}})+\mathscr{L}_2(V^n)+\mathscr{L}_3(V^n)+\mathscr{L}_4(V^{n-\frac{1}{2}})+\mathscr{L}_5(V^n)-rV_{ij}^{n-\frac{1}{2}}, \tag{5.23}$$

and

$$\frac{V_{ij}^{n-1}-V_{ij}^{n-\frac{1}{2}}}{\Delta\, t/2}=\mathscr{L}_1(V^{n-\frac{1}{2}})+\mathscr{L}_2(V^{n-1})+\mathscr{L}_3(V^{n-\frac{1}{2}})+\mathscr{L}_4(V^{n-\frac{1}{2}})+\mathscr{L}_5(V^{n-1})-rV_{ij}^{n-1}, \tag{5.24}$$

where

$$\mathscr{L}_1(V^n)=\frac{\alpha_{ij}^n(S_1)_i^2}{2}\frac{V_{i+1,j}^n-2V_{ij}^n+V_{i-1,j}^n}{(\Delta S_1)^2},$$

$$\mathscr{L}_2(V^n)=\frac{\gamma_{ij}^n(S_2)_j^2}{2}\frac{V_{i,j+1}^n-2V_{ij}^n+V_{i,j-1}^n}{(\Delta S_2)^2},$$

$$\mathscr{L}_3(V^n) = 2\beta_{ij}^n (S_1)_i (S_2)_j \frac{V_{i+1,j+1}^n - V_{i-1,j+1}^n - V_{i+1,j-1}^n + V_{i-1,j-1}^n}{4\Delta S_1 \Delta S_2},$$

$$\mathscr{L}_4(V^n) = r(S_1)_i \frac{V_{i+1,j}^n - V_{i-1,j}^n}{2\Delta S_1},$$

$$\mathscr{L}_5(V^n) = r(S_2)_j \frac{V_{i,j+1}^n - V_{i,j-1}^n}{2\Delta S_2}.$$

where the computational domain is truncated as $V(S_1,S_2,t) \in [0,(S_1)_{max}] \times [0,(S_2)_{max}] \times [0,T)$ and V_{ij}^n is a discrete approximation to $V(i\Delta S_1, j\Delta S_2, n\Delta t)$. Here $(S_1)_i = i\Delta S_1$, $i = 0,\cdots,N_{S_1}-1$, $(S_2)_j = j\Delta S_2$, $j = 0,\cdots,N_{S_2}-1$, and $n = 0,\cdots,N_t-1$, where N_{S_1} and N_{S_2} are the numbers of grid points for spatial discretisations on S_1 and S_2 directions, and N_t is the number of grid points for temporal discretisation. The sizes of meshes are calculated as $\Delta S_1 = (S_1)_{max}/(N_{S_1}-1), \Delta S_2 = (S_2)_{max}/(N_{S_2}-1), \Delta t = T/(N_t-1)$.

Therefore equations (5.23) and (5.24) provide an approximation for getting the answer V^{n-1} from V^n and can be re-formulating as the following

$$H_1(V^{n-\frac{1}{2}})V^{n-\frac{1}{2}} = f_1(V^n), \tag{5.25}$$

$$H_2(V^{n-1})V^{n-1} = f_2(V^{n-\frac{1}{2}}), \tag{5.26}$$

where H_1 is a $(N_{S_1}-2)\times(N_{S_1}-2)$ tri-diagonal matrix defined as $[a,b,c]$ and a,b,c are $(N_{S_1}-2)\times 1$ column vectors represent the lower, main and upper diagonal entries of H_1 which are

$$a_i = \frac{\Delta t}{4}\left(\frac{-r(S_1)_i}{\Delta S_1} + \frac{\alpha_{ij}^{n-\frac{1}{2}}(S_1)_i^2}{(\Delta S_1)^2}\right),$$

$$b_i = -1 - \frac{\Delta t}{2}\left(\frac{\alpha_{ij}^{n-\frac{1}{2}}(S_1)_i^2}{(\Delta S_1)^2} - r\right),$$

$$c_i = \frac{\Delta t}{4}\left(\frac{r(S_1)_i}{\Delta S_1} + \frac{\alpha_{ij}^{n-\frac{1}{2}}(S_1)_i^2}{(\Delta S_1)^2}\right),$$

where $i = 1, \cdots N_{S_1} - 2$. $f_1(V^n)$ represents the right hand side values which can be evaluated from the known values V^n at the current time step.

Similarly H_2 is a $(N_{S_2} - 2) \times (N_{S_2} - 2)$ tri-diagonal matrix defined as $[d, e, f]$ and d, e, f are $(N_{S_2} - 2) \times 1$ column vectors represent the lower, main and upper diagonal entries of H_2 which are

$$d_j = \frac{\Delta t}{4}\left(\frac{-r(S_2)_j}{\Delta S_2} + \frac{\gamma_{ij}^{n-1}(S_2)_j^2}{(\Delta S_2)^2}\right),$$

$$e_j = -1 - \frac{\Delta t}{2}\left(\frac{\gamma_{ij}^{n-1}(S_2)_j^2}{(\Delta S_2)^2} - r\right),$$

$$f_j = \frac{\Delta t}{4}\left(\frac{r(S_2)_j}{\Delta S_2} + \frac{\gamma_{ij}^{n-1}(S_2)_j^2}{(\Delta S_2)^2}\right),$$

where $j = 1, \cdots N_{S_2} - 2$. $f_2(V^{n-\frac{1}{2}})$ again represents the right hand side values which can be evaluated from the known values $V^{n-\frac{1}{2}}$ when moving to the second step of the Peaceman-Rachford scheme.

The advantage of applying the Peaceman-Rachford scheme is readily seen from the matrices H_1 and H_2 which both very simple tri-diagonal structures of smaller size than the system for the fully implicit scheme. The nonlinear problem at each step is similar to the one-dimensional case, and the methodology of solving the nonlinear systems (5.25) and (5.26) is again to use the root-finding approach which can be expressed as solving

$$G_1(V^{n-\frac{1}{2}}) = H_1(V^{n-\frac{1}{2}})V^{n-\frac{1}{2}} - f_1(V^n) = 0, \tag{5.27}$$

$$G_2(V^{n-1}) = H_2(V^{n-1})V^{n-1} - f_2(V^{n-\frac{1}{2}}) = 0, \tag{5.28}$$

where G_1 and G_2 are the root-finding functions for the first and the second step in the Peaceman-Rachford scheme. Equations (5.27) and (5.28) can be solved by either applying the frozen coefficient method or Newton-Raphson method as in Chapter 2. Note that using the Newton-Raphson method, the explicit Jacobian matrix is not easily obtained as the volatility

function is complicated and consequently finite difference approximation and Broyden type methods have to be employed. The nonlinearities can also be approximated using the initial guesses V^* and V^{**} in the iterative scheme. The developed solver, termed the ADI-Newton method, is explained in Algorithm 7.

Algorithm 7: ADI-Newton Method

Input: initial guesses V^*, V^{**}, terminal condition $V^{N_t-1} = V(S_1, S_2, t = T)$, tol
Output: $V^0 = V(S_1, S_2, t = 0)$
for $n = N_t - 1 : 1$ **do**
  <ADI Step 1>
  for $j = 2 : N_{S_2} - 1$ **do**
    for $k = 1 : N_{ite}$ **do**
      1. Calculate $G_1(V^*)$ by equation (5.27);
      2. Calculate $res = -\left[Jac(G_1(V^*))\right]^{-1} G_1(V^*)$;
      if $\frac{\|G_1(V^*)\|}{\max(1, \|V^*\|)} < tol$ **then**
        $V^{n-\frac{1}{2}} = V^*$, go to *<ADI Step 2>* ;
        else
          $V^* = V^* + res$, go back to 1.;
        end
      end
    end
  end
  <ADI Step 2>
  for $i = 2 : N_{S_1} - 1$ **do**
    for $k = 1 : N_{ite}$ **do**
      3. Calculate $G_2(V^{**})$ by equation (5.28);
      4. Calculate $res = -\left[Jac(G_2(V^{**}))\right]^{-1} G_2(V^{**})$;
      if $\frac{\|G_2(V^{**})\|}{\max(1, \|V^{**}\|)} < tol$ **then**
        $V^{n-1} = V^{**}$, break;
        else
          $V^{**} = V^{**} + res$, go back to 3.;
        end
      end
    end
  end
end

5.4 Numerical Experiments

Two numerical experiments were designed to examine the accuracy of using this ADI-Newton method by calculating the experimental order of convergence (*eoc*) at a given point. The *eoc* number was constructed similarly to the methodology in Chapter 2. Let V_h be the solution evaluated with mesh sizes $h\Delta S_1$ for spatial discretisation on S_1 direction, $h\Delta S_2$ for spatial discretisation on S_2 direction and $h\Delta t$ for temporal discretisation, then the ratio of error with asymptotic behaviour was constructed as:

$$R = \frac{\|V_h - V_{h/2}\|}{\|V_{h/2} - V_{h/4}\|}, \tag{5.29}$$

where the $eoc = \log_2(R)$. The other choice is taking the solution evaluated with small mesh size to be the true solution and then the ratio of error can be defined as:

$$\hat{R} = \frac{\|V_h - \hat{V}\|}{\|V_{h/2} - \hat{V}\|}, \tag{5.30}$$

where $\hat{V}$ is the solution evaluated with refined grid points and the $e\hat{o}c = \log_2(\hat{R})$.

The first numerical experiment focuses on spread option pricing and the payoff function was chosen to be

$$V(S_1, S_2, T) = \Phi(S_1, S_2, K) = \max(S_1 - S_2 - K, 0).$$

The truncated domain for finite difference discretisation was set as $S_1 \in [0, 1000]$, $S_2 \in [0, 400]$. The strike price, maturity, and interest rate were fixed as $K = 100, T = 1/12, r = 0.005$. The volatilities also the liquidities were chosen to be

$$\begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix} = \begin{pmatrix} 0.4 & 0 \\ 0 & 0.1 \end{pmatrix}, \quad \begin{pmatrix} \rho_{11} & \rho_{12} \\ \rho_{21} & \rho_{22} \end{pmatrix} = \begin{pmatrix} 0.01 & 0 \\ 0 & 0.005 \end{pmatrix},$$

Table 5.1 ADI-Newton method with the l_2 norm for spread option case.

$N_{S_1} = N_{S_2}$	N_t	Err	R	eoc	$\hat{Err}$	$\hat{R}$	$\hat{eoc}$
11	1001	0.053	2.44	1.29	0.083	2.76	1.46
21	1001	0.021	3.36	1.75	0.030	3.64	1.86
41	1001	0.006	3.70	1.89	0.008	4.70	2.23
81	1001	0.001			0.001		
161	1001						

and boundary conditions were taken as

$$V(0,S_2,t) = 0, \ \frac{\partial V}{\partial S_2}(S_1,0,t) = \frac{\partial V}{\partial S_1}(S_{1max},S_2,t) = \frac{\partial V}{\partial S_2}(S_1,S_{2max},t) = 0, \tag{5.31}$$

for $t \in [0,T)$.

Table 5.1 shows the results of *eoc* numbers where N_{S_1}, N_{S_2} are grid numbers for spatial discretisations in S_1, S_2 directions, and N_t is the grid number of temporal discretisation which is fixed to be $N_t = 1001$ in order to observe the convergent behaviour of the error from spatial discretisations. The number *eoc* actually presents the accuracy for the spread option case as $O((\Delta S_1)^{2-\varepsilon} + (\Delta S_2)^{2-\varepsilon}))$ where ε is a small number.

Fig 5.1 shows the computation time for using the ADI-Newton method, with different approaches for evaluating the Jacobian matrix. It can be observed that Broyden's method is faster than finite difference as the number of grid points are getting larger. Fig 5.2 also shows the average number of iterations for the two steps of the ADI-Newton method. In step 1 the number is higher, which can is understandable as the numerical experiment was designed for the truncated domain in which S_1 is greater than S_2 and it reflects the fact that the mesh size $\Delta S_1 > \Delta S_2$ with a fixed number of grid points along both spatial axes. This also shows the reason why the computation time in step 1 is higher than in step 2. Note that both steps actually only require around 5 iterations to guarantee convergence which shows the benefit of using Newton-Raphson method.

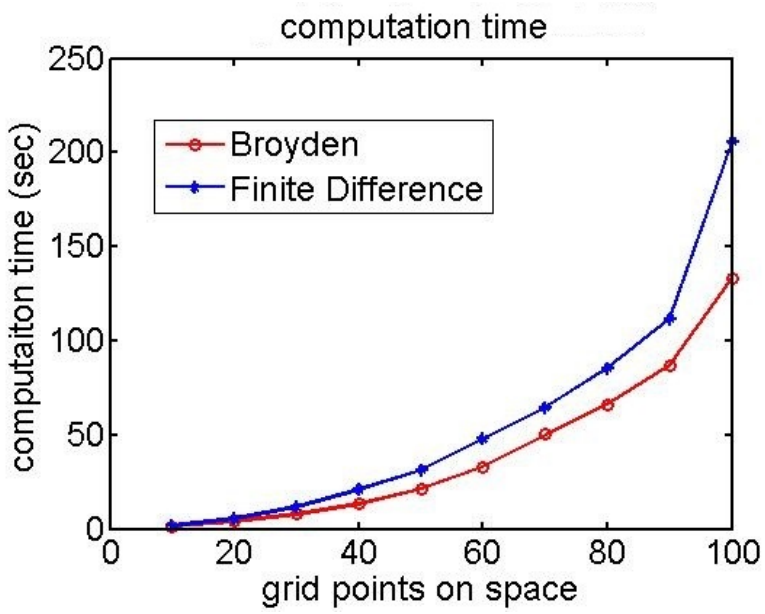

Fig. 5.1 Computation time for using finite difference method with Broyden's method to approximate the Jacobian matrix.

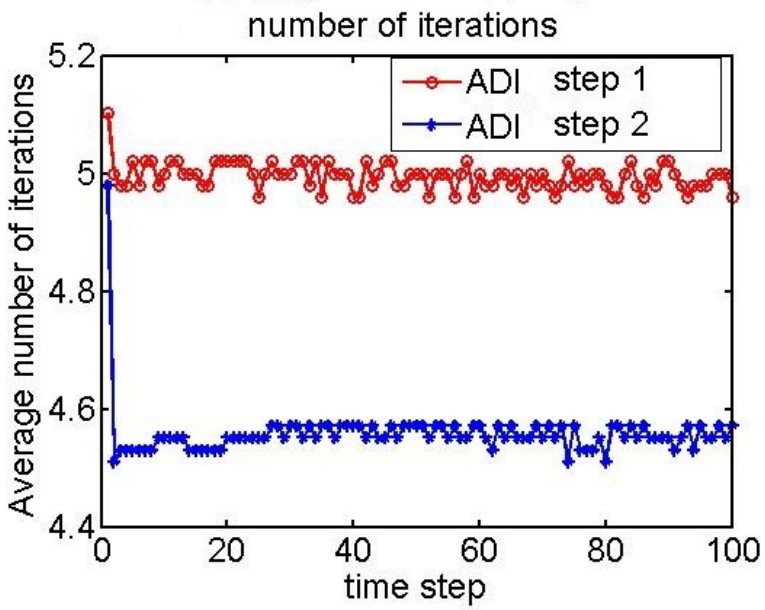

Fig. 5.2 Average of number of iterations of ADI-Newton method at step 1 and step 2 for spread option case where $N_{S_1} = N_{S_2} = 101$.

Table 5.2 ADI-Newton method with the l_2 norm for basket option case.

$N_{S_1} = N_{S_2}$	N_t	Err	R	eoc	$\hat{Err}$	$\hat{R}$	$\hat{eoc}$
16	1001	0.5002	1.76	0.81	0.8610	2.38	1.25
31	1001	0.2838	4.57	2.19	0.3608	4.68	2.22
61	1001	0.0620	4.13	2.05	0.0770	5.13	2.35
121	1001	0.0150			0.0150		
241	1001						

The second numerical experiment involved calculating the price of a basket option with the payoff function chosen to be

$$V(S_1,S_2,T) = \Phi(S_1,S_2,K) = \max\left(\frac{S_1+S_2}{2} - K, 0\right).$$

The truncated domain for the finite difference discretisation was set as $S_1 \in [0,300]$, $S_2 \in [0,300]$. Other parameters were chosen to be the same as the case in spread option. The boundary conditions were taken as

$$\frac{\partial V}{\partial S_1}(0,S_2,t) = \frac{\partial V}{\partial S_2}(S_1,0,t) = \frac{\partial V}{\partial S_1}(S_{1max},S_2,t) = \frac{\partial V}{\partial S_2}(S_1,S_{2max},t) = 0. \tag{5.32}$$

Table 5.2 presents the results for *eoc* numbers and again $N_t = 1001$ for observing the convergent behaviour of the error from spatial discretisations. In this case the number *eoc* can reach around 2 which essentially gives the accuracy of second order in space when Δt is fixed.

Fig 5.3 again shows the average number of iterations for the two steps of the ADI-Newton method. The only difference occurs in the first few time steps, and after both ADI steps actually only require 3 iterations on average to converge in general. The small number of iterations is the benefit of using ADI-Newton method to solve multi-dimensional nonlinear equations.

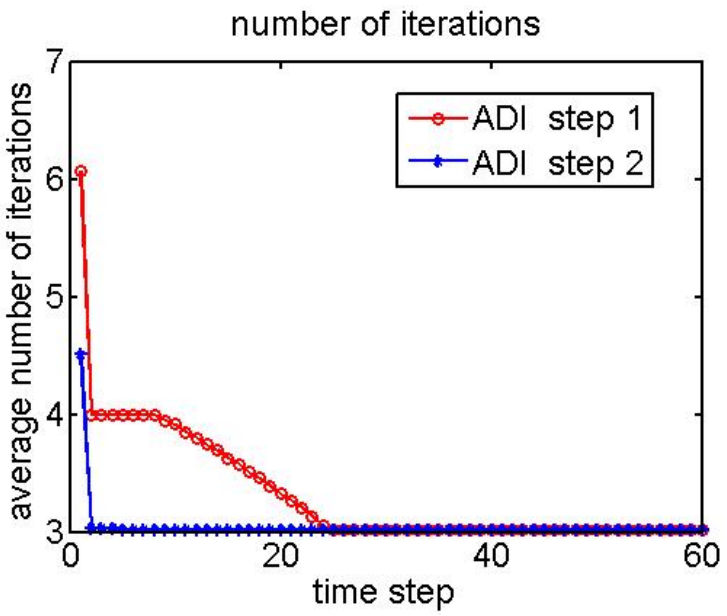

Fig. 5.3 Average of number of iterations of ADI-Newton method at step 1 and step 2 for basket option case where $N_{S_1} = N_{S_2} = 61$.

5.5 Summary

In this chapter, the idea of using the Frey-Patie model to explore the option price behaviour in illiquid markets is extended for options which depend on multiple assets. A multi-dimensional nonlinear Black-Scholes equation is derived for the multi-asset problem and this approach is followed for pricing such options.

Two cases including spread and basket options in two dimensions are examined with a merged solver which incorporates both the ADI method and the Newton-based solver. This ADI-Newton solver has the advantage of solving a nonlinear system with a smaller matrix size and requires only a small number of iterations. The accuracy of the solver depends on the problem but can reach second order on space when taking the mesh size of temporal discretisation to be very small.

Chapter 6

Conclusion

6.1 Conclusion

In this thesis an overview of various nonlinear problems in option pricing is given in the introduction, and a detailed discussion in terms of mathematical modelling, numerical nonlinear PDE solvers and parallel computing has been given. The model provided by Frey and Patie in [43] is chosen as the focus and extended to high-dimensional cases. A brief summary of research contributions is as follows.

First of all, efficient and robust nonlinear PDE solvers based on Newton-Raphson method are constructed for solving nonlinear Black-Scholes equations. A principal conclusion for the one-dimensional case is that applying Newton-Raphson method with a direct solver is considered to be best as it performs efficiently due to the simple tri-diagonal structure of the nonlinear system and fast convergence of the iterative scheme. Numerical experiments have demonstrated that this constructed Newton-based solver does achieve the expected accuracy after suitable adjustments. These results have been published in [29, 33, 35].

Secondly, a parallel batched Newton-based solver is built for solving large-scale nonlinear PDEs with implementations on GPUs. These large-scale problems come from multiple contracts where the option pricing uses nonlinear models, and the batched Newton-based solver

enables pricing these independent options with multiple parameters at the same time. Good performance solutions can be obtained by using sophisticated GPU libraries and allocating memory appropriately, and numerical tests demonstrate that good speedups can be achieved when the size of problems becomes large.

Finally, a feature of the constructed Newton-based solvers is the solvers are able to be combined with other numerical schemes when handling high-dimensional problems. Two case studies, including Asian option and multi-asset problems, have been described generating the SL-Newton and ADI-Newton solvers. An important issue to address is that the expected accuracy from the chosen scheme is sometimes not easily achievable as the merged schemes contain different sources of numerical errors. However these merged schemes reduce the computation time and also data storage and therefore it is recommended to use them for two-dimensional problems.

These combined solvers will add to the numerical tools for the wide range of nonlinear PDEs available to the finance industry. The parallel computing implementation allows the solution to be obtained efficiently. With these tools it is feasible to generate practical insights and understanding of the effects of nonlinearities in the real market.

6.2 Outlook

The numerical solvers described in the thesis are suitable for dealing with basic nonlinear problems in finance. To have more generic, compact and powerful solutions, there are still some challenging issues which need to be overcome and are considered as future work.

6.2.1 Robust Methods to Handle the Singularity

The first one is developing systematic ways to handle singularities which occur at specific discontinuous points, addressed and discussed with some remedies in [48] and in Chapter 2. These issues appear in high-dimensional cases as well depending on the payoff functions since the nonlinear volatility functions still depend on the sensitivities which may generate more instabilities in the nonlinear solvers. Therefore extensions of these adjustments for singularities are needed for incorporation into the developed SL-Newton and ADI-Newton solvers. The techniques provided in [48] are a good basis for extensions for handling high-dimensional nonlinear option pricing problems.

6.2.2 GPU Computing for High-dimensional Problems

The solvers introduced in Chapter 4 and 5 can potentially be the basis for large-scale problems, where GPU computing is very suitable using the framework proposed in Chapter 3. Some extra settings would need to be employed to adjust the SL-Newton and ADI-Newton solvers to formulate the batched Newton-based solver. The main challenge comes from the memory allocation for doing extra calculations, e.g. interpolations for Asian option pricing which is memory bound and takes more time than doing the batch operation or tri-diagonal solver. The parallel ADI solver proposed in [22] which the authors used to solve a three-factor model PDE would be followed and compared. Also in [14] the parallel PDE solver using the semi-Lagrangian scheme actually solves similar equations for doing Asian option pricing using similar ideas. These references focus on the constant volatility, and essentially the aim would be to obtain reasonable speedups optimise the performance for nonlinear case.

6.2.3 American Options

The option pricing problems discussed in this thesis concentrate on European options, which only allow exercise at maturity. There are more possibilities such as American options, which give the right to early exercise, in real financial markets. There is only a small literature base discussing the nonlinearity impact from volatility in American option pricing as it is

difficult to handle both the free boundary problem also to build a nonlinear PDE solver. Some techniques using moving boundary transformation proposed in [33] enable the solution of the nonlinear American option pricing problem in the call option case and the authors examined the case of transaction costs model given by [7]. It is considered to be future work to transfer this technique to the Frey-Patie model and to construct Newton-based solvers which are suitable for the American option case.

References

[1] S. Alanko. Stability of Regression-Based Monte Carlo Methods for Solving Nonlinear PDEs. *Communications on Pure and Applied Mathematics*, 69(5):958–980, 2016.

[2] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen. LAPACK: A portable linear algebra library for high-performance computers. In *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, pages 2–11. IEEE Computer Society Press, 1990.

[3] M. Avellaneda, A. Levy, and A. Parás. Pricing and hedging derivative securities in markets with uncertain volatilities. *Applied Mathematical Finance*, 2(2):73–88, 1995.

[4] M. Avellaneda and A. Paras. Managing the volatility risk of portfolios of derivative securities: the Lagrangian uncertain volatility model. *Applied Mathematical Finance*, 3(1):21–52, 1996.

[5] G. Barles. Convergence of Numerical Schemes for Degenerate Parabolic Equations Arising in finance Theory. *Numerical methods in finance*, 13:1, 1997.

[6] G. Barles, C. Daher, and M. Romano. Convergence of numerical schemes for parabolic equations arising in finance theory. *Mathematical models and methods in applied Sciences*, 5(01):125–143, 1995.

[7] G. Barles and H. M. Soner. Option pricing with transaction costs and a nonlinear Black-Scholes equation. *Finance and Stochastics*, 2(4):369–397, 1998.

[8] F. Black and M. Scholes. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

[9] L. A. Bordag. *Geometrical Properties of Differential Equations: Applications of Lie Group Analysis in Financial Mathematics*. World Scientific Publishing Co Inc, 2015.

[10] L. A. Bordag and R. Frey. Pricing options in illiquid markets: symmetry reductions and exact solutions. 2008.

[11] B. Bouchard and N. Touzi. Discrete-time approximation and Monte-Carlo simulation of backward stochastic differential equations. *Stochastic Processes and their Applications*, 111(2):175–206, 2004.

[12] P. Boyle, M. Broadie, and P. Glasserman. Monte Carlo methods for security pricing. *Journal of Economic Dynamics and Control*, 21(8):1267–1321, 1997.

[13] Z. Buckova, M. Ehrhardt, and M. Günther. Alternating direction explicit methods for convection diffusion equations. *Acta Mathematica Universitatis Comenianae*, 84(2):309–325, 2015.

[14] D. Castillo, A. Ferreiro, J. A. García-Rodríguez, and C. Vázquez. Numerical methods to solve PDE models for pricing business companies in different regimes and implementation in GPUs. *Applied Mathematics and Computation*, 219(24):11233–11257, 2013.

[15] L.-W. Chang and W. H. Wen-mei. A Guide for Implementing Tridiagonal Solvers on GPUs. In *Numerical Computations with GPUs*, pages 29–44. Springer, 2014.

[16] I. J. Clark. *Commodity Option Pricing: A Practitioner's Guide*. John Wiley & Sons, 2014.

[17] R. Company, L. Jódar, and J.-R. Pintos. A consistent stable numerical scheme for a nonlinear option pricing model in illiquid markets. *Mathematics and Computers in Simulation*, 82(10):1972–1985, 2012.

[18] S. Cook. *CUDA PROGRAMMING: A DEVELOPER'S GUIDE TO PARALLEL COMPUTING WITH GPUs*. Newnes, 2012.

[19] I. J. Craig and A. D. Sneyd. An alternating-direction implicit scheme for parabolic equations with mixed derivatives. *Computers & Mathematics with Applications*, 16(4):341–350, 1988.

[20] M. G. Crandall, H. Ishii, and P.-L. Lions. User's guide to viscosity solutions of second order partial differential equations. *Bulletin of the American Mathematical Society*, 27(1):1–67, 1992.

[21] M. G. Crandall and P.-L. Lions. Viscosity solutions of Hamilton-Jacobi equations. *Transactions of the American Mathematical Society*, 277(1):1–42, 1983.

[22] D.-M. Dang, C. Christara, and K. R. Jackson. A Parallel Implementation on GPUs of ADI Finite Difference Methods for Parabolic PDEs with Applications in Finance. 2010.

[23] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton Methods. *SIAM Journal on Numerical analysis*, 19(2):400–408, 1982.

[24] J. W. Demmel, M. T. Heath, and H. A. Van Der Vorst. Parallel numerical linear algebra. *Acta numerica*, 2:111–197, 1993.

[25] Y. d'Halluin, P. A. Forsyth, and G. Labahn. A semi-Lagrangian approach for American Asian options under jump diffusion. *SIAM Journal on Scientific Computing*, 27(1):315–345, 2005.

[26] J. Douglas. Alternating direction methods for three space variables. *Numerische Mathematik*, 4(1):41–63, 1962.

[27] J. Douglas and H. H. Rachford. On the Numerical Solution of Heat Conduction Problems in Two and Three Space Variables. *Transactions of the American Mathematical Society*, 82(2):421–439, 1956.

[28] D. J. Duffy. *Finite difference methods in financial engineering: A Partial Differential Equation Approach*. John Wiley & Sons, 2013.

[29] K. Ďuriš, S.-H. Tan, C.-H. Lai, and D. Ševčovič. Comparison of the Analytical Approximation Formula and Newton's Method for Solving a Class of Nonlinear

Black–Scholes Parabolic Equations. *Computational Methods in Applied Mathematics*, 16(1):35–50, 2016.

[30] D. Egloff. GPUs in Financial Computing Part II: Massively Parallel PDE Solvers on GPUs. *Wilmott Magazine*, pages 50–53, 2010.

[31] D. Egloff. Part I: High-Performance Tridiagonal Solvers on GPUs for partial differential equations. *Wilmott Magazine*, pages 32–40, 2010.

[32] D. Egloff. GPUs in Financial Computing Part III: ADI Solvers on GPUs with Application to Stochastic Volatility. *Wilmott*, 52:51–53, 2011.

[33] V. Egorova, S.-H. Tan, C.-H. Lai, R. Company, and L. Jódar. Moving boundary transformation for American call options with transaction cost: finite difference methods and computing. *International Journal of Computer Mathematics*, 94(2):345–362, 2017.

[34] M. Ehrhardt. *Nonlinear Models in Mathematical Finance: New Research Trends in Option Pricing*. Nova Science Publishers, 2008.

[35] M. Ehrhardt, M. Günther, and E. J. W. ter Maten. *Novel Methods in Computational Finance*, volume 25. Springer, 2017.

[36] M. Ehrhardt and R. Valkov. Numerical analysis of nonlinear European option pricing problem in illiquid markets. 2014.

[37] N. El Karoui, S. Peng, and M. C. Quenez. Backward stochastic differential equations in finance. *Mathematical Finance*, 7(1):1–71, 1997.

[38] A. Fahim, N. Touzi, and X. Warin. A probabilistic numerical method for fully nonlinear parabolic PDEs. *The Annals of Applied Probability*, pages 1322–1364, 2011.

[39] Y. Fan and H. Zhang. The pricing of Asian options in uncertain volatility model. *Mathematical Problems in Engineering*, 2014, 2014.

[40] X. Feng, R. Glowinski, and M. Neilan. Recent developments in numerical methods for fully nonlinear second order partial differential equations. *SIAM Review*, 55(2):205–267, 2013.

[41] W. H. Fleming and H. M. Soner. *Controlled Markov Processes and Viscosity Solutions*, volume 25. Springer Science & Business Media, 2006.

[42] R. Frey. Market illiquidity as a source of model risk in dynamic hedging. *Model Risk*, pages 125–136, 2000.

[43] R. Frey and P. Patie. Risk management for derivatives in illiquid markets: A simulation study. In *Advances in Finance and Stochastics*, pages 137–159. Springer, 2002.

[44] R. Frey and U. Polte. Nonlinear Black–Scholes Equations in Finance: Associated Control Problems and Properties of Solutions. *SIAM Journal on Control and Optimization*, 49(1):185–204, 2011.

[45] M. Giles, E. László, I. Reguly, J. Appleyard, and J. Demouth. GPU implementation of finite difference solvers. In *Proceedings of the 7th Workshop on High Performance Computational Finance*, pages 1–8. IEEE Press, 2014.

[46] M. Giles and I. Reguly. Trends in high-performance computing for engineering calculations. *Phil. Trans. R. Soc. A*, 372(2022):20130319, 2014.

[47] M. B. Giles and R. Carter. Convergence analysis of Crank-Nicolson and Rannacher time-marching. *Journal of Computational Finance*, pages 89–112, 2006.

[48] K. J. Glover, P. W. Duck, and D. P. Newton. On nonlinear models of markets with finite liquidity: some cautionary notes. *SIAM Journal on Applied Mathematics*, 70(8):3252–3271, 2010.

[49] E. Gobet. *Méthodes de Monte-Carlo et processus stochastiques: du linéaire au non-linéaire*. Editions de l'Ecole Polytechnique, 2013.

[50] E. Gobet, J.-P. Lemor, X. Warin, et al. A regression-based Monte Carlo method to solve backward stochastic differential equations. *The Annals of Applied Probability*, 15(3):2172–2202, 2005.

[51] E. Gobet, J. López-Salas, P. Turkedjiev, and C. Vázquez. Stratified regression Monte-Carlo scheme for semilinear PDEs and BSDEs with large scale parallelization on GPUs. *SIAM Journal on Scientific Computing*, 38(6):C652–C677, 2016.

[52] G. H. Golub and C. F. Van Loan. *Matrix Computations*, volume 3. JHU Press, 2012.

[53] J. Guo and W. Wang. On the numerical solution of nonlinear option pricing equation in illiquid markets. *Computers & Mathematics with Applications*, 69(2):117–133, 2015.

[54] J. Guyon and P. Henry-Labordère. *Nonlinear Option Pricing*. CRC Press, 2013.

[55] T. Haentjens, K. i. Hout, T. E. Simos, G. Psihoyios, and C. Tsitouras. ADI Finite Difference Discretization of the Heston-Hull-White PDE. In *AIP Conference Proceedings*, volume 1281, pages 1995–1999. AIP, 2010.

[56] T. Haentjens and K. J. In't Hout. Alternating direction implicit finite difference schemes for the Heston-Hull-White partial differential equation. *Journal of Computational Finance*, 16(1):83, 2012.

[57] P. Heider. Numerical methods for non-linear Black-Scholes equations. *Applied Mathematical Finance*, 17(1):59–81, 2010.

[58] A. Hirsa. *Computational Methods in Finance*. CRC Press, 2012.

[59] J. C. Hull. *OPTIONS, FUTURES, AND OTHER DERIVATIVES*. Pearson Education India, 2006.

[60] W. Hundsdorfer. Accuracy and stability of splitting with stabilizing corrections. *Applied Numerical Mathematics*, 42(1-3):213–233, 2002.

[61] W. Hundsdorfer and J. G. Verwer. *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*, volume 33. Springer Science & Business Media, 2013.

[62] K. in't Hout and C. Mishra. Stability of the modified Craig–Sneyd scheme for two-dimensional convection–diffusion equations with mixed derivative term. *Mathematics and Computers in Simulation*, 81(11):2540–2548, 2011.

[63] K. in't Hout and C. Mishra. Stability of ADI schemes for multidimensional diffusion equations with mixed derivative terms. *Applied Numerical Mathematics*, 74:83–94, 2013.

[64] K. in't Hout, T. E. Simos, G. Psihoyios, and C. Tsitouras. ADI Schemes in the Numerical Solution of the Heston PDE. In *AIP Conference Proceedings*, volume 936, pages 10–14. AIP, 2007.

[65] K. In't Hout and B. Welfert. Stability of ADI schemes applied to convection-diffusion equations with mixed derivative terms. *Applied Numerical Mathematics*, 57(1):19–35, 2007.

[66] K. In't Hout and B. Welfert. Unconditional stability of second-order ADI schemes applied to multi-dimensional diffusion equations with mixed derivative terms. *Applied Numerical Mathematics*, 59(3-4):677–692, 2009.

[67] M. Jandačka and D. Ševčovič. On the risk-adjusted pricing-methodology-based valuation of vanilla options and explanation of the volatility smile. *Journal of Applied Mathematics*, 2005(3):235–258, 2005.

[68] H. B. Keller. Numerical methods in boundary-layer theory. *Annual Review of Fluid Mechanics*, 10(1):417–433, 1978.

[69] C. T. Kelley. *Solving Nonlinear Equations with Newton's Method*, volume 1. SIAM, 2003.

[70] A. G. Kemna and A. Vorst. A pricing method for options based on average asset values. *Journal of Banking & Finance*, 14(1):113–129, 1990.

[71] E. Kirk. Correlation in the energy markets. *Managing Energy Price Risk*, 1:71–78, 1995.

[72] D. A. Knoll and D. E. Keyes. Jacobian-free Newton–Krylov methods: a survey of approaches and applications. *Journal of Computational Physics*, 193(2):357–397, 2004.

[73] M. Kratka. No mystery behind the smile. *RISK-LONDON-RISK MAGAZINE LIMITED-*, 11:67–71, 1998.

[74] J. Kraus, M. Schlottke, A. Adinetz, and D. Pleiter. Accelerating a C++ CFD code with OpenACC. In *Proceedings of the First Workshop on Accelerator Programming Using Directives*, pages 47–54. IEEE Press, 2014.

[75] W. Kress and B. Gustafsson. Deferred correction methods for initial boundary value problems. *Journal of Scientific Computing*, 17(1-4):241–251, 2002.

[76] C. Labart and J. Lelong. A Parallel Algorithm for solving BSDEs-Application to the pricing and hedging of American options. *arXiv preprint arXiv:1102.4666*, 2011.

[77] C. Labart and J. Lelong. A parallel algorithm for solving BSDEs. *Monte Carlo Methods and Applications*, 19(1):11–39, 2013.

[78] E. László, Z. Nagy, M. B. Giles, I. Reguly, J. Appleyard, and P. Szolgay. Analysis of parallel processor architectures for the solution of the Black-Scholes PDE. In *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, pages 1977–1980. IEEE, 2015.

[79] A. T. Layton and M. L. Minion. Implications of the choice of quadrature nodes for Picard integral deferred corrections methods for ordinary differential equations. *BIT Numerical Mathematics*, 45(2):341–373, 2005.

[80] H. E. Leland. Option pricing and replication with transactions costs. *The Journal of Finance*, 40(5):1283–1301, 1985.

[81] D. C. Lesmana and S. Wang. An upwind finite difference method for a nonlinear Black–Scholes equation governing European option valuation under transaction costs. *Applied Mathematics and Computation*, 219(16):8811–8828, 2013.

[82] T. J. Lyons. Uncertain volatility and the risk-free synthesis of derivatives. *Applied Mathematical Finance*, 2(2):117–133, 1995.

[83] W. Margrabe. The value of an option to exchange one asset for another. *The Journal of Finance*, 33(1):177–186, 1978.

[84] J. M. Martınez. Practical quasi-Newton methods for solving nonlinear systems. *Journal of Computational and Applied Mathematics*, 124(1):97–121, 2000.

[85] M. A. Milevsky and S. E. Posner. A closed-form approximation for valuing basket options. *The Journal of Derivatives*, 5(4):54–61, 1998.

[86] P. G. Multigrid. Numerical solution of partial differential equations on parallel computers. *Lecture Notes in Computational Science and Engineering*, 51, 2006.

[87] NVIDIA. cuBLAS LIBRARY DOCUMENTATION. *http://docs.nvidia.com/pdf/CUBLAS_Library.pdf*.

[88] NVIDIA. CUDA Toolkit Documentation. *http://docs.nvidia.com/cuda*.

[89] NVIDIA. cuSPARSE LIBRARY DOCUMENTATION. *http://docs.nvidia.com/pdf/CUSPARSE_Library.pdf*.

[90] NVIDIA. GPU architecture. *http://www.nvidia.com/object/gpu-architecture.html*.

[91] NVIDIA. Kepler Compute Architecture White Paper. *http://international.download.nvidia.com/pdf/kepler/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf*.

[92] J. M. Ortega and R. G. Voigt. Solution of partial differential equations on vector and parallel computers. *SIAM review*, 27(2):149–240, 1985.

[93] E. Pardoux and S. Peng. Backward stochastic differential equations and quasilinear parabolic partial differential equations. *Stochastic partial differential equations and their applications*, pages 200–217, 1992.

[94] A. Parrott and S. Rout. Semi-Lagrange time integration for PDE models of Asian options. In *Progress in Industrial Mathematics at ECMI 2004*, pages 432–436. Springer, 2006.

[95] K. Parrott, N. Clarke, et al. A parallel solution of early exercise asian options with stochastic volatility. In *Proceedings of the 11th Domain Decomposition Conference, Greenwich*, volume 19, 1998.

[96] D. W. Peaceman and H. H. Rachford, Jr. The numerical solution of parabolic and elliptic differential equations. *Journal of the Society for Industrial and Applied Mathematics*, 3(1):28–41, 1955.

[97] E. Polizzi and A. Sameh. SPIKE: A parallel environment for solving banded linear systems. *Computers & Fluids*, 36(1):113–120, 2007.

[98] E. Polizzi and A. H. Sameh. A parallel hybrid banded system solver: the SPIKE algorithm. *Parallel Computing*, 32(2):177–194, 2006.

[99] D. M. Pooley, P. A. Forsyth, and K. R. Vetzal. Numerical convergence properties of option pricing PDEs with uncertain volatility. *IMA Journal of Numerical Analysis*, 23(2):241–267, 2003.

[100] R. Rannacher. Finite element solution of diffusion problems with irregular data. *Numerische Mathematik*, 43(2):309–327, 1984.

[101] L. C. G. Rogers and Z. Shi. The value of an Asian option. *Journal of Applied Probability*, 32(04):1077–1088, 1995.

[102] K. Rupp, J. Weinbub, F. Rudolf, A. Morhammer, T. Grasser, and A. Jüngel. A Performance Comparison of Algebraic Multigrid Preconditioners on CPUs, GPUs, and Xeon Phis. *Under Review*, 2015.

[103] J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming, Portable Documents*. Addison-Wesley Professional, 2010.

[104] L. Schubert. Modification of a quasi-Newton method for nonlinear equations with a sparse Jacobian. *Mathematics of Computation*, 24(109):27–30, 1970.

[105] D. Ševcovic, B. Stehlıková, and K. Mikula. Analytical and numerical methods for pricing financial derivatives. *Nova Science Publ. ISBN*, pages 978–1, 2011.

[106] D. Ševčovič and M. Žitňanská. Analysis of the nonlinear option pricing model under variable transaction costs. *Asia-Pacific Financial Markets*, 23(2):153–174, 2016.

[107] S. E. Shreve. *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer Science & Business Media, 2004.

[108] J. Stoer and R. Bulirsch. *Introduction to numerical analysis*, volume 12. Springer Science & Business Media, 2013.

[109] D. Tangman, A. Peer, N. Rambeerich, and M. Bhuruth. Fast simplified approaches to Asian option pricing. *Journal of Computational Finance*, 14(4):3, 2011.

[110] D. Tavella and C. Randall. *Pricing financial instruments: The finite difference method*, volume 13. John Wiley & Sons, 2000.

[111] S. Williams, D. D. Kalamkar, A. Singh, A. M. Deshpande, B. Van Straalen, M. Smelyanskiy, A. Almgren, P. Dubey, J. Shalf, and L. Oliker. Optimization of geometric multigrid for emerging multi-and manycore processors. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 96. IEEE Computer Society Press, 2012.

[112] P. Wilmott, S. Howison, and J. Dewynne. *The mathematics of financial derivatives: a student introduction*. Cambridge University Press, 1995.

[113] P. Wilmott and P. J. Schönbucher. The feedback effect of hedging in illiquid markets. *SIAM Journal on Applied Mathematics*, 61(1):232–272, 2000.

[114] M. M. Wolf, M. A. Heroux, and E. G. Boman. Factors impacting performance of multithreaded sparse triangular solve. In *International Conference on High Performance Computing for Computational Science*, pages 32–44. Springer, 2010.

[115] M. Wyns et al. Convergence of the Modified Craig–Sneyd scheme for two-dimensional convection–diffusion equations with mixed derivative term. *Journal of Computational and Applied Mathematics*, 296:170–180, 2016.

[116] Y. Zhang, J. Cohen, and J. D. Owens. Fast tridiagonal solvers on the GPU. *ACM Sigplan Notices*, 45(5):127–136, 2010.