A comparison between finite difference and binomial methods for solving American single-stock options

ALEXANDER ERIKSSON

Master of Science Thesis
Stockholm, Sweden 2013

A comparison between finite difference and binomial methods for solving American single-stock options

A L E X A N D E R E R I K S S O N

Master's Thesis in Numerical Analysis (30 ECTS credits)
Degree Progr. in Engineering Physics 270 credits
Royal Institute of Technology year 2013
Supervisor at KTH was Anders Szepessy
Examiner was Michael Hanke

TRITA-MAT-E 2013:17
ISRN-KTH/MAT/E--13/17--SE

Royal Institute of Technology
School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden

URL: www.kth.se/sci

Abstract

In this thesis, we compare four different finite-difference solvers with a binomial solver for pricing American options, with a special emphasis on achievable accuracy under computational time constraints. The three finite-difference solvers are: an operator splitting method suggested by S. Ikonen and J. Toivanen, a boundary projection method suggested by M. Brennan and E. Schwartz, projected successive overrelaxation and second order accurate operator splitting method known as Peaceman-Rachford. The binomial method is a modified variant employing an analytical final step as suggested by M. Broadie and J. Detemple. The model problem is an American put option, and we empirically examine the effects of the relevant numerical parameters on the quality of the solutions. For the finite-difference methods we utilize both a Crank-Nicolson discretization and a fully implicit second-order-in-time discretization.

We conclude that the operator splitting method suggested by S. Ikonen and J. Toivanen is the Alternating Direction Implicit algorithm known as the Douglas-Rachford algorithm. We also conclude that the accuracy of the Peaceman-Rachford algorithm degrades to first order for the American option problem.

Of the finite-difference methods tried, the Douglas-Rachford algorithm has the highest performance in terms of accuracy under computational time constraints. We conclude that it does, however, not outperform the modified binomial model.

Referat

En jämförelse mellan finita differens och binomial metod för prisättning av Amerikanska aktieoptioner med en underliggande

I denna uppsats jämför vi fyra olika finita-differensmetoder med en binomial metod för prissättning av amerikanska optioner, med särskild tonvikt på noggrannhet som kan uppnås inom begränsningar satta på beräkningstiden. De tre finita differens lösarna är: en operatoruppdelande metod som föreslagits av S. Ikonen och J. Toivanen, en projektion-påhindret metod föreslagen av M. Brennan och E. Schwartz, projicerad successiv överrelaxering och en andra ordningens noggrann operaturuppdelande metod känd som Peaceman-Rachford. Den använda binomial metoden använder sig av ett analytiskt sista steg som föreslagits av M. Broadie och J. Detemple. Modelproblemet är en amerikansk säljoption, och vi undersöker empiriskt effekterna som de relevanta numeriska parametrar har på kvaliteten hos lösningarna. För finita-differensmetoderna undersöks både Crank-Nicolson diskretisering och en i tiden andra ordningens implicit diskretisering.

Vi drar slutsatsen att operatoruppdelningsmetoden som föreslagits av S. Ikonen och J. Toivanen är den Alternerande Riktning Implicita algoritm som är känd som Douglas-Rachfords algoritm. Vi drar också slutsatsen att noggranheten hos Peaceman-Rachfords algoritm degenerar till första ordningen för det Amerikanska optionsproblemet.

Av de finita differensmetoder vi undersökt här har Douglas-Rachford den högsta prestandan i fråga om noggrannhet under beräkningsmässiga tidsbegränsningar. Vi drar slutsatsen att den dock inte presterar bättre än den modifierade binomialmodellen.

Contents

1 Introduction **1**
- 1.1 The European and American options . . . 1
- 1.2 No Arbitrage . . . 2
- 1.3 The Black-Scholes model for the underlying . . . 3
- 1.4 Reformulating the European and the American options . . . 4

2 The binomial model **7**
- 2.1 An outline of the binomial model . . . 7

3 Numerical Theory for finite-difference methods **11**
- 3.1 Finite difference Methods - Discrete Grid . . . 11
  - 3.1.1 The discrete equation and Crank-Nicolson . . . 14
  - 3.1.2 Backwards Differentiation Formula 2 - BDF2 . . . 20

4 Describing the finite difference methods examined **23**
- 4.1 Projected Successive Over Relaxation, PSOR . . . 23
  - 4.1.1 Jacobi and Gauss-Seidel for European options . . . 23
  - 4.1.2 Successive Overrelaxation (SOR) for European options . . . 25
  - 4.1.3 Projected Successive Overrelaxation (PSOR) for American options . . . 25
  - 4.1.4 PSOR with BDF2 . . . 25
  - 4.1.5 Comments on PSOR . . . 26
- 4.2 Interlude: LU-decomposition . . . 26
- 4.3 Operator splitting . . . 27
  - 4.3.1 What Ikonen-Toivanen suggested . . . 28
  - 4.3.2 Theoretical Background, rediscovering Douglas-Rachford . . . 31
  - 4.3.3 With BDF2 discretization . . . 40
- 4.4 Direct solution with projection on the boundary - Brennan-Schwartz 40
  - 4.4.1 Brennan-Schwartz with BDF2 . . . 41
- 4.5 A connection to operator splitting . . . 41
- 4.6 Peaceman-Rachford . . . 42

5 Numerical Practicalities **43**

6 Results **45**
6.1 Error as a function of calculation time 45
6.1.1 Rannacher Start up and other time stepping 47
6.1.2 Backward differentation formula of order 2 50
6.1.3 Shifting the truncation of the boundary 52
6.1.4 Peaceman-Rachford 53
6.2 Conclusions 54
6.3 Things of note 55
6.3.1 Improvements on the implementation 55
6.3.2 Preconditioner for PSOR 56
6.3.3 Multigrid methods 56
6.3.4 Theoretical error analysis 56
6.3.5 The selection of $s_{\max}$ - revisited 56

Appendices **57**

A Additional comments and theory **59**
A.1 Machine specifications 59
A.2 Other models for the underlying 59
A.3 Differential inclusion 60
A.4 LU-decomposition and the TDMA 60
A.5 Demonstrating that the Operator Split actually approximates the problem 62
A.6 Yet another way to write Douglas-Rachford 63
A.7 Resolvents and Projections 64
A.8 Justifying the approximation for the reference values 64
A.9 Empirical tests for the numerical parameters 66
A.9.1 Grid size and its effects on the error 67
A.9.2 Within a grid size 69
A.10 Motivating the L^∞ error 71

Bibliography **73**

Chapter 1

Introduction

The objective of this paper is to provide a comparison between American option pricing using on the one hand the binomial model and on the other various finite difference methods, with an emphasis on studying the trade off between accuracy and calculation times. The goal is to study the advantages and disadvantages of the different methods, not just in theory, but also to explore how to employ them practically. Because of the intended application, the interesting region in terms of accuracy and computational speed is an error of at most the order 10^{-3}, at one millisecond or less of computational time.

We begin by explaining some preliminaries relating to modelling the evolution of the asset price that underlies the option, and two common option styles of interest, in order to better understand the problem we are seeking to analyze. If you have previous experience with options and financial mathematics you may wish to skip ahead to the last section of this chapter, which contains the partial differential equation we solve to price the options, and related notation that will be useful later.

1.1 The European and American options

The holder of an option has the right but not the obligation to perform a particular action, hence the name. Basic examples of options are *call* and *put* options, where a call option means you have a right to purchase something, and a put option gives you the right to sell something. In most common applications, the price at which the option is exercised - how much you have the right to buy or sell something for - is known when the option is first issued. This price is called the *strike* price, and will be denoted by K.

Two common option styles are of interest here, European and American options. European options (for example, European call options) are options where the option is issued at time zero, and the ability to exercise it occurs at the end if its lifespan (the *maturity* of the option). Once the option has matured, you are asked

whether you wish to exercise it or not. To summarize for clarity, a European call option on a particular stock is a contract which gives the holder of the contract the right, not the obligation, to buy a stock at a specified strike price on a specified date (maturity).

An American option on the other hand, allows exercise from the time it is issued until maturity. In other words, the right to buy a stock at the strike price is always available during the time leading up to expiry. There are other option styles, though these are the two simplest, and tend to be readily available to be purchased on exchanges.

Of particular interest to our examination is the fact that European options can be priced using analytical formulae in the framework we lay out below, where as American options can not, and we are thus forced to resort to numerical approximations to pursue our chosen problem. Because of our emphasis on limiting calculation time, we will not examine Monte Carlo methods - which are calculation intense but can solve the same problem - but focus on a few finite difference methods.

1.2 No Arbitrage

Arbitrage profit, in the context of financial mathematics, refers to profits that are risk-free (though strictly speaking, the risk-free property is in many cases an approximation). A *portfolio* is a combination of assets, for example stocks, options and savings or loans, or more tangible things like physical properties or commodities. Note that we also include negative assets, i.e. debt or other asset related obligations in a portfolio, making it possible to construct a portfolio containing both assets and debt that is worth 0 today.

An *arbitrage portfolio*, in turn, is a portfolio that is worth nothing today, but that is worth strictly more than zero with probability 1, at some future point in time. A mathematical definition, found in [18, p. 7], states that the portfolio h is with value V_0^h at time zero and value V_1^h at time 1 is an arbitrage portfolio if,

$$V_0^h = 0, \tag{1.1}$$

$$V_1^h > 0, \quad \text{with probability 1.} \tag{1.2}$$

A useful example is to consider a portfolio that arises from borrowing money, then buying something with that money, and then selling that at some future point in time. The combination of the sum you borrowed and the worth of what you bought can be zero, but if you can sell what you bought for more than what you owe in the future, and you know that this occurs with probability 1 within the constraints of your model, then you have an arbitrage portfolio.

Practically, the existence of an arbitrage portfolio is viewed as an example of mispricing - the value of things traded on an exchange (under a few other idealising constraints) must be such that there are no arbitrage opportunities, and if it this is not the case then something has been priced incorrectly. Obviously, this is either an opportunity to make risk-free money, or a somewhat more opaque way to give away money for nothing depending on your stake in the portfolio.

When dealing with theoretical prices, one then tends to make assumptions that ensure that prices do not give rise to arbitrage opportunities - this creates a connection between "reasonable" prices of options and the price of the underlying of the option, with some involvement of the available risk-free interest rate. This allows us to begin by first deciding on a model for the underlying, which we can then use to construct a price for the option.

1.3 The Black-Scholes model for the underlying

To determine the price of an option, one needs a way to model the evolution of the price of the underlying. In the case of options on stocks or stock indices, a very common model is the Black-Scholes model, proposed by Fischer Black and Myron Scholes in 1973. Without going into the all of the specifics of the assumptions made in the model, the model states that the stock price can be modelled with a stochastic differential equation. We introduce the stock price $S(t)$, a stochastic process.[1] We also introduce a stochastic Wiener process, $W(t)$. A Wiener process is a stochastic process that has independent and normally distributed increments, starts at 0 and has continuous trajectories. A more detailed description can be found in, for example, [18, p. 40]. In the original Black-Scholes model, we have two constants, σ, the *volatility*, and r, the risk-free rate of interest.

The stochastic differential equation (SDE) suggested by Black and Scholes to model the price evolution of a stock price (where the stock pays no dividends), using the notation introduced above, is

$$dS(t) = \mu dt + \sigma dW(t). \tag{1.3}$$

The magnitude of the volatility decides how large the stochastic variation will be, and the constant μ denotes the *drift*, which is a deterministic part of the evolution that depends only on time. (μ does not need to be constant either, in some models it depends not only on t but also on S).

The relation (1.3) is the stochastic differential equation that governs the evolution of the underlying asset (a stock price) in the traditional Black-Scholes model. Using a model for a risk-free bank account where the evolution of the amount of

[1] Formally, S is the stochastic process consisting of a collection of ordered stochastic variables, and $S(t)$ is a stochastic variable ordered by t.

money B in the bank account can be characterized according to $dB = rBdt$ and a no-arbitrage argument while constructing a portfolio using the bank account, the stock (modelled as in (1.3)) and an option on the stock, we arrive at the Black-Scholes equation, a partial differential equation for the price f of an option. The specifics are available in for example [18, p. 98-102], we will settle for simply listing it here,

$$\frac{\partial f}{\partial t} + rs\frac{\partial f}{\partial s} + \frac{1}{2}s^2\sigma^2\frac{\partial^2 f}{\partial s^2} = rf. \tag{1.4}$$

The solution to this PDE gives the only option price that is consistent with the idea of no arbitrage in the Black-Scholes framework. We will use the Black-Scholes model for the underlying in our examination, as it is commonly used in practice.

1.4 Reformulating the European and the American options

The goal of this section is to examine the Black-Scholes option pricing PDE seen in (1.4), and setting the stage for pricing American options. To this end, we begin by simplifying the notation of the PDE by introducing the operator A, much as in [17]. When employing operators, we will use $A(x)$ and Ax to indicate that A operates on x.

$$\begin{aligned}
&\frac{\partial f}{\partial t} + rs\frac{\partial f}{\partial s} + \frac{1}{2}s^2\sigma^2\frac{\partial^2 f}{\partial s^2} = rf \\
&\frac{\partial f}{\partial t} = -rs\frac{\partial f}{\partial s} - \frac{1}{2}s^2\sigma^2\frac{\partial^2 f}{\partial s^2} + rf \\
&\frac{\partial f}{\partial t} + A(f) = 0, \quad \text{where } A(f) = rs\frac{\partial f}{\partial s} + \frac{1}{2}s^2\sigma^2\frac{\partial^2 f}{\partial s^2} - rf.
\end{aligned} \tag{1.5}$$

Different options imply different boundary conditions for the equation above. For a European option, $f(s,T) = g(s)$ where g is the payoff function of that particular option. For a standard European put option, $g(s) = \max(K - s, 0)$ where K is the strike price.

For an American option, in addition to the boundary condition at time T presented for the European option above, the early exercise constraint must be taken into account. This constraint leads to a time dependent **L**inear **C**omplementarity **P**roblem (LCP)[2] [17]. The value of an American option must satisfy the time

[2]In reality, applying Black-Scholes, the American option constraints and the idea that no arbitrage opportunities can persist in a market equilibrium leads to a free boundary problem, which one can show is equivalent to the Linear Complementarity Problem presented above, including that any solution to one is a solution to the other. However, demonstrating this equivalence involves concepts from functional analysis and is considered beyond the scope of this text.

dependent LCP below:

$$\begin{cases} \frac{\partial f}{\partial t} + A(f) \geq 0, \\ f(s,t) \geq g(s), & \forall (s,t) \in [0,\infty) \times [0,T], \\ (f-g)(\frac{\partial f}{\partial t} + A(f)) = 0, \end{cases} \tag{1.6}$$

where A is the operator from (1.5) and g is the payoff function at exercise. This is the problem that we will aim to solve numerically. The reader interested in more theory related to LCPs and the solution thereof is referred to [19].

Remark The model problem examined throughout this thesis is an American put option. We choose the put over a call because in a no-dividends framework, the American call option has exactly the same value as the European call option, making it a less interesting problem to examine. These options are also quite commonly traded, which makes it important to have algorithm that can value them rapidly.

Chapter 2

The binomial model

The system in (1.6) does not readily yield a closed-form solution, and so we turn to numerical analysis to solve the problem numerically.

We will focus on a "tree method" in the case of the binomial method, or finite difference methods in case of the others. We will first present the binomial model, as it has the honour of being the one we wish to size the others against, and because it needs less supportive theory.

2.1 An outline of the binomial model

In the binomial approach, as outlined in [4], the evolution of the stock prices modelled by Black-Scholes is approximated by a succession of independent binomial stochastic variables. These variables take one of two possible values, where the probabilities for either is determined via no arbitrage arguments [1]. The values themselves depend on how many time steps we divide the relevant interval into and the volatility of the underlying.

The tree starts with the spot price today, and depending on the outcome of a binomial stochastic variable, either takes an up or down step with the up-or-down probabilities (specified below) in the next time step - corresponding to multiplying the current price with either the up or down step value. At each of the two points generated, we repeat the procedure, so that for every time step forward in time, the number of nodes in the tree increases. Depending on your choice of parameters, and the problem you wish to solve using the method, you can recombine the tree, and reach the same node in the next time step by either going up or down from two different nodes in the current time steps. If recombination is possible the number

[1] For the binomial model employed here the probabilities used are the "martingale probabilities", and they depend on the risk free rate of interest. This choice of the probabilities leads to the risk neutral measure, though defining it in greater detail is considered beyond the scope of this text. See for example [18, Ch. 2] for a more in-depth look at the probability theory involved.

of nodes increase by 1 for every time step. Using the recombining characterization given by [4] is therefore preferable. For some path dependent options recombination is not possible, and the tree doubles in size for every time step.

Once a tree for the evolution of the underlying price has been generated, this tree can be used to calculate the price of an option. Schematically, we can say that we generate a second tree, this one containing option values. By applying the boundary for $t = T$ to the branches of the spot tree and storing this in the branches of the option tree we have a starting point. We then time step backwards from T to $t = 0$. At each node we take note of the discounted expected value of holding on to the option to the next time step. This is possible because we have defined the probabilities for reaching those nodes, and in the previous time step we calculated the value in those same nodes.

The final node in the option tree will then contain the value of the option at $t = 0$. In the case of an American option, at each node in the option tree, we compare and take the maximum of the value if exercised with the expected value of holding on to the option [18, Ch. 2]. The binomial model can for European options be shown to converge to the price calculated by the Black-Scholes analytic formula as the number of steps increases. (And for a trinomial tree rather than a binomial one, it can also be seen as an application of the explicit finite difference method outlined below.)

Expressed in more mathematical terms, the step - either "up" or "down" from the spot price in the previous node - are defined according to (more in-depth reasoning as to why is available in [18, Ch. 2])

$$\begin{aligned} u &= e^{\sigma\sqrt{\Delta t}}, \\ d &= e^{-\sigma\sqrt{\Delta t}}. \end{aligned} \tag{2.1}$$

If take a step up, we multiply the spot price with u. If we take a step down, we multiply the spot price with d. The probability of taking a step upwards is defined by q_u, and the probability of taking a step downwards is defined by q_d. These probabilities are called martingale probabilities, and they are the only probabilities consistent with the no-arbitrage condition. We will not analyze this claim further here, the interested reader can consult [18, p. 9], and we settle for stating them according to

$$\begin{aligned} q_u &= \frac{e^{r\Delta t} - d}{u - d}, \\ q_d &= 1 - q_u. \end{aligned} \tag{2.2}$$

Next, we look closer at the spot price at an arbitrary node. We denote the spot price today by s_0. We note that for a recombining tree there are many pathways to reach a specific node, but for the construction of spot prices we only really need

one such pathway, due to the martingale property.[2] We let the integer k denote the number of "upticks" necessary to reach a given node, and the integer l denote the number of "downticks". The total number of ticks, i.e. time steps, that have occurred can be characterized by $k+l$ for each node. We enumerate each time step by $i = 0, 1, 2, \ldots, N$. All possible underlying spot prices at the time $t = i\Delta t$ are then the $s_{k,l}$ for which $k + l = i$.

Using the preceding,

$$s_{k,l} = s_0 u^k d^l, \tag{2.3}$$

will characterize the spot price of the underlying at every node.

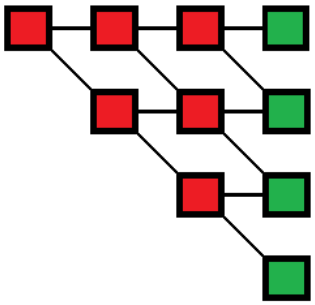

Figure 2.1. Schematic overview of a small recombining binomial tree. The elements of the rightmost column are known values, and represent the value of the option at T. There will be $N + 1$ columns in total. By combining the nodes in pairs, the value in a previous node is calculated. This is repeated until the single node located at $t = 0$, which then contains the answer.

Let $f_i(s_{k,l})$ be the option price at the node given by $s_{k,l}$ (note that $i = k + l$). $g(s_{k,l})$ corresponds to the exercise payoff at that spot price node, for an American put option, $g(s) = \max(K - s, 0)$. Then the option price for $i = 0, 1, 2, \ldots, N - 1$ can be characterized according to

$$\begin{aligned} f_i(s_{k,l}) &= \max\left(\mathbb{E}[f_{i+1}], g(s_{k,l})\right) = \\ &= \max\left(q_u f_{i+1}(s_{k+1,l}) + q_d f_{i+1}(s_{k,l+1}),\ g(s_{k,l})\right). \end{aligned} \tag{2.4}$$

For $i = N$, the option price is instead $g(s_{k,l})$ for all k, l such that $N = k + l$.

In computational practice

Although we above described the construction using two trees, one for the spot price and one for the option price, in practice we have an analytical expression for

[2] In this context, the martingale property simply means that the spot price is path independent: The expected value in a future node depends only on the value in the current node, not the value in past node. For a more detailed explanation, consult once more the early chapters of [18].

the spot price at each node where we are interested in the option price, so we only need one tree. More importantly however, we can relate the spot price at each node to the price of its neighbours with just multiplication and division.

In terms of computational effort, the ability to relate the spot price to the price at neighbouring nodes using only division and multiplication matters a great deal, as multiplication is less costly than exponentiation.

As we need a price at time T to start with, we calculate analytically the spot price at the top-most node, that is, the node constructed from only upticks at each time step, $s_0 u^N d^0 = s_{N,0}$. The other spot prices at time T are then calculated by multiplying $s_{N,0}$ with a combination of inverse upticks (u^{-1}) and downticks (d), as $s_{k-1,l+1} = s_{k,l} \cdot u^{-1} \cdot d$ and $s_{k-1,l} = s_{k,l} \cdot u^{-1}$. These two calculations are both cheaper than the calculations involving exponentiations and can easily be implemented alongside the time stepping scheme.

Smoothing oscillations via analytical formulae

One of the difficulties of the binomial model (shared by the finite difference method described later) is oscillations introduced by non-smooth boundary conditions, or barriers. Such oscillations mean convergence will not be monotone, meaning that an increase in the number of time steps do not necessarily decrease the error. The impact this has on the error decreases as we increase the number of timesteps, but as the purpose of our examination is speed we have an interest in keeping the number of time steps as low as possible.

One way to address such oscillations is use an analytic formula in the final step, as suggested and tested in [9] under the name "Binomial Black-Scholes", or BBS. In the case of American options with European counterparts that have closed form expressions, in the final step, we calculate the expected value of the payoff using the closed form expression instead of the binomial approximation. We illustrate with an example on the American put, using the notation previously introduced. For $i = N-2, N-3, \ldots, 1, 0$ we still use (2.4). For $i = N-1$ we replace (2.4) with

$$f_{N-1}(s_{k,l}) = \max\left(P(s_{k,l}, (N-1)\Delta t), g(s_{k,l})\right), \tag{2.5}$$

where $P(s,t)$ is the Black-Scholes price for the *European* put option, which is well-known and available in any standard work, for example, [20, p. 313].

As the BBS method significantly increases the accuracy for a lower number of time steps, our implementation will feature the BBS modification.

Chapter 3

Numerical Theory for finite-difference methods

Next, we present the finite difference methods. Finite difference methods (FDM) are built on the idea of approximating the partial derivatives in the continuous PDE with approximations from Taylor series expansion around the points of interest. These approximations are referred to as finite difference approximations as they involve using small, but finite rather than infinitesimal, differences of the dependent variable (here $f(s,t)$). [25, ch. 8]. Readers interested in a more general overview of numerical approximation can consult [6, p. 206-224].

We begin by examining the approximation of the continuous domain, followed by closer look at the Crank-Nicolson discretization, and then - reusing much of the notation in the Crank-Nicolson case - present the discretization according to the Backward Differentiation Formula of order 2.

3.1 Finite difference Methods - Discrete Grid

In order to apply finite difference methods, we will need to discretize and truncate the domain of $f(s,t)$, as defined in (1.5). This means we create new spaces corresponding to the domain, and corresponding discrete functions by projecting the linear functions onto the discrete grid. The resulting discrete functions and discrete domains are used to approximate the actual, continuous, functions and domain.

We will perform this discretization with evenly spaced steps in either direction, that is both in time and in spot price[1]. The discretization in the direction of the spot price will be referred to as the space discretization, due to ease of generalization

[1] Although more complicated schemes could be applied, for example, the number of grid points near the price of interest can be increased and the number of grid points far away could be decreased. Figuring out how to distribute them near the interesting region, or identifying the interesting region, is not computationally free however, and is not given a great deal of attention here.

to other problems. Although one could carry out substitutions that would transform our problem to heat-diffusion, we will not explore that route as it is makes it harder to generalize the numerical schemes for other models of the underlying.

To perform the actual discretization, we begin by letting N be the number of time *steps*, and let $i = 0, 1, 2, \ldots, N$.

We then let $\Delta t_i = \Delta t = \frac{T}{N}$. [2] Then $N + 1$ is the number of time *points*, $i = 0$ gives $i\Delta t = 0$ and $i = N$ gives $i\Delta t = T$. Generally in this thesis, $t_i = i\Delta t$, assuming that the distance between each point is the constant Δt.

Spatial discretization and domain truncation
Discretization in the space direction is more complex, as it is an infinite domain rather than a bounded one. This is addressed by truncating the domain, introducing $s_{\max}$ and discretizing from $s = 0$ to $s = s_{\max}$. In general, the nature of the Black-Scholes PDE (a diffusion-convection equation) means that errors originating from imperfect boundary conditions on the truncated boundary are attenuated, as long as the truncated boundary is sufficiently far away from the point of interest. But for a uniform grid, increasing $s_{\max}$ will increase the number of points, even though these points will contain no useful data.

Care must therefore be taken to ensure that $s_{\max}$ is sufficiently far from the interesting region to attenuate numerical errors, but not too far, as the distance increase required computational effort.

In addition, choosing the boundary condition to place on the truncated boundary requires careful evaluation. A simple method is to simply apply the payoff as a Dirichlet condition, which will generate good results if $s_{\max}$ is large. A more complex way is to instead impose conditions on the second derivative, which is more accurate, yet carries with it a more complex discretization that ruins the tridiagonal structure of the system. This, in turn, means more time consuming methods may need to be applied to solve the system, and any gains in terms of computational efficiency made from choosing more exact boundary conditions are consumed by the increased time for solving the system. By previous work in the field, we conclude that $s_{\max}$ at four times the exercise of the option should ensure that the errors generated by the truncation are small enough so as to be negligible, and choose to use the standard Dirichlet payoff on the boundaries [22, p. 120-130] [15].

Having established a suitable point for truncation of the spatial domain, we continue with the actual discretization of that domain. Let M be the number of space steps, and let $j = 0, 1, 2, \ldots, M$. Then we have $\Delta s_j = \Delta s = \frac{s_{\max}}{M}$. Analogous to the time grid, throughout this thesis it holds that $s_j = j\Delta s$ - but one could also

[2] Distinguishing Δt_i from Δt is only useful for time steps that are not uniform

constructs discretizations where this is not the case.

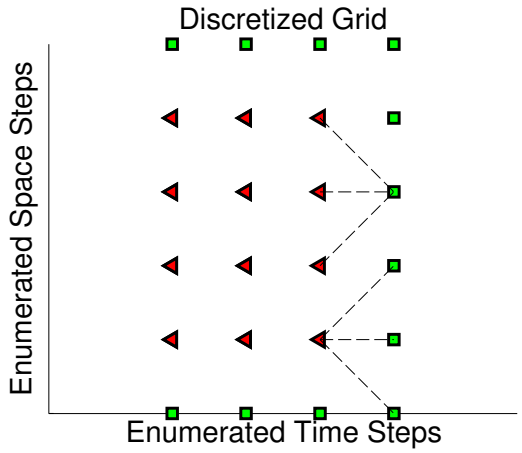

Figure 3.1. A rough image of the grid points for some (small) values of N and M. The green squares correspond to the known boundary values - to the right $t = T$, and the top and bottom correspond to 0 and $s_{\max}$ respectively. The red triangles represents the values that will be calculated. The calculation will start from the right and move left, one column per time step. The leftmost edge will correspond to the price today. The dashed lines demonstrate implicit (top set) and explicit (bottom set) schemes, by showing which known grid points are used to find a value for an unknown grid point.

With the above formulation, each column is a vector corresponding to a given time step, and each row corresponds to a given spot price. The column with the highest index, with the notation used here $N + 1$, corresponds to the time T. This column will then contain the boundary values at time T for the option we wish to price. The bottom and the top of this grid, corresponding to $s = 0$ and $s = s_{\max}$ respectively, will also require boundary values that depend on the nature of the option.

Interpolation and requirements on grid points

The discretization introduces a problem in any calculation, particularly noticeable for a smaller number of steps. In the spatial direction, we might be seeking a value that does not lie on a grid point in our discretization, so we cannot just use the calculated price at a point as the price of the option.

Two principal ways of dealing with this suggest themselves. The first is to choose $s_{\max}$ and M so that the desired price is on the grid. While easy to implement, this complicates our error analysis by decoupling Δs from M, and is therefore less useful for our experiments.[20, p. 456].

The other is interpolation. In [22, p. 112-120] a study of a "worst-case" example of having to use interpolation is carried out. From it, we conclude that a cubic spline interpolation will satisfy our demands on error tolerance, and guided by [24, p. 113-116] we implement cubic spline interpolation with natural splines.

Remark Interpolation assists our examination because it makes the total error more predictable, while simulating the practical conditions under which a method is used. Using interpolation lets us ask the FD methods the same question as we ask the binomial method, and receive a comparable answer.

3.1.1 The discrete equation and Crank-Nicolson

It is useful to provide a brief overview of two different classifications of discretizing the time derivative.

If one uses an *implicit* time discretization (e.g. Euler backwards), at each time step, a system of linear equations needs to be solved. We express the solution (for this time step) in a way that makes the equations for each spatial point connected via the linear system to all other spatial points. This method is robust: as long as Δs and Δt tend to zero, the numerical solution will converge to the exact solution. However, solving the resulting linear system can be very time consuming.

An *explicit* method (e.g. Euler forward) will instead use several of the previous values in order to generate the new values, in a way that leaves the values in the currently examined step uncoupled. No linear system of equations arises, and the solution is much easier to acquire computationally. However, for these methods, the solution might not actually solve the problem - there is a limited area of stability that can instead cause the solution to explode, i.e. increase in an unbounded fashion that has nothing to do with how the real system behaves. [20, p. 456-461].[3]

With the preceding in mind, we turn towards discretizing the Black-Scholes equation. We will first do so by the Crank-Nicolson method. Crank-Nicolson is a weighted "average" of an implicit and an explicit discretization, where each of the methods are given equal weights. This combination leads to a method that has a faster rate of convergence than Euler backward, although oscillations are introduced for problems such as ours where the approximations of the boundaries cannot be continuous. [20, p. 465]

In order to readily offer an overview of implicit, explicit and Crank-Nicolson at the same time, we will deal with the averaging via a parameter θ. For $\theta = \frac{1}{2}$ this discretization leads to the Crank-Nicolson method, for $\theta = 1$ it leads to a fully implicit method (Euler backwards) and for $\theta = 0$ a fully explicit (Euler forward)

[3]For a visual representation of implicit and explicit, return to figure 3.1 - the top set of connecting arrows represents the implicit solver, and the bottom the explicit.

method. First, let us recall the LCP from (1.6), but with the Black-Scholes operator inserted for clarity:

$$\begin{cases} \frac{\partial f}{\partial t} + rs\frac{\partial f}{\partial s} + \frac{1}{2}s^2\sigma^2\frac{\partial^2 f}{\partial s^2} - rf \geq 0, \\ f(s,t) \geq g(s), & \forall s,t \in [0,\infty) \times [0,T]. \\ (f(s,t) - g(s))(\frac{\partial f}{\partial t} + rs\frac{\partial f}{\partial s} + \frac{1}{2}s^2\sigma^2\frac{\partial^2 f}{\partial s^2} - rf) = 0. \end{cases} \quad (3.1)$$

Our next task is to perform a finite difference approximation of the partial derivatives above, starting with the space derivatives. In order to do this we introduce a simplifying piece of notation along with the discrete approximation;

$$f(s_j, t_i) = f(j\Delta s, i\Delta t) \approx f_{i,j} \quad (3.2)$$

We then note the following well-known central difference approximations for the space derivatives, where $\mathcal{O}$ is ordo and is used to denote the order of the (error) term remaining:

$$\frac{\partial f_{i,j}}{\partial s} = \frac{f_{i,j+1} - f_{i,j-1}}{2\Delta s} + \mathcal{O}(\Delta s^2) \quad (3.3)$$

$$\frac{\partial^2 f_{i,j}}{\partial s^2} = \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{\Delta s^2} + \mathcal{O}(\Delta s^2) \quad (3.4)$$

We introduce $\underline{\mathrm{A}}$ as the discretized Black-Scholes operator and ignore the error term (as Δs tends to zero, so will the error term, and it will do so faster than the rest of the approximation).

$$\underline{\mathrm{A}}(f_{i,j}) = rs_j\frac{f_{i,j+1} + f_{i,j-1}}{2\Delta s} + \frac{1}{2}s_j^2\sigma^2\frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{\Delta s^2} - rf_{i,j}. \quad (3.5)$$

We prepare to apply the discretization of the Black-Scholes operator to our LCP by looking at an example problem, $\frac{\partial f}{\partial t} + Af = 0$. Using the notation above, Euler backward applied to the example problem:

$$\frac{f_{i+1,j} - f_{i,j}}{\Delta t} + \underline{\mathrm{A}}(f(s_j, t_i)) = 0. \quad (3.6)$$

Euler forward, same notation:

$$\frac{f_{i+1,j} - f_{i,j}}{\Delta t} + \underline{\mathrm{A}}(f(s_j, t_{i+1})) = 0. \quad (3.7)$$

Formally, Crank-Nicolson, where the parameter $\theta = \frac{1}{2}$, is then written as:

$$\frac{f_{i+1,j} - f_{i,j}}{\Delta t} + \theta\underline{\mathrm{A}}(f(s_j, t_i)) + (1-\theta)\underline{\mathrm{A}}(f(s_j, t_{i+1})) = 0. \quad (3.8)$$

And by inserting $\underline{\mathrm{A}}$ into (3.8), we can then rearrange the terms, isolating the known terms on one side and the unknown terms on the other side. This will allow us

to create a discrete (matrix) formulation for the terms of the first equation in the LCP.

$$\begin{aligned}&\frac{f_{i+1,j}-f_{i,j}}{\Delta t}+\theta\left(rs\frac{f_{i,j+1}-f_{i,j-1}}{2\Delta s}+\frac{1}{2}s_j^2\sigma^2\frac{f_{i,j+1}-2f_{i,j}+f_{i,j-1}}{\Delta s^2}-rf_{i,j}\right)+\\&+(1-\theta)\left(rs_j\frac{f_{i+1,j+1}-f_{i+1,j-1}}{2\Delta s}+\frac{1}{2}s_j^2\sigma^2\frac{f+1_{i,j+1}-2f_{i+1,j}+f_{i+1,j-1}}{\Delta s^2}-rf_{i+1,j}\right)=\\&=0.\end{aligned}\tag{3.9}$$

Recalling that $s_j = j\Delta s$ and substituting s_j and s_j^2 by $j\Delta s$ and $j^2(\Delta s)^2$ respectively, we get after some trivial algebra:

$$\begin{aligned}&f_{i,j-1}\left(-\Delta t\theta\sigma^2j^2+\frac{\Delta t\theta rj}{2}\right)+f_{i,j}\left(1+\Delta tr+\Delta t\theta\sigma^2j^2\right)+f_{i,j+1}\left(\frac{\Delta t\theta\sigma^2j^2+\Delta t\theta rj}{2}\right)=\\&=f_{i+1,j-1}\left(\Delta t(1-\theta)\sigma^2j^2-\frac{\Delta t(1-\theta)rj}{2}\right)+f_{i+1,j}\left(1-\Delta t(1-\theta)\sigma^2j^2\right)+\\&+f_{i+1,j+1}\left(\frac{\Delta t(1-\theta)\sigma^2j^2+\Delta t(1-\theta)rj}{2}\right),\end{aligned}\tag{3.10}$$

where the left hand side includes terms that are known at time step i (as we move backwards in time, $i+1$ is always known at time i), and the right hand side contains the unknown option values at that time. In the interest of legibility, we introduce some simplifying notation. Note that all of these 6 coefficients depend on j, and they are defined pointwise in the spatial direction. For $j = 1, 2, 3, \ldots, M-1$

$$\begin{aligned}a_j&=\left(\frac{-\Delta t\theta\sigma^2j^2+\Delta t\theta rj}{2}\right),\\b_j&=\left(1+\Delta tr+\Delta t\theta\sigma^2j^2\right),\\c_j&=\left(\frac{-\Delta t\theta\sigma^2j^2-\Delta t\theta rj}{2}\right),\\\alpha_j&=\left(\frac{\Delta t(1-\theta)\sigma^2j^2-\Delta t(1-\theta)rj}{2}\right),\\\beta_j&=\left(1-\Delta t(1-\theta)\sigma^2j^2\right),\\\kappa_j&=\left(\frac{\Delta t(1-\theta)\sigma^2j^2+\Delta t(1-\theta)rj}{2}\right).\end{aligned}\tag{3.11}$$

Reintroducing those into (3.10), we get:

$$a_jf_{i,j-1}+b_jf_{i,j}+c_jf_{i,j+1}=\alpha_jf_{i+1,j-1}+\beta_jf_{i+1,j}+\kappa_jf_{i+1,j+1}.\tag{3.12}$$

Further, we introduce a variable that collects the right hand side of the Crank-Nicolson formulation, $Z_{i+1,j} = \alpha_j f_{i+1,j-1} + \beta_j f_{i+1,j} + \kappa_j f_{i+1,j+1}$. In every time step, $Z_{i+1,j}$ contains only things that are either constant for every given grid point, or is known from the previous time step.

We can apply the above to formulate the entire system using matrices and vectors for the inner points of the grid. For this purpose, introduce the following vectors of length $M-1$, one for each $i = 0, 1, 2, \ldots, N-1$ ($i = N$ corresponds to the expiry time). Recall that the total number of grid points within a time step (i.e. a given i) is $M+1$, the total number of steps in between these points in the spatial direction is M, and that $j = 0$ and $j = M$ leads to spatial boundary points.

$$\boldsymbol{f}_i = \begin{pmatrix} f_{i,1} \\ f_{i,2} \\ f_{i,3} \\ \vdots \\ f_{i,M-2} \\ f_{i,M-1} \end{pmatrix}, \quad \boldsymbol{b_i} = \begin{pmatrix} a_1 f_{i,0} \\ 0 \\ 0 \\ \vdots \\ 0 \\ c_{M-1} f_{i,M} \end{pmatrix}, \quad \boldsymbol{g} = \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_{M-2} \\ g_{M-1} \end{pmatrix}, \quad \check{\boldsymbol{b}}_{\boldsymbol{i+1}} = \begin{pmatrix} \alpha_1 f_{i+1,0} \\ 0 \\ 0 \\ \vdots \\ 0 \\ \kappa_{M-1} f_{i+1,M} \end{pmatrix}, \tag{3.13}$$

where $\boldsymbol{f_i}$ is the option prices at time i (generally, these are the vectors of the unknowns as we move backwards in time through the entire grid). The vector $\boldsymbol{b_i}$ contains the boundary values, to match the matrix formulation to (3.12). The vector $\boldsymbol{g}$ contains the option based constraints. The vector $\check{\boldsymbol{b}}_{\boldsymbol{i}}$ also contains the boundary values, but is not used in the construction of the coefficient matrix, it is used in the construction of the known variables vector $\boldsymbol{Z}_{i+1}$. Using the previous notation,

$$\begin{aligned} \boldsymbol{Z}_{i+1} &= \begin{pmatrix} Z_{i+1,1} \\ Z_{i+1,2} \\ Z_{i+1,3} \\ \vdots \\ Z_{i+1,M-2} \\ Z_{i+1,M-1} \end{pmatrix} = \begin{pmatrix} \alpha_1 f_{i+1,0} + \beta_1 f_{i+1,1} + \kappa_1 f_{i+1,2} \\ \alpha_2 f_{i+1,1} + \beta_2 f_{i+1,2} + \kappa_2 f_{i+1,3} \\ \alpha_3 f_{i+1,2} + \beta_3 f_{i+1,3} + \kappa_3 f_{i+1,4} \\ \vdots \\ \alpha_{M-2} f_{i+1,M-3} + \beta_{M-2} f_{i+1,M-2} + \kappa_{M-2} f_{i+1,M-1} \\ \alpha_{M-1} f_{i+1,M-1} + \beta_{M-1} f_{i+1,M-1} + \kappa_{M-1} f_{i+1,M} \end{pmatrix} = \\ &= \begin{pmatrix} \beta_1 & \kappa_1 & 0 & 0 & \ldots & 0 \\ \alpha_2 & \beta_2 & \kappa_2 & 0 & \ldots & 0 \\ 0 & \alpha_3 & \beta_3 & \kappa_3 & \ldots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \alpha_{M-2} & \beta_{M-2} & \kappa_{M-2} \\ 0 & \ldots & 0 & 0 & \alpha_{M-1} & \beta_{M-1} \end{pmatrix} \begin{pmatrix} f_{i+1,1} \\ f_{i+1,2} \\ f_{i+1,3} \\ \vdots \\ f_{i+1,M-2} \\ f_{i+1,M-1} \end{pmatrix} + \begin{pmatrix} \alpha_1 f_{i+1,0} \\ 0 \\ 0 \\ \vdots \\ 0 \\ \kappa_{M-1} f_{i+1,M} \end{pmatrix} = \\ &= \boldsymbol{M} \boldsymbol{f}_{i+1} + \check{\boldsymbol{b}}_{i+1}. \end{aligned} \tag{3.14}$$

Note that the matrix $\boldsymbol{M}$, defined above, is actually constant through time. Similarly, the actual coefficient matrix $\boldsymbol{\mathcal{A}}$ can be defined according to:

$$\boldsymbol{\mathcal{A}} = \begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & 0 \\ 0 & a_3 & b_3 & c_3 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a_{M-2} & b_{M-2} & c_{M-2} \\ 0 & \dots & 0 & 0 & a_{M-1} & b_{M-1} \end{pmatrix}. \tag{3.15}$$

This matrix is also constant in time. We have now established the notation and performed the rudimentary calculations necessary to revisit the LCP and reformulate it in terms of matrices, completing the discretization. With $i = 0, 1, 2, \dots, N-1$, corresponding to the temporal discretization, we seek the vectors $\boldsymbol{f}_i$, one for each i starting from $N-1$ and working backwards to zero[4].

$$\begin{cases} \boldsymbol{\mathcal{A}}\boldsymbol{f}_i - \boldsymbol{M}\boldsymbol{f}_{i+1} - \check{\boldsymbol{b}}_{i+1} + \boldsymbol{b}_i \geq 0, \\ \boldsymbol{f}_i \geq \boldsymbol{g}_i, \\ (\boldsymbol{f}_i - \boldsymbol{g}_i)^T(\boldsymbol{\mathcal{A}}\boldsymbol{f}_i - \boldsymbol{M}\boldsymbol{f}_{i+1} - \check{\boldsymbol{b}}_{i+1} + \boldsymbol{b}_i) = 0, \end{cases} \tag{3.16}$$

where the inequalities and the equalities are taken componentwise of the vectors.

An alternative formulation

The purpose of this section is to present another formulation of (3.16), where we keep the time derivative and the space discretization visibly separate. The advantage of this form is that it is more readily generalizable, and it is easy to trace which parts the calculation relate to the choice of Crank-Nicolson, and which parts is the space discretization. This formulation is useful for understanding the operator splitting demonstrated later on, though implementing it would lead to more computational effort. We introduce the new coefficients according to:

$$\begin{aligned} \underline{\mathrm{a}}_j &= \left(\frac{-\sigma^2 j^2 + rj}{2} \right), \\ \underline{\mathrm{b}}_j &= \left(\sigma^2 j^2 \right), \\ \underline{\mathrm{c}}_j &= \left(-\frac{\sigma^2 j^2 - rj}{2} \right). \end{aligned} \tag{3.17}$$

[4]For illustrative purposes, one can imagine how we place each vector on top of the grid points as we step backwards through time, thus gradually adding columns to the inner discrete matrix (the points on three of the edges of the matrix are boundary points and known from the start, and the last boundary will contain the vector of answers for $t = 0$).

Using the above, we can express the space discretization with the matrix and vectors below.

$$\underline{\boldsymbol{A}} = \begin{pmatrix} \underline{\mathrm{b}}_1 & \underline{\mathrm{c}}_1 & 0 & 0 & \dots & 0 \\ \underline{\mathrm{a}}_2 & \underline{\mathrm{b}}_2 & \underline{\mathrm{c}}_2 & 0 & \dots & 0 \\ 0 & \underline{\mathrm{a}}_3 & \underline{\mathrm{b}}_3 & \underline{\mathrm{c}}_3 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \underline{\mathrm{a}}_{M-2} & \underline{\mathrm{b}}_{M-2} & \underline{\mathrm{c}}_{M-2} \\ 0 & \dots & 0 & 0 & \underline{\mathrm{a}}_{M-1} & \underline{\mathrm{b}}_{M-1} \end{pmatrix}, \quad \underline{\boldsymbol{b}}_i = \begin{pmatrix} \underline{a}_1 f_{i,0} \\ 0 \\ 0 \\ \vdots \\ 0 \\ \underline{c}_{M-1} f_{i,M} \end{pmatrix},$$

$$\check{\underline{\boldsymbol{b}}}_{i+1} = \begin{pmatrix} \underline{a}_1 f_{i+1,0} \\ 0 \\ 0 \\ \vdots \\ 0 \\ \underline{c}_{M-1} f_{i+1,M} \end{pmatrix}, \quad \boldsymbol{g} = \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_{M-2} \\ g_{M-1} \end{pmatrix}. \tag{3.18}$$

We can then express the first equation of the discretized LCP, (3.16), in a few different but equivalent forms. Formulating it using the control parameter θ as before, it takes the following form

$$\frac{\boldsymbol{f}_{i+1} - \boldsymbol{f}_i}{\Delta t} + \theta \underline{\boldsymbol{A}} \boldsymbol{f}_i + \theta r \boldsymbol{f}_i + \theta \underline{\boldsymbol{b}}_i + (1-\theta) \underline{\boldsymbol{A}} \boldsymbol{f}_{i+1} + (1-\theta) r \boldsymbol{f}_i + (1-\theta) \check{\underline{\boldsymbol{b}}}_{i+1} \geq 0. \tag{3.19}$$

We note that with the given definitions the following holds:

$$\begin{aligned} &\frac{\boldsymbol{f}_{i+1} - \boldsymbol{f}_i}{\Delta t} + \theta \underline{\boldsymbol{A}} \boldsymbol{f}_i + \theta r \boldsymbol{f}_i + \theta \underline{\boldsymbol{b}}_i + (1-\theta) \underline{\boldsymbol{A}} \boldsymbol{f}_{i+1} + (1-\theta) r \boldsymbol{f}_i + (1-\theta) \check{\underline{\boldsymbol{b}}}_{i+1} = \\ &= \boldsymbol{\mathcal{A}} \boldsymbol{f}_i - \boldsymbol{M} \boldsymbol{f}_{i+1} - \check{\boldsymbol{b}}_{i+1} + \boldsymbol{b}_i. \end{aligned} \tag{3.20}$$

We introduce the operator $\underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1})$ for a more compact notation, i.e. it takes the values at two consecutive time steps and the "current" time step as arguments, according to

$$\underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}) = \theta \underline{\boldsymbol{A}} \boldsymbol{f}_i + \theta r \boldsymbol{f}_i + \theta \underline{\boldsymbol{b}}_i + (1-\theta) \underline{\boldsymbol{A}} \boldsymbol{f}_{i+1} + (1-\theta) r \boldsymbol{f}_i + (1-\theta) \check{\underline{\boldsymbol{b}}}_{i+1}. \tag{3.21}$$

The alternative formulation of (3.16) in its entirety becomes

$$\begin{cases} \frac{\boldsymbol{f}_{i+1} - \boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}, i) \geq 0, \\ \boldsymbol{f}_i \geq \boldsymbol{g}, \\ (\boldsymbol{f}_i - \boldsymbol{g})^T (\frac{\boldsymbol{f}_{i+1} - \boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}, i)) = 0, \end{cases} \tag{3.22}$$

where the inequalities and the equalities hold for each element of the vectors, and $i = 0, 1, 2, \dots, N-2, N-1$.

Rannacher time stepping

An undesirable property of the Crank-Nicolson is that while it is unconditionally stable, it does not dampen sudden shocks particularly fast. If the boundary condition is "sufficiently non-smooth" the solution will exhibit oscillations, though these oscillations decrease in severity the more time steps we take - and in the relevant applications, dividends and payoffs give rise to such oscillations.[14]

One can however modify the Crank-Nicolson method to achieve better stability properties, as suggested by Rannacher. Rannacher's suggestion was to replace the first few time steps by pure backward Euler steps (i.e. solving those time steps implicitly only) and then continue with the full Crank-Nicolson.

In terms of implementation, this simply corresponds to changing the value of the averaging parameter θ to 1 for the first few steps, and then changing it back to $\frac{1}{2}$ for the remaining steps. Further research for European options suggests that the best results are reached if additional time steps are introduced, so that where in the unmodified Crank-Nicolson two steps are taken, with the Rannacher modification, we instead take four steps of implicit Euler [14].

The downside to this method is that even though we reduce the error from the oscillations, we may still reduce overall accuracy. The constraints of American options, and the algorithms that we need to use to address them, will influence the results of a Crank-Nicolson discretization. For one thing, the constraints should reduce the oscillations all on their own.

3.1.2 Backwards Differentiation Formula 2 - BDF2

The Euler backwards formula is really a special case of another method: Backwards Differentiation Formulas. The Euler backward is BDF1, and we briefly concern ourself with BDF2, a method which is second order accurate in time. We will not be as detailed as above, as the general principles are the same as for the Crank-Nicolson discretization. The primary reason for examining this method is the oscillations picked up by Crank-Nicolson in the presence of barriers and sharp boundaries: Such oscillations are dampened much faster with BDF2.
In practical terms, BDF2 will give rise to a different coefficient for the LHS coefficient matrix diagonal, and no matrix at all on the RHS, merely a combination of terms. BDF2 is implicit, but does not just include data from the previous time step, but from the time step before that as well (BDF3, predictably, includes 3 time steps). Expressed in the style of (3.6), the scheme looks as follows:

$$\frac{3}{2}\frac{(f_{i+1,j}-f_{i,j})}{\Delta t}-\frac{1}{2}\frac{(f_{i+2,j}-f_{i+1,j})}{\Delta t}+\underline{\mathrm{A}}f_{i,j}=0. \tag{3.23}$$

This leads to

$$\frac{3}{2}\frac{(f_{i+1,j}-f_{i,j})}{\Delta t}-\frac{1}{2}\frac{(f_{i+2,j}-f_{i+1,j})}{\Delta t}+ \\ +\left(rs_j\frac{f_{i,j+1}-f_{i,j-1}}{2\Delta s}+\frac{1}{2}s_j^2\sigma^2\frac{f_{i,j+1}-2f_{i,j}+f_{i,j-1}}{\Delta s^2}-rf_{i,j}\right)=0. \tag{3.24}$$

Which in turn can be reformulated to

$$f_{i,j-1}\left(-\Delta t\sigma^2j^2+\frac{\Delta trj}{2}\right)+f_{i,j}\left(\frac{3}{2}+\Delta tr+\Delta t\sigma^2j^2\right)+f_{i,j+1}\left(\frac{\Delta t\sigma^2j^2+\Delta trj}{2}\right)= \\ =2f_{i+1,j}-f_{i+2,j}, \tag{3.25}$$

where we can see that the RHS is very similar to the one produced by Crank-Nicolson if we set $\theta = 1$, turning the Crank-Nicolson scheme the implicit Euler, and replace a term in the diagonal coefficient b_j: 1 is now $\frac{3}{2}$.

Implementing the BDF2 discretization is then not very different from Crank-Nicolson with a Rannacher start up, we need merely modify some coefficients. As it is so easy to move from Crank-Nicolson to BDF2, we will continue to use Crank-Nicolson as the "primary" scheme to illustrate relevant formulae.

Chapter 4

Describing the finite difference methods examined

This chapter will explore the finite difference methods examined in this thesis. The first method is an iterative solver that approximates the solution of the system of equations arising from the LCP, and the other two methods involve direct solution of the same system. We will examine what is needed to implement them, and some practical concerns raised during the course of actual implementation.

4.1 Projected Successive Over Relaxation, PSOR

Projected Successive Over Relaxation (PSOR) is an iterative method. At each time step, the solution for that time step is arrived at by an iterative algorithm which refines a solution until the change between iterations is below a previously specified error tolerance. The current solution is then said to be the solution for that time step, and passed to the next time step where the process is repeated.

The easiest way to explain how to implement PSOR algorithm is to explain first the Jacobi iterative method, then generalize that method to the Gauss-Seidel method, which is then in turn generalized to the SOR method. [25, p. 150-160] [22, p. 99-100]

4.1.1 Jacobi and Gauss-Seidel for European options

Recall the system (3.16), for now as in the European formulation without the American constraints. That is, we turn the first equation in the LCP into an equality, and toss the constraints for all times except $t = T$. If we examine one equation in the arising system, we will find (4.2), using the previously introduced notation and boundary conditions.

$$a_j f_{i,j-1} + b_j f_{i,j} + c_j f_{i,j+1} = Z_{i+1,j}. \tag{4.1}$$

We rearrange (4.2) so that we can study the element $f_{i,j}$, according to

$$f_{i,j} = \frac{1}{b_j} \left(Z_{i+1,j} - a_j f_{i,j-1} - c_j f_{i,j+1} \right). \tag{4.2}$$

This formulation leads us to the Jacobi method of iteratively solving the system, starting from $j = 1$ and moving to $j = M - 1$. By taking an initial guess on the values of $f_{i,j-1}$ and $f_{i,j+1}$ on the right hand side, we will then be able to refine that guess using (4.2). For the first iteration in each time step, this guess is taken to be the value from the previous time step. Formally, if we introduce the superscript k to denote the iteration numbering, with $k + 1$ being the current iteration, k being the previous iteration and $k = 0$ means we take the value from the previous time step, we can use (4.2) to express the actual algorithm.

$$f_{i,j}^{k+1} = \frac{1}{b_j} \left(Z_{i+1,j} - a_j f_{i,j-1}^{k} - c_j f_{i,j+1}^{k} \right). \tag{4.3}$$

The above formula is then used for each j. This is then repeated until the desired accuracy has been reached, as defined by an error measure, for example the below where $\boldsymbol{f_i}$ refers to the entire vector of inner point space discretized variables in the time step i:

$$||\boldsymbol{f_i}^{k+1} - \boldsymbol{f_i}^{k}||^2 = \sum_{j=1}^{M-1} \left(f_{i,j}^{k+1} - f_{i,j}^{k} \right)^2. \tag{4.4}$$

Once we are satisfied that (4.4) is small enough, we say that $f_{i,j} \approx f_{i,j}^{k+1}$ for all j and cease the iterations, and move to the next time step. In this new time step (i.e. i has changed) we then set $f_{i,j}^{0} = f_{i+1,j}^{k+1}$ and repeat the process for each of the time steps necessary.

That was the Jacobi iterated solution[25, p. 150-152], whereby an algebraic calculation of each point in the space grid is repeated within every time step. The Gauss-Seidel method is similar, except we utilize the fact that in (4.3), when the time has come to perform the calculation at $j = 2, 3, 4, \ldots, M - 1$, we have already calculated a value for $j-1$ within the current iteration $(k+1)$. Thus, we can replace $f_{i,j-1}^{k}$ with $f_{i,j-1}^{k+1}$. In simple terms, in the Jacobi method we carry out the approximating calculation for the entire vector of space points within each iteration. In Gauss-Seidel, instead of first calculating the entire vector in a given iteration and using those elements in the next iteration, we use the elements of the vector as soon as they become available within the current iteration. Formally, (4.3) is modified to

$$f_{i,j}^{k+1} = \frac{1}{b_j} \left(Z_{i+1,j} - a_j f_{i,j-1}^{k+1} - c_j f_{i,j+1}^{k} \right). \tag{4.5}$$

The calculation is then carried out in the same way as for the Jacobi method, repeated until the approximation error has reached the desired level.

4.1.2 Successive Overrelaxation (SOR) for European options

In SOR, the Gauss-Seidel method is further modified to bring about faster convergence of the approximating iterates to the actually sought solution, reducing the number of iterations necessary to reach the desired level of approximation. This is accomplished by averaging the current Gauss-Seidel iterate with the previous Gauss-Seidel iterate, using a weighing parameter ω.

$$\begin{aligned} \tilde{f}_{i,j}^{k+1} &= \frac{1}{b_j}\left(Z_{i+1,j} - a_j f_{i,j-1}^{k+1} - c_j f_{i,j+1}^{k}\right), \\ f_{i,j}^{k+1} &= \omega \tilde{f}_{i,j}^{k+1} + (1-\omega) f_{i,j}^{k}, \end{aligned} \tag{4.6}$$

where $\tilde{f}$ denotes a temporary variable used only for illustrative purposes. We use ω to denote the overrelaxation parameter. The optimal value of ω depends on the problem at hand, and while an optimal value in the interval $1 < \omega < 2$ can be shown to exist, locating the relevant value for a given problem requires more effort than adjusting ω in each time step to reduce the number of iterations in the next time step, as suggested in [25, p. 153]. We will use the dynamic adjustment suggested in [25, p. 153], where the number of iterations taken with time step i and $i+1$ are tracked, and ω is either increased or decreased depending on which was greater, so that the new value is used in time step $i-1$. The size of the change to ω is a fixed value, chosen as in [22, p. 99-100] to be 0.05, though we never allow ω to drop below 1.

4.1.3 Projected Successive Overrelaxation (PSOR) for American options

Transition from SOR to PSOR is the easiest of the previous steps, in terms of practical implementation. Recall from the formulation of the LCP (3.16) that $\boldsymbol{g}_i$ is the vector of the constraints for each spatial point, at the time i. We simply modify (4.6) according to:

$$\begin{aligned} \tilde{f}_{i,j}^{k+1} &= \frac{1}{b_j}\left(Z_{i+1,j} - a_j f_{i,j-1}^{k+1} - c_j f_{i,j+1}^{k}\right), \\ f_{i,j}^{k+1} &= \max\left(g_j \ , \ \omega \tilde{f}_{i,j}^{k+1} + (1-\omega) f_{i,j}^{k}\right). \end{aligned} \tag{4.7}$$

That is, within every time step at every element within an iteration, we enforce the constraint $f_{i,j}^{k+1} \geq g_j$. Once we have stepped through all the time steps, we will then have our answer.

4.1.4 PSOR with BDF2

The BDF2 implementation involves replacing $Z_{i+1,j}$, the right hand side element, with

$$Z_{i,j} = 2f_{i+1,j} - \frac{1}{2} f_{i+2,j}. \tag{4.8}$$

The right hand side for the implicit start up step becomes $Z_{i,j} = f_{i+1,j}$.

The coefficients, a, b, c are found by setting $\theta = 1$, but for the BDF2 steps we also add $\frac{1}{2}$ to b. To see why, study the middle coefficient of the right hand side in (3.25) and compare it to the same coefficient from (3.10).

4.1.5 Comments on PSOR

Iterative Solver

One advantage of the iterative solver is that the practical implementation does not change much if the complexity of the linear system increases. In a non-tridiagonal setting, additional terms appear in 4.1 and all of the subsequent equations, but no fundamental change needs to be made to the implementation. This is not the case for direct solvers treated for the remaining two methods: A non-tridiagonal system would have a significant adverse impact on calculation time.

Omega

As was briefly mentioned above, selection of the weighing parameter ω is not entirely trivial. In [25], it is suggested that one get around this theoretical problem by simply adjusting the value of ω between iterations. This is the approach that has been employed here. It can be characterized as follows:

Set a $\Delta\omega$, here and in [25] chosen as 0.05 unless otherwise specified. If the number of loops carried out in this time step is larger than the number of loops carried out in the previous time step, we alter ω by subtracting or adding $\Delta\omega$. Practically, this is done with an if-statement that changes the sign of $\Delta\omega$, and then it is always added to ω, resulting in an increase or decrease. An additional "safety check" is performed to ensure that ω cannot drop below 1 - if it drops below 1 we instead have "underrelaxation" which should not be better then standard Gauss-Seidel.[24, p. 866]

Effects of finer grids

As noted in [17], PSORs convergence will degrade for finer grids. In order to maintain convergence as the grid gets finer, the error tolerance needs to be lowered. This has an adverse impact on the calculation time.

4.2 Interlude: LU-decomposition

The two FD methods treated below involves the direct solution of a linear system. Solving that system is an important part of the calculation, as its contribution to the total calculation time is significant. Primarily, we will concern ourself with a solver algorithm specifically adapted to solve tridiagonal systems, as they are the

ones relevant for our examples, and solving them is faster than solutions of full systems.[1]

LU decomposition means we decompose a matrix into two matrices that when multiplied with each other returns the original matrix, with the additional property that one of the decomposed matrices is lower triangular and the other is upper triangular (the former has only zeros above the diagonal, the latter below the diagonal). The lower triangular matrix is traditionally denoted L and the upper triangular matrix is traditionally denoted U.

In the appendix A.4, we demonstrate in greater detail how to use LU-decomposition to solve a tridiagonal system of linear equations, using an algorithm known as the Tridiagonal Matrix Algorithm, TDMA, from [24, p. 50-51]. There are two primary properties of TDMA that we need to take note of.

The system must be tridiagonal. If this is not the case, we must employ more general Gaussian elimination techniques that are more computationally expensive.

If the matrix we seek to invert is constant in time, a portion of the TDMA can be carried out once outside the time stepping procedure. Not repeating this calculation for every time step reduces computational cost.

4.3 Operator splitting

The basic idea of operator splitting can be readily illustrated using (1.5) as an example. Recalling then (1.5),

$$\begin{aligned}
&\frac{\partial f}{\partial t} + rs\frac{\partial f}{\partial s} + \frac{1}{2}s^2\sigma^2\frac{\partial^2 f}{\partial s^2} = rf, \\
&\frac{\partial f}{\partial t} = -rs\frac{\partial f}{\partial s} - \frac{1}{2}s^2\sigma^2\frac{\partial^2 f}{\partial s^2} + rf, \\
&\frac{\partial f}{\partial t} + A(f) = 0, \quad \text{where } A(f) = rs\frac{\partial f}{\partial s} + \frac{1}{2}s^2\sigma^2\frac{\partial^2 f}{\partial s^2} - rf.
\end{aligned} \tag{4.9}$$

But with the addition of a European constraint, i.e. a constraint at $t = T$, denoted once more by g. Let us examine the operator A, and noting that we could easily

[1] We note however, that not all models for the underlying leads to tridiagonal systems, even if we use the same discretization scheme.

separate it into multiple operators of less complex structure

$$A(f) = rs\frac{\partial f}{\partial s} + \frac{1}{2}s^2\sigma^2\frac{\partial^2 f}{\partial s^2} - rf$$
$$A(f) = (A_1 + A_2 + A_3)(f), \text{ where } \begin{cases} A_1(f) = rs\frac{\partial f}{\partial s} \\ A_2(f) = \frac{1}{2}s^2\sigma^2\frac{\partial^2 f}{\partial s^2} \\ A_3(f) = -rf \end{cases} \tag{4.10}$$

(Note that this notation does not survive this example, we will redefine A_1 and A_2 later!) This is a componentwise splitting, whereby we have split the operator A, creating multiple smaller subproblems[2] - though the exact nature of these subproblems is not contained in the above, as they depend on the selection of a calculation scheme. There are many other ways in which the split could be performed, depending on which properties one wishes to use or discard for a particular calculation scheme (an overview of many such methods is provided in [6] and the references therein).

In general however, one seeks to approximate an operator with a sum or a product of operators, that when recombined according to an appropriate algorithm, are more susceptible to numerical methods. Depending on the problem at hand this can be used to deconstruct the integrands into components more easily integrated. One could also perform the splitting on the discretized operator matrices, decomposing a non-tridiagonal matrix into a succession of tridiagonal matrices (using permutations on the separated problems to make them tridiagonal) so that TDMA can be applied.

For our particular problem, the goal is to use the splitting so as to allow us to treat the early-exercise constraint in one step, and perform a "bulk" calculation in the other. This is similar to one possible approach to Navier-Stokes equations for incompressible fluids, and can be seen in [12, Ch. 2].

4.3.1 What Ikonen-Toivanen suggested

First, we will describe the method proposed in [13], which is based on operator splitting. After we have examined it and its practical implementation, we will look closer at the theoretical background, and demonstrate that their proposal is in fact the Douglas-Rachford algorithm. Recall the discretized LCP in matrix and vector form, (3.16). We will now modify this with an additional variable, λ, which introduces an additional constraints to maintain equivalence. With $\boldsymbol{\lambda}_i$ corresponding to a vector of size $M-1$, with one λ for each grid point (i.e. in reality, we are introducing $\lambda_{i,j}$ when we discretize λ)

[2] We will refer to these subproblems as fractional time steps, but no actual time advancement occurs between subproblems!

$$\begin{cases} \boldsymbol{\mathcal{A}}\boldsymbol{f}_i - \boldsymbol{M}\boldsymbol{f}_{i+1} - \check{\boldsymbol{b}}_{i+1} + \boldsymbol{b}_i = \Delta t \boldsymbol{\lambda}_i, \\ \boldsymbol{f}_i \geq \boldsymbol{g}, \\ \boldsymbol{\lambda}_i \geq 0, \\ (\boldsymbol{f}_i - \boldsymbol{g}_i)^T(\boldsymbol{\lambda}_i) = 0. \end{cases} \tag{4.11}$$

What Ikonen and Toivanen suggested in their first article is splitting the LCP in (4.11) into two smaller subproblems, each of which is solved in every time step. The first problem is an unconstrained linear system, using $\boldsymbol{\lambda}_{i+1}$. The second problem is a LCP, but one that is uncoupled - each element can be solved for separately.

Solving the first fractional time step

Using the above, we can write the first step of the operator splitting method according to[3]

$$\boldsymbol{\mathcal{A}}\tilde{\boldsymbol{f}}_i = \boldsymbol{M}\boldsymbol{f}_i + \check{\boldsymbol{b}}_{i+1} - \boldsymbol{b}_i + \Delta t \boldsymbol{\lambda}_{i+1}. \tag{4.12}$$

At time i, the only unknown in the system above is $\tilde{\boldsymbol{f}}_i$. For the very first time step ($i = N - 1$), we set all the $\lambda_{N,j}$ to zero. If $\boldsymbol{\mathcal{A}}$ is tridiagonal, we can safely use the TDMA to solve the system, and acquire $\tilde{\boldsymbol{f}}_i$ so that it is known for the second step.

Solving the second fractional time step

In the second step, we solve the problem

$$\begin{cases} \boldsymbol{f}_i - \tilde{\boldsymbol{f}}_i - \Delta t \left(\boldsymbol{\lambda}_i - \boldsymbol{\lambda}_{i+1}\right) = 0, \\ \boldsymbol{f}_i \geq \boldsymbol{g}, \\ \boldsymbol{\lambda}_i \geq 0, \\ (\boldsymbol{f}_i - \boldsymbol{g})^T(\boldsymbol{\lambda}_i) = 0, \end{cases} \tag{4.13}$$

where all equalities and inequalities are taken componentwise, which means that each element is uncoupled from the others.

Every component in the vectors above corresponds to its own LCP, with two unknowns for each problem: $f_{i,j}$ and $\lambda_{i,j}$. For ease of reference, we examine one of

[3]Note that $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{M}$ are constant matrices through time, unless Rannacher time stepping is used, in which case they will be different for the first four calculations. In some texts, the intermediate solution variable is denoted as a fractional time step, so $\tilde{\boldsymbol{f}}_i = \boldsymbol{f}_{i+\frac{1}{2}}$

these problems in isolation[4].

$$\begin{cases} f_{i,j} - \tilde{f}_{i,j} - \Delta t\,(\lambda_{i,j} - \lambda_{i+1,j}) = 0, & \text{(1st condition)} \\ f_{i,j} \geq g_j, & \text{(2nd condition)} \\ \lambda_{i,j} \geq 0, & \text{(3rd condition)} \\ (f_{i,j} - g_j)(\lambda_{i,j}) = 0, & \text{(4th condition)} \end{cases} \tag{4.14}$$

Solving the problem is done by observing that the fourth condition requires that either $\lambda_{i,j} = 0$ or $f_{i,j} = g_j$ [5]. The $\lambda_{i,j}$ can in other words be non-zero only when early exercise occurs, though they can be zero even though early exercise occurs.

Hence, we begin by assuming that $\lambda_{i,j} = 0$ holds and insert this into the first equation in (4.14). The equation with two unknowns is then reduced to having a single unknown, and by straightforward algebraic manipulation, one can reach an expression for $f_{i,j}$ according to

$$f_{i,j} = \tilde{f}_{i,j} - \Delta t \lambda_{i+1,j}. \tag{4.15}$$

We have then used 3 of the 4 conditions: By setting it to zero, the third and fourth conditions are automatically fulfilled, and the first condition gives rise to (4.15). If this solution satisfies the last condition, $f_{i,j} \geq g_j$, then the solution to the problem (4.14) has been found, according to

$$\begin{cases} \lambda_{i,j} = 0, & \text{(By assumption)} \\ f_{i,j} = \tilde{f}_{i,j} - \Delta t \lambda_{i+1,j} \end{cases} \tag{4.16}$$

If this does not satisfy $f_{i,j} \geq g_j$, then this is not the solution to (4.14) and instead of assuming a value for $\lambda_{i,j}$, we assume $f_{i,j} = g_j$. In perfect analogue to the previous assumption, this guarantees that the second and fourth condition are satisfied. Then, we use the first condition to acquire an expression for $\lambda_{i,j}$ according to $\lambda_{i,j} = \frac{g_j - \tilde{f}_{i,j} + \Delta t \lambda_{i+1,j}}{\Delta t}$. If this value of $\lambda_{i,j}$ satisfies the third condition, then all conditions are fulfilled and the solution to (4.14) is given by

$$\begin{cases} f_{i,j} = g_j & \text{(By assumption)} \\ \lambda_{i,j} = \frac{g_j - \tilde{f}_{i,j} + \Delta t \lambda_{i+1,j}}{\Delta t}. \end{cases} \tag{4.17}$$

[4]Readers familiar with optimization might note that these are essentially restatements of the Karush-Kuhn-Tucker conditions (KKT-conditions), which are first order necessary conditions for optimality within nonlinear optimization. Employing the KKT conditions allows one to examine a nonlinear optimization problem by examining a problem that looks exactly like each of the problems in (4.13). The λ is in fact what would be called a "KKT-multiplier". This traces back to equivalent formulations of the free boundary problem: In addition to the variational inequality that is equivalent to the LCP, variational inequalities can also have equivalences in optimization problems.

[5]It is this *complementarity* of these two expressions that gives the LCP its name.

One of these two possible solutions is the real solution to (4.14), and thus solving the second fractional time step of Ikonen-Toivanens operator splitting method is simply a matter of checking which of (4.16) or (4.17) satisfies the necessary constraints for each component of the vectors in (4.13).

Once both the first and second fractional steps have been solved, both $\boldsymbol{f}_i$ and $\boldsymbol{\lambda}_i$ have been found, and we are ready to proceed to $i-1$ and repeat the procedure.

The reader interested in more complex LCPs can consult [19], a general reference work regarding LCPs and their solution.

4.3.2 Theoretical Background, rediscovering Douglas-Rachford

Previous publications in the field of computational finance related to the operator splitting method described above are vague as to *why* this method works. This section aims to demonstrate that the method suggested by Ikonen and Toivanen has solid theoretical foundations, by demonstrating its equivalence to the Douglas-Rachford algorithm. The interested reader should consult [5], which demonstrates the convergence of the Douglas-Rachford algorithm.

For an operator split to be useful, the algorithm used on the split operators must converge. In addition to [5] an overview of several such algorithms can be found in [6] or [12, Ch. 2] which contains specific problems related to ours. From the latter, we introduce the Douglas-Rachford scheme, an alternating direction scheme[6] originally suggested in [1] and reviewed in [12, p. 60-63]. First, it will be useful to recall (3.21), the alternative formulation of the discretization. Reintroducing the operator $\underline{\boldsymbol{Q}}$:

$$\underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}) = \theta \underline{\boldsymbol{A}} \boldsymbol{f}_i + \theta r \boldsymbol{f}_i + \theta \underline{\boldsymbol{b}}_i + (1-\theta)\underline{\boldsymbol{A}}\boldsymbol{f}_{i+1} + (1-\theta) r \boldsymbol{f}_i + (1-\theta)\check{\underline{\boldsymbol{b}}}_{i+1}, \tag{4.18}$$

The λ modified LCP in (4.11) can now be rewritten as

$$\begin{cases} \frac{\boldsymbol{f}_{i+1}-\boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}) - \boldsymbol{\lambda}_i = 0, \\ \boldsymbol{f}_i \geq \boldsymbol{g}, \\ \boldsymbol{\lambda}_i \geq 0, \\ (\boldsymbol{f}_i - \boldsymbol{g})^T(\boldsymbol{\lambda}_i) = 0. \end{cases} \tag{4.19}$$

We will now formulate the Douglas-Rachford scheme. For clarity, we reuse the intermediate variable $\tilde{\boldsymbol{f}}_i$, and the algorithm then looks as in (4.20) (with $i = N, N-$

[6] Alternating direction scheme in this context means that one operator is used implicitly and the other explicitly. A more suggestive way of viewing the algorithm to see this is presented in the appendix, A.6

$1, N-2, \ldots, 2, 1, 0$ and $i = N$ being the known boundary at expiry, just as before)[7]

$$\begin{cases} \frac{\tilde{\boldsymbol{f}}_i - \boldsymbol{f}_{i+1}}{\Delta t} + A_1(\tilde{\boldsymbol{f}}_i, \boldsymbol{f}_{i+1}) + A_2(\boldsymbol{f}_{i+1}) = 0, \\ \frac{\boldsymbol{f}_i - \tilde{\boldsymbol{f}}_i}{\Delta t} + A_2(\boldsymbol{f}_i) - A_2(\boldsymbol{f}_{i+1}) = 0 \end{cases} \tag{4.20}$$

where A_1 and A_2 are the split operators. The convergence of this scheme was originally proven in [1], for a componentwise split on the multidimensional heat equation. The scheme has first order accuracy. Unfortunately, the LCP in (4.19) places much stricter demands on the nature of A_1 and A_2 than the componentwise split Douglas-Rachford originally performed - in fact, the LCP means that the computationally useful splits requires one of the operators to be multi-valued, i.e. return a *set* of values rather than a single value. For a intuitive grasp of why this may be, consider the inequalities in the LCP. All together, they give rise to a unique value, the solution to the problem, but the purpose of the split is to allow treatment of the inequalities separately. If we split the operators we can no longer guarantee that there will be a single value that satisfies the constraints for the split operators - the operator will become multi-valued and a return a *set* of values of satisfying the criteria contained in the operator.

Fortunately, in [5], Lions and Mercier demonstrates that the scheme converges as long as A_1 and A_2 are "maximal monotone operators", *even if* they are multivalued operators. [8] We briefly state the requirements for monotonicity, as the convergence of the algorithm hinges on it.

If an operator A_o operates on a Hilbert space H it can be characterized as a subset of $H \times H$, in which case we can use (x, y) to describe the map - x is the argument, and y is the result after the operator has operated on x. With $\langle \cdot, \cdot \rangle$ denoting the inner product of H, an operator A_o is monotone if the following holds, as in [7]

$$\langle x' - x, y' - y \rangle \leq 0, \ \forall (x, y), (x', y') \in A_o, \tag{4.21}$$

Further, it is a maximal monotone operator if no other operator properly contains it.

The first operator

Armed with the assurance of convergence from [5], we then only need to define the operators A_1 and A_2 in such a way as to actually solve our original problem. The

[7] The difference between this formulation and the one on page 63 in [12] is that we count time backwards rather than forward. This means that $i+1$ is known at step i, where as Glowinski uses n for the previous and known step and $n+1$ for the current and unknown step.

[8] One can then demonstrate, as in [7], that the generalized Douglas-Rachford algorithm is really a variation of the proximal point algorithm or as in [23, p. 617-622] that it is equivalent to an Augmented Lagrangian algorithm. Interestingly, many different approaches to different problems over the years turn out to have yielded the same underlying method in different forms.

first operator, as can be readily seen from Ikonen-Toivanen, is simply:

$$A_1(\tilde{\boldsymbol{f}}_i, \boldsymbol{f}_{i+1}) = \underline{\boldsymbol{Q}}(\tilde{\boldsymbol{f}}_i, \boldsymbol{f}_{i+1}). \tag{4.22}$$

That is, the first operator is the space discretization.

Apart from the computational benefits this infers (as outlined above), this also makes it relatively easy to perform modifications - if we wish to apply a different space discretization, we are free to do so by modifying only this operator. (We note, but do not further explore, that the domain of this operator is a Sobolev space).

The second operator

The overall goal of this section will be to define the second operator, and then to show how this operator in conjunction with the first operator, inserted into Douglas-Rachford gives rise to the method presented by Ikonen-Toivanen - which then connects the theoretical work of Lions and Mercier to the method suggested by Ikonen and Toivanen. We begin by presenting the foundation needed to explain the second operator.

First, we define a set of functions satisfying the early exercise constraint. The definition is due [5], and we will not attempt to explain every detail of it. We set $\Omega = [0, \infty) \times [0, T]$ to be the domain on which the original problem (1.6) is defined, and introduce the domain $\Omega_s = [0, s_{\max}]$, i.e. the discretized spatial domain. In order to strike a balance between generalizations and our practical problem, we characterize the functions by f. At each discretized time step, we examine all possible functions f from $L^2(\Omega_s)$. Defining the set of functions satisfying the early exercise constraint as

$$Z = \{f \in L^2(\Omega_s) | f \geq g \text{ a.e. } s \in \Omega_s\}, \tag{4.23}$$

where g is simply the function describing the early exercise constraint for that time step. To be clear, we are examining the problem at each time step, and the arbitrary functions f will change with the time step[9].

Next we introduce an indicator function on the set Z, $I_Z : L^2(\Omega_s) \to \mathbb{R} \cup \{+\infty\}$ so that

$$I_Z(f) = \begin{cases} 0, & f \in Z, \\ +\infty, & f \notin Z. \end{cases} \tag{4.24}$$

Because we will require it in the next section, it is useful to demonstrate that this function is in fact convex. Drawing once more from [21, p. 24-25][10], we note that I_Z

[9] One could also imagine options under which g is time dependant as well, in which case a new set Z would be defined in each time step.

[10] Page 24 in [21] contains the useful rules of arithmetic calculations involving infinities we will employ, and page 25 in [21] lists the relevant theorems from which we draw our test.

is a convex function if and only if the following holds for all $f_a \in L^2(\Omega_s)$, $f_b \in L^2(\Omega_s)$

$$I_Z((1-d)f_a + df_b) \leq (1-d)I_Z(f_a) + dI_Z(f_b), \tag{4.25}$$

for all $d \in [0,1]$. If either $f_a \notin Z$ or $f_b \notin Z$, then the RHS side of (4.25) turn into infinity, which is trivially satisfied. If both $f_a \in Z$ and $f_b \in Z$, we know that $f_a \geq g$ and $f_b \geq g$. Then

$$(1-d)f_a + df_b \geq (1-d)g + dg = g, \tag{4.26}$$

and hence, $I_Z((1-d)f_a + df_b) = 0$. Thus, I_Z is a convex function.

Subdifferentials

The next step requires the concept of subdifferentials, for which we will provide a shorter overview. The interested reader can consult for example [21, p. 213-226] for a more detailed description. We denote the subdifferential of a function v by ∂v.

Informally, the subdifferential is a set *of sets* that satisfy a "relaxed" definition of a classical derivative, a generalization of the regular gradient called a *subgradient*, shown in (4.27).

We illustrate this with a small example, using $v(x) = |x|$. At $x = 0$, this function has no derivative, but it has a subdifferential at that point, corresponding to all values in the interval $[-1, 1]$, so $\partial v(0) = \{a | a \in [-1, 1]\}$. If $x > 0$ then $\partial v(x) = \{1\}$, and if $x < 0$ $\partial v(x) = \{-1\}$. The subdifferential then is set valued, and contains information for all possible tangential functions at a certain point.[11] While this way of illustrating a subdifferential is somewhat lacking in formalism, it serves as a useful illustration.

Formally, we employ the following definition due [21, p. 214].

Definition Let V be a vector space equipped with an inner product $\langle \cdot, \cdot \rangle$ and let $F : V \to \mathbb{R} \cup +\infty$ be a convex function.

A vector $v \in V$ is a *subgradient vector*, for the function F at the point $x \in V$, if the subgradient inequality, shown below, is satisfied for all $y \in V$

$$F(y) \geq F(f) + \langle v, (y - f) \rangle, \quad \forall y \in V. \tag{4.27}$$

The *subdifferential* of the function F at the point $x \in V$, denoted $\partial F(x)$, is the set of all subgradient vectors for the function F at the point x. □

[11] Note that if the function is differentiable at a point, the subdifferential at that point is simply a set with one element - the derivative in the regular sense.

Using the above, we introduce $\partial I_Z(f)$. Our examined function is then I_Z instead of F, and Domain(F) is then Domain(I_Z) $= L^2(\Omega_s)$, and each subgradient vector v at the point given by $I_Z(f)$ must satisfy

$$I_Z(y) \geq I_Z(f) + \langle v, (y - f) \rangle, \quad \forall y \in L^2(\Omega_s). \tag{4.28}$$

Lemma 4.3.1 *The entire subdifferential of the indicator function on a convex set can then be expressed as*

$$\partial I_Z(f) = \begin{cases} \{v \in L^2(\Omega_s) | \, \langle v, y - f \rangle \leq 0, \quad \forall y \in Z\} & f \in Z, \\ \emptyset & f \notin Z, \end{cases} \tag{4.29}$$

Proof Assume $f \notin Z$. Then, in (4.28), $I_Z(f) = \infty$, but as $I_Z(y)$ when $y \in Z$ equals zero, then (4.28) is *never* satisfied, hence there are *no* subgradients, and thus the set of all possible subgradients is empty. This proves the second case.

Assume $f \in Z$. Then consider the following: $\langle v, y - f \rangle \leq 0$ is equivalent to (4.28), as I_Z only takes the values $+\infty$ and 0. This leads to two possible cases, one in which $\langle v, y - f \rangle$ is required to be smaller than infinity (which is always satisfied for all choices of v from L^2) and occurs when $y \notin Z$. The other case is where $\langle v, y - f \rangle$ is required to be smaller than zero, arising from $y \in Z$. As the requirement must hold for *all* $y \in L^2(\Omega_s)$, ≤ 0 is sufficient. This proves the first case. □

Characterizing the subdifferential

Recalling that the objective is to demonstrate how Ikonen-Toivanen is the generalized Douglas-Rachford algorithm for suitably chosen operators, we will examine the subdifferential of the indicator function closer. We begin by stating the complete characterization of the subdifferential of the indicator function,

Lemma 4.3.2 *Let* $Z = \{f \in L^2(\Omega_s) | f(s) \geq g(s)$ *a.e.* $s \in \Omega_s\}$*, and taking the indicator function* I_Z *defined according to*

$$\partial I_Z(f) = \begin{cases} \{v \in L^2(\Omega_s) | \, \langle v, y - f \rangle \leq 0, \quad \forall y \in Z\} & f \in Z, \\ \emptyset & f \notin Z, \end{cases} \tag{4.30}$$

we may then further refine the characterization in (4.30)*. Dividing* Ω_s *into the intervals* Q*,* R *and* P*, such that* $Q \cup R \cup P = \Omega_s$*, and say that all* s *for which* $f(s) < g(s)$ *are in* P*, all* s *for which* $f(s) = g(s)$ *are in* Q *and all* s *for which* $f(s) > g(s)$ *are in* R*. Then*

$$\partial I_Z(f) = \begin{cases} \emptyset, & P \neq \emptyset, \\ \{v \in L^2(\Omega_s) | v(s) = 0, \ \forall s \in R \text{ and } v(s) \leq 0, \ \forall s \in Q\}, & P = \emptyset. \end{cases} \tag{4.31}$$

Remark Thus, if $R = \Omega_s$, then $\partial I_Z(f) = \{0\}$.

Proof We prove this by examining the different possible cases for the arguments to the indicator function $f(s) < g(s)$, $f(s) > g(s)$ and $f(s) = g(s)$ for all $s \in \Omega_s$. For the remainder of this section, we will abbreviate the above notation using f, g, y and v for $f(s)$, $g(s)$, $y(s)$ and $v(s)$ (they are all functions from $L^2(\Omega_s)$).

If $P \neq \emptyset$ holds, then $f \notin Z$ and (4.29) tells us that $\partial I_Z(f) = \emptyset$. To treat the second case, assume now that $P = \emptyset$.

We then know that (4.29) must hold, so $\partial I_Z(f) = \{v \in L^2(\Omega_s) | \langle v, y - f \rangle \leq 0, \quad \forall y \in Z\}$. Using the inner product of $L^2(\Omega_s)$, we then have

$$0 \geq \int_{\Omega_s} v(s) \left(y(s) - f(s)\right) ds. \tag{4.32}$$

We can separate the area of integration into two or more intervals. In the shown case below, we use two intervals, the first from $[0, a]$ and the second from $[a, s_{\max}]$. The union of the intervals equals Ω_s, under the assumption that $P = \emptyset$.

$$0 \geq \int_{\Omega_s} v(s) \left(y(s) - f(s)\right) ds = \int_0^a v(s) \left(y(s) - f(s)\right) ds + \int_a^{s_{\max}} v(s) \left(y(s) - f(s)\right) ds \tag{4.33}$$

We assume that we are able to choose a such that all $s \in [0, a]$ is in Q, i.e. $f(s) = g(s)$ for all $s \in [0, a]$, and that all points in $s \in [a, s_{\max}]$ are in R, i.e. $f(s) > g(s)$ for all $s \in [a, s_{\max}]$ [12].

Arguing by contradiction, we assume that

$$0 < \int_a^{s_{\max}} v(s) \left(y(s) - f(s)\right) ds, \tag{4.34}$$

and that (4.33) is satisfied for *all* $y \in Z$, in accordance to the definition of the subdifferential from (4.29). We can then choose any y such that $y(s) = g(s)$ when $s \in [0, a]$, which reduces (4.33) to

$$0 \geq 0 + \int_a^{s_{\max}} v(s) \left(y(s) - f(s)\right) ds, \tag{4.35}$$

which directly contradicts the assumption in (4.34). The argument is identical if we reverse the roles, and instead assume that $\int_0^a v(s) \left(y(s) - f(s)\right) ds$ is positive. Hence, when dividing Ω_s into two integrals, we must require that each new integral can satisfy the inequality on its own.

[12] Incidentally, this is the scenario that occurs for a standard American put option.

In fact, this applies regardless of how we carry out the division of Ω_s, or how many intervals we divide it into - if the inequality is not satisfied over each local interval, then it can not be satisfied globally, as we can always find a y that turns the inequality to zero for all intervals except one.

Without loss of generality, we can therefore assume that R and Q are connected intervals of Ω_s such that all s for which $f(s) = g(s)$ are in Q and all s for which $f(s) > g(s)$ are in R and (still assuming $P = \emptyset$) $Q \cup R = \Omega_s$.

We now show that $v(s) = 0$ for all $s \in R$. Where $s \in R$ holds, i.e. $f > g$ holds, then $f \in Z$ and (4.29) gives us that the following must hold for all $y \in Z$

$$0 \geq \int_R v(s)\,(y(s) - f(s))\,ds. \tag{4.36}$$

One v that satisfies this is $v = 0$, for obvious reasons. Are there any others? We assume that there are, and that they then must satisfy

$$0 > \int_R v(s)\,(y(s) - f(s))\,ds, \tag{4.37}$$

as the equality is only satisfied for $v(s) = 0$, due to y being all functions in Z.

Assume without loss of generality that v is continuous over the relevant interval. Arguing by contradiction, assume that there exists a point $p \in R$ such that $v(p) > \epsilon$ where ϵ is a positive constant. Then, because of continuity, for a small interval around p we have $v(x) > \frac{\epsilon}{2}$, where $x \in R$. Next, we examine $y - f$. Because $f > g$, there are $y \in Z$ such that $y - f$ is negative, positive or zero for some $s \in R$. If we take $v(p)$, we can then choose y such that $y - f$ is positive in the small interval around p and zero everywhere else. But then, when we carry out the integration in (4.37), the result will be positive. This violates the condition in (4.37), and thus v cannot anywhere admit a value larger than some positive constant ϵ - in other words, v cannot take positive values and so we have shown $v(x) \leq 0$ for all $x \in R$.

Now, once more arguing by contradiction, we instead assume that there exists a point $p \in R$ such that $v(p) < \epsilon$ where ϵ is a *negative* constant. Then, because of continuity, for a small interval around p we have $v(x) < \frac{\epsilon}{2}$. We can choose y such that $y - f$ is negative in a small interval around p and zero everywhere else. Carrying out the integration in (4.37) would then give a positive value of the integral, contradicting (4.37) - which gives us that $v(x) \geq 0$ for all $x \in R$.

Having now demonstrated that $v(x) \leq 0$ and $v(x) \geq 0$ both hold for all $x \in R$, we then have that $v(x) = 0$ over each such interval of R.
And so, we have proven $\partial I_Z(f) = \{0\}$ if $s \in R$.

We now show that $v(s) \geq 0$ for all $s \in Q$. Where $s \in Q$ holds, i.e. $f = g$ holds, then $f \in Z$. Now, we must have the condition

$$0 \geq \int_Q v(y - f)ds, \quad \text{for all } y \in Z, \tag{4.38}$$

Assume, without loss of generality, that v is a continuous function. Arguing by contradiction, we assume further for some $p \in Q$ that $v(p) > \epsilon$ a positive constant ϵ - i.e. assume that v is positive somewhere. Because of continuity of v there are $q \in Q$ such that $v(q) > \frac{\epsilon}{2}$ in a small interval near p. We must examine all choices of $y \in Z$, but because $f = g$, $y(s) - f(s)$ must always be zero or positive. If $y(s) - f(s)$ was negative for any $s \in Q$, then y would be smaller than f somewhere, but since $f = g$, f is everywhere as small as can be while remaining in Z.

Hence, $y(s) - f(s)$ is always positive or zero, and we can then always find a y such that $y(s) - f(s)$ is zero outside the small interval around p, but a positive value inside the interval. Carrying out the integration in (4.38) would then however result in a positive value (zero contribution from outside the interval, positive contribution from within the interval), contradicting (4.38).

And so the assumption that v can be larger than some positive constant is false, and v must be smaller than all positive constants. Hence, $v(s) \leq 0,\ s \in Q$. We then have $\partial I_Z(f) = \{v | v(s) \leq 0\}$ if $f = g$. □

We have now completely characterized the subdifferential of the indicator function. We did all this because the second operator turns out to be precisely the subdifferential of this indicator function!

Reconstructing Ikonen-Toivanen

We restate the subdifferential of the indicator function from lemma 4.3.2.

$$\partial I_Z(f) = \begin{cases} \emptyset, & P \neq \emptyset, \\ \{v \in L^2(\Omega_s) | v(s) = 0,\ \forall s \in R \text{ and } v(s) \leq 0,\ \forall s \in Q\}, & P = \emptyset. \end{cases} \tag{4.39}$$

where P is the set of $s \in \Omega_s$ for which $f(s) < g(s)$, R is the set of $s \in \Omega_s$ for which $f(s) > g(s)$ and Q is the set of $s \in \Omega_s$ for which $f(s) = g(s)$.

Recalling the second step of Douglas-Rachford from (4.20), we insert the subdifferential from above as the second operator. By replacing the equality with an element inclusion to further illustrate that we have a multi-valued operator and specifying further that the f we seek is in fact going to be f_i, we arrive at (4.40)[13]

$$\frac{\boldsymbol{f}_i - \tilde{\boldsymbol{f}}_i}{\Delta t} + \partial I_Z(\boldsymbol{f}_i) - \partial I_Z(\boldsymbol{f}_{i+1}) \ni 0, \tag{4.40}$$

[13]In the appendix, A.3, we give a brief example of what this *differential inclusion* notation means.

where $\partial I_Z(\boldsymbol{f}_{i+1})$ will be a known vector of constants (single-valued!) from the previous time step. Note that we generalize the operator to take vector inputs by simply treating each component independently. For this reason, we will somewhat informally use f_i to mean an arbitrary element of the vector $\boldsymbol{f}_i$.

Further, $\tilde{\boldsymbol{f}}_i$ is also assumed to be known from the previous *fractional* time step, and the only unknowns are the elements of the vector $\boldsymbol{f}_i$, which are used as the input argument into ∂I_Z. What then, does (4.39) tell us about (4.40)?

If $f_i < g$, then (4.40) can never be satisfied as the empty set will "annihilate" the left hand side. So, to solve (4.40) we must demand that all the unknown f_i take values that do not result in ∂I_Z becoming the empty set.

Note that this requirement corresponds to requiring that $f_i \geq g$, which we recall is precisely one of the constraints of the second step in Ikonen-Toivanen!

The remaining solutions of the element inclusion in (4.40) depend on the remaining case of (4.39). We will make the suggestive substitution of setting $\partial I_Z(f_i) = -\lambda_i$. Implementing the substitution gives,

$$\begin{aligned} \frac{f_i - \tilde{f}_i}{\Delta t} + (-\lambda_i) - (-\lambda_{i+1}) \ni 0, \\ f_i \geq g, \end{aligned} \tag{4.41}$$

where λ_i depends on f_i according to (4.39). What are the possible values λ_i can take? From the second case in (4.39), we readily note that λ_i is restricted to taking values according to $-\lambda_i \leq 0$, which corresponds to $\lambda_i \geq 0$.
The second case in (4.31) further tells that if $f_i > g$, then $-\lambda_i = 0$ which is equivalent to $\lambda_i = 0$. We express this as $(f_i - g)\lambda_i = 0$, which when taken in conjunction with $\lambda_i \geq 0$ only imposes the desired restriction. We can now express (4.41) as

$$\begin{cases} \frac{f_i - \tilde{f}_i}{\Delta t} + \lambda_{i+1} - \lambda_i = 0, \\ f_i \geq g, \\ \lambda_i \geq 0, \\ (f_i - g)\lambda_i = 0. \end{cases}$$

Which we recognize as the second step of Ikonen-Toivanen, (4.14), reconstructed from Douglas-Rachford!

A short summary

If we apply the generalized Douglas-Rachford algorithm,

$$\begin{cases} \frac{\tilde{\boldsymbol{f}}_i - \boldsymbol{f}_{i+1}}{\Delta t} + A_1(\tilde{\boldsymbol{f}}_i, \boldsymbol{f}_{i+1}) + A_2(\boldsymbol{f}_{i+1}) = 0, \\ \frac{\boldsymbol{f}_i - \tilde{\boldsymbol{f}}_i}{\Delta t} + A_2(\boldsymbol{f}_i) - A_2(\boldsymbol{f}_{i+1}) = 0 \end{cases} \tag{4.42}$$

and choose $A_1(\boldsymbol{f}_i) = Q(\boldsymbol{f}_i)$, and $A_2(\boldsymbol{f}_i) = \partial I_Z(\boldsymbol{f}_i)$, the multivalued operator A_2 in conjunction with the second Douglas-Rachford step reconstructs exactly the equation and the constraints in the second step in the Ikonen-Toivanen method, if we make the substitution $\partial I_Z(\boldsymbol{f}_i) = -\lambda_i$.

Remark In appendix A.5 we demonstrate that by choosing A_1 and A_2 as above we actually reconstruct the full LCP.

4.3.3 With BDF2 discretization

With the BDF2 discretization, we simply modify (4.20) by replacing the first term in the left-hand side and the first operator. With $\boldsymbol{\mathcal{A}}$ modified in the same fashion as in PSOR (i.e. the diagonal having $\frac{1}{2}$ added to every element) and setting $\theta = 1$.

$$\boldsymbol{\mathcal{A}}_{\mathrm{BDF2}}\tilde{\boldsymbol{f}}_i = 2\boldsymbol{f}_{i+1} - \frac{1}{2}\boldsymbol{f}_{i+2} - \boldsymbol{b}_i + \Delta t\boldsymbol{\lambda}_{i+1}. \tag{4.43}$$

In the second step, only the first constraint is modified to account for the change in discretization,

$$\begin{cases} \boldsymbol{f}_i - \tilde{\boldsymbol{f}}_i - \frac{2}{3}\Delta t\left(\boldsymbol{\lambda}_i - \boldsymbol{\lambda}_{i+1}\right) = 0, \\ \boldsymbol{f}_i \geq \boldsymbol{g}, \\ \boldsymbol{\lambda}_i \geq 0, \\ (\boldsymbol{f}_i - \boldsymbol{g})^T(\boldsymbol{\lambda}_i) = 0, \end{cases} \tag{4.44}$$

As we can see, shifting a Crank-Nicolson implementation to a BDF2 implementation is fairly easy.

4.4 Direct solution with projection on the boundary - Brennan-Schwartz

Another way to address the LCP in (3.16), suggested by Brennan and Schwartz in [2] and examined further in [16], is by first solving the unconstrained problem directly, and then simply projecting the solutions of that onto the exercise constraint. Each time step would then involve two fractional time steps, one in which we solve the linear problem and one in which we project that solution onto the exercise condition (the free boundary).

Using previously introduced notation, with the discretization as before and using the described tridiagonal LU solver for the solution of the linear system, we get

$$\begin{aligned} &\boldsymbol{\mathcal{A}}\tilde{\boldsymbol{f}}_i - \boldsymbol{M}\boldsymbol{f}_{i+1} - \check{\boldsymbol{b}}_{i+1} + \boldsymbol{b}_i = 0, \\ &\boldsymbol{f}_i = \max\left(\tilde{\boldsymbol{f}}_i, \boldsymbol{g}\right). \end{aligned} \tag{4.45}$$

Where $\max(\cdot,\cdot)$ is understood to be performed componentwise.

Notice the similarity between what happens in (4.7) and in (4.45). The underlying principle between the two methods is the same - solve the system, and then project it onto the boundary. But in PSOR the system is solved using an iterative solver, and in the above the system is solved using a direct solver. The strength of this method then rests entirely on the accuracy of the direct solution compared to the iterative, and its weakness on the time it takes to carry out that calculation.

4.4.1 Brennan-Schwartz with BDF2

In perfect analogue to the previous two BDF2 displays, we then modify the coefficient matrix $\boldsymbol{\mathcal{A}}$ by adding $\frac{1}{2}$ to the diagonal elements, setting $\theta = 1$ and then modifying the right hand side by replacing $\boldsymbol{M}\boldsymbol{f}_{i+1} + \check{\boldsymbol{b}}_{i+1}$ with $2\boldsymbol{f}_{i+1} - \frac{1}{2}\boldsymbol{f}_{i+2}$.

$$\begin{aligned} \boldsymbol{\mathcal{A}}_{\text{BDF2}}\tilde{\boldsymbol{f}}_i &= 2\boldsymbol{f}_{i+1} - \frac{1}{2}\boldsymbol{f}_{i+2} - \boldsymbol{b}_i, \\ \boldsymbol{f}_i &= \max\left(\tilde{\boldsymbol{f}}_i, \boldsymbol{g}\right). \end{aligned} \tag{4.46}$$

Which, once again, is easy to implement if you already have a Crank-Nicolson implementation.

4.5 A connection to operator splitting

Interestingly, Brennan and Schwartz method can also be traced to operator splitting, as pointed out in [5]. With the same split-operators as in Ikonen-Toivanen, but using a simpler algorithm, we can express Brennan-Schwartz. This algorithm does not have the same general convergence as Douglas-Rachford, which makes it less desirable for general use.[14] For our examination, however, we will rely on empirical studies.

Using the notation we used to characterize the Douglas-Rachford algorithm, we can characterize this algorithm according to

$$\begin{cases} \frac{\tilde{\boldsymbol{f}}_i - \boldsymbol{f}_{i+1}}{\Delta t} + A_1(\tilde{\boldsymbol{f}}_i, \boldsymbol{f}_{i+1}) = 0, \\ \frac{\boldsymbol{f}_i - \tilde{\boldsymbol{f}}_i}{\Delta t} + A_2(\boldsymbol{f}_i) = 0. \end{cases} \tag{4.47}$$

Rewriting this so as to remove the intermediate step

$$\boldsymbol{f}_i = (I + \Delta t A_2)^{-1}(I + \Delta t A_1)^{-1}\boldsymbol{f}_{i+1} \tag{4.48}$$

[14] Lions has actually examined when it does converge in another article, [3], which the interested reader is welcome to consult.

Noting that the resolvent of A_2, i.e. $(I + \Delta t A_2)^{-1}$ is actually the projection operator[15], one can see that the above is exactly the Brennan and Schwartz algorithm - though we now arrived at it via operator splitting.

Remark If we further realize that SOR is really about approximating $(I+\Delta t A_1)^{-1}$ so as to not carry out the inversion, we have gained a measure of insight into how PSOR works as well.

4.6 Peaceman-Rachford

Another ADI scheme is the Peaceman-Rachford algorithm. Unlike Douglas-Rachford or any of the other methods previously described, it has second order in time accuracy for sufficiently smooth solutions. It is this theoretically higher upper bound on the accuracy that makes the method interesting to us. Our empirical examination will reveal whether the accuracy degenerates and compare the calculation time with the previously described methods.

Peaceman-Rachford was examined both in [5] and in [12, p. 52-60], and we will use the characterization from [12] here. Peaceman-Rachford is the scheme described by[16]

$$\begin{cases} \frac{\tilde{\boldsymbol{f}}_i - \boldsymbol{f}_{i+1}}{\frac{\Delta t}{2}} + A_1(\tilde{\boldsymbol{f}}_i, \boldsymbol{f}_{i+1}) + A_2(\boldsymbol{f}_{i+1}) = 0, \\ \frac{\boldsymbol{f}_i - 2\tilde{\boldsymbol{f}}_i + \boldsymbol{f}_{i+1}}{\frac{\Delta t}{2}} + A_2(\boldsymbol{f}_i) - A_2(\boldsymbol{f}_{i+1}) = 0 \end{cases}. \tag{4.49}$$

Using the two operators described for Douglas-Rachford, with the same reasoning as in "Reconstructing Ikonen-Toivanen", we construct the first fractional time step according to

$$\frac{\tilde{\boldsymbol{f}}_i - \boldsymbol{f}_{i+1}}{\frac{\Delta t}{2}} + \underline{\boldsymbol{Q}}(\tilde{\boldsymbol{f}}_i, \boldsymbol{f}_{i+1}) - \boldsymbol{\lambda}_{i+1} = 0, \tag{4.50}$$

which is Douglas-Rachford except for the alteration to Δt. The second step becomes

$$\begin{cases} \boldsymbol{f}_i - 2\tilde{\boldsymbol{f}}_i + \boldsymbol{f}_{i+1} - \frac{\Delta t}{2}(\boldsymbol{\lambda}_i - \boldsymbol{\lambda}_{i+1}) = 0, \\ \boldsymbol{f}_i \geq \boldsymbol{g}, \\ \boldsymbol{\lambda}_i \geq 0, \\ (\boldsymbol{f}_i - \boldsymbol{g})^T(\boldsymbol{\lambda}_i) = 0. \end{cases} \tag{4.51}$$

These two steps are solved in precisely the same manner as the corresponding steps for Douglas-Rachford. For an alternative formulation without an intermediate step, see appendix A.6.

[15] In [5] this is stated as a fact, and in appendix A.7, we demonstrate that it follows directly from the definition of the subdifferential.

[16] Note that Peaceman-Rachford features a different time step for the intermediate step than Douglas-Rachford. If one had a time dependant operator - we do not - the construction of Peaceman-Rachford and Douglas-Rachford would differ further, see [12, p. 52-60] for more details.

Chapter 5

Numerical Practicalities

In order to perform a useful comparison, these methods needs to be tested under similar conditions corresponding to the intended practical application. The purpose of the tests will be to compare accuracy as a function of computational time.

Based on the practical application, we can set an error tolerance. We will aim to maintain an error of at most order 10^{-3}, as very few currencies are such that it makes sense to be able to report a value in that currency with an accuracy below 10^{-2}. In order to calculate an error in the first place, we need a reference value. For our problem, this value needs to be calculated numerically, and we choose to use the binomial method with 12 000 time steps to generate reference values. This choice of reference accuracy is consistent with our desired error tolerance, and a desire to keep the time required to generate reference values low. For more details for the justification of this choice, see appendix A.8.

We also need an error measure, and we choose to use the L^∞-norm, which means we examine the maximum absolute error of the FD solution vectors. This means that the error assigned to each method for every combination of numerical parameters is a "worst-case" error, which makes it appropriate for applications in financial mathematics. In the appendix A.10 this is illustrated further. As the binomial method returns only a single value, we will repeat multiple calculations using the binomial method for different starting spots in the same interval as the FD methods operate. In other words, we select a number of time steps for the binomial method, we then ask for option prices in the intervall $[0, s_{\max}]$, and then we take the maximum absolute error of these answers to represent the binomial method for that number of time steps.

The next step is to choose numerical parameter values for the four methods. In the case of the binomial method, there is only one parameter. Based on some simple tests against the reference value, we look at steps in the interval 10 to 1000 in most cases.

For the FD methods we need N and M, and for PSOR we also need an iteration error tolerance ϵ. Selecting suitable values for these parameters is more complicated, as their impact on the numerical answer depends on the other parameters - they cannot be selected independently. We select these parameters for our final results based on empirical tests. While we need the parameter selection rules produced by those tests for practical implementation, explaining the tests in great detail distracts from the main purpose of thesis. Therefore, appendix A.9 contains the tests and surrounding reasoning. We settle for noting that N, M and ϵ were varied, and by examining the error for various combinations of the three, we found that the following parameter selection rules were suitable.

For all 3 FD methods, we vary M from 100 to 500.

For Brennan-Schwartz, we select the number of time steps N to be approximately 90% of the number of space steps.

For Ikonen-Toivanen (Douglas-Rachford), we select the number of time steps N to be approximately 15% of the number of space steps.

For PSOR, setting N to be approximately 30% of M is appropriate, and the interesting range of ϵ was determined to be between 10^{-4} and 10^{-7}.

For Peaceman-Rachford, our calibration reveals that the number of time steps must increase faster than the number of time steps to ensure convergence. As this is not the case for Douglas-Rachford - which has a similar computational cost *per time step* - we can already note that if the accuracy of Peaceman-Rachford does degenerate, it will be strictly dominated by Douglas-Rachford, as Douglas-Rachford will require less time steps for the same accuracy.

Using the above, we can now proceed with the examination of accuracy as a function of computational time. For varying option parameters employing the methods described, we examine the error and the calculation time. Practically, we carry out the numerical calculations in standalone programs written in C, and the results are then processed in MATLAB to generate visual representations. The hardware used for the experiments is specified in appendix A.1.

Chapter 6

Results

6.1 Error as a function of calculation time

The plot on the next page is made as follows: the interval $[0, 400 = s_{\max}]$ was divided into 400 points. We examine the error at each of these points for varying parameters of M, N and ϵ. Whenever needed, a cubic spline interpolation is carried out to reach the price at points in between the elements in the FDM answer vectors. For each combination of numerical parameters (where M and N follow the selection rules set out earlier) we take the maximum absolute error among the 400 points. This error will then include both the numerical error, and the interpolation error. These are then plotted against an average of the calculation time across the 400 points in a numerical parameter set.

The binomial line in the plot is generated similarly, though here we only have have one numerical parameter: the number of time steps. The time for each calculation is an average over several calculation.

Initially, we compare only the methods that are first order accurate in time, i.e. every method except Peaceman-Rachford, which will be treated last.

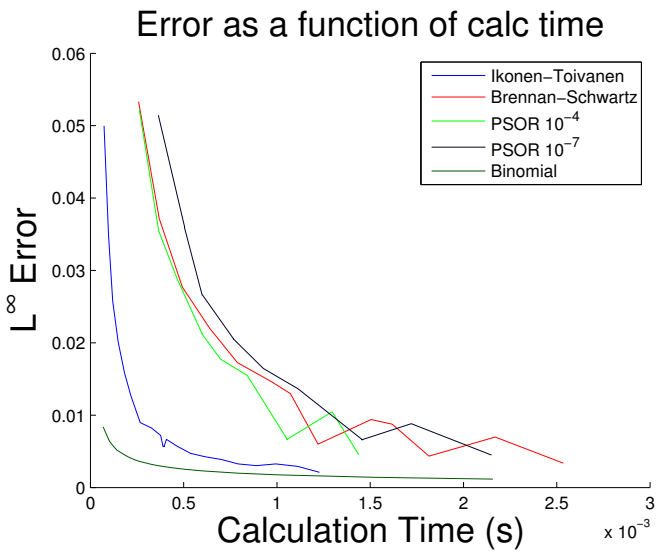

Figure 6.1. Option parameters in this plot: $r = 0.02$, $\sigma = 0.15$, $T = 1$, $K = 120$. Note that the this plot contains interpolation where necessary. The bottom line is the binomial method.

There are several important things to observe in Figure 6.1: Ikonen-Toivanen outperforms the other FD methods significantly in the time scale highlighted above, but is outperformed by the binomial method. There is a visible jaggedness to three of the FD methods, relating to the combination of grid shape and problem parameters. We know, from the preceding calibration and theory, that the PSOR method with an error tolerance of 10^{-4} will experience a worsening of convergence properties for finer grids than the ones tested here, but on this time scale, it is capable of performance better or comparable to Brennan-Schwartz.[1] We cannot lower the error tolerance of PSOR without seeing some additional degradation of convergence, and increasing the error tolerance will also increase the calculation time.

Next, we repeat the same set up, except instead of testing 400 points and interpolating, we test only the numerical error at the points the FD methods produce, and the binomial method is represented at each time step by the maximum error of 850 examined points.

[1]The machine specifications used to generate all of the plots in the thesis are available in appendix A.1.

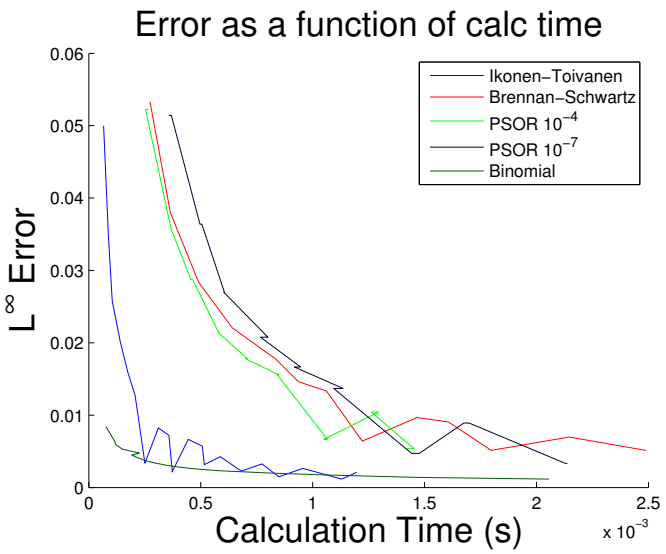

Figure 6.2. Option parameters in this plot: $r = 0.02$, $\sigma = 0.15$, $T = 1$, $K = 120$. Error is derived from one FDM answer vector (an average of several calculations, but of the same FDM answer vector, unlike in the previouse plot). The bottom line is the binomial method.

As we can see in Figure 6.2, the most important qualities are retained: The relative ordering from "best to worst" at each time remains approximately the same as in Figure 6.1. The Ikonen-Toivanen method appears more flattering in this light - due to error-term related factors related to the specific combination of option and numerical parameters used here - and there are some artefacts on all methods.

Based on these two figures, we can conclude that the Douglas-Rachford scheme used by Ikonen and Toivanen strictly outperforms Brennan and Schwartz much simpler alternating direction algorithm. PSOR is harder to dismiss out of hand, as PSOR has ease-of-implementation potential for more complicated models - anything that breaks the tridiagonal algorithm will require workarounds in both Brennan-Schwartz and in Ikonen-Toivanen (workarounds that will use additional calculation time) where as modifying PSOR to deal with it is a matter of increasing the number of terms in some sums.

6.1.1 Rannacher Start up and other time stepping

We implement Rannacher time stepping on Ikonen-Toivanen, and once more plot the error as a function of calculation time, using evenly spaced test values that we interpolate to when needed. By the suggestion in [14] where they specifically seek to locate the optimum Rannacher implementation for the Black-Scholes framework, we replace the first Crank-Nicolson time step, with four steps of implicit time stepping.

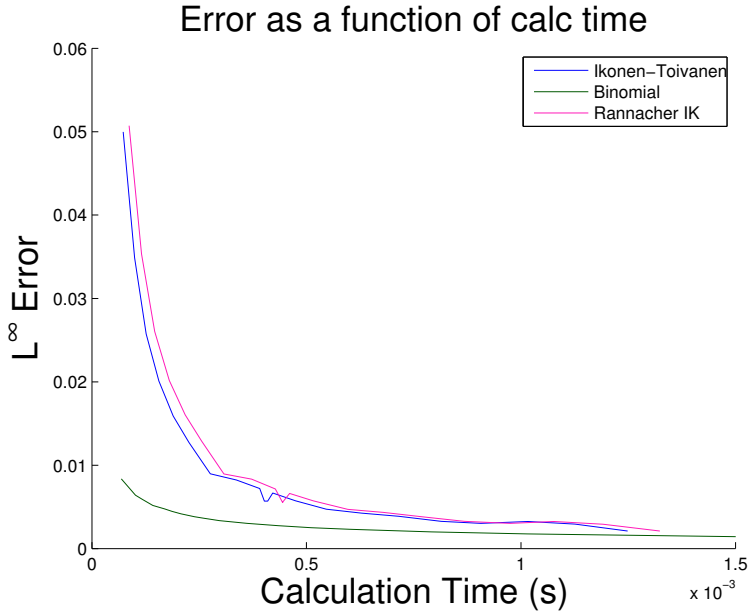

Figure 6.3. Option parameters in this plot: $r = 0.02$, $\sigma = 0.15$, $T = 1$, $K = 120$. Note that the this plot contains interpolation where necessary. The bottom line is the binomial method.

We see in Figure 6.3 that the Rannacher alteration either performs worse, or so close as to make the difference negligible in either direction. The early parts of the graph is generated using comparatively few time steps, and thus the percentage increase in the total number of time steps is larger and so then is its relative impact on the calculation time - an increase in accuracy must also offset the fact that more time steps are taken. The actual implementation of the tridiagonal solver also changes in a way that increases calculation time: For a constant time step, the decomposition and forward substitution can be carried out once, and then holds for all time steps. For a changing time step, this needs to be recalculated for every change, so the Rannacher implementation we used requires that these calculations be repeated twice.

The Rannacher implementation does however have a use separate from option *pricing*: The calculation of the financial "greeks", nicknames for a set of derivatives of the option price used to gauge how changes in various parameters - perhaps most commonly the volatility and underlying spot price - change the price of the option. The calculation of these reinforces the oscillations that can present themselves in a scheme based on a Crank-Nicolson discretization, and the Rannacher start-up alleviates these problems. We do not explore this further, the interested reader can consult [14] where it is done using European options. (Though of course, one might then also attempt another second-order accurate scheme such as BDF2.)

For completeness sake, we include Rannacher time marching on the Brennan-Schwartz algorithm as well.

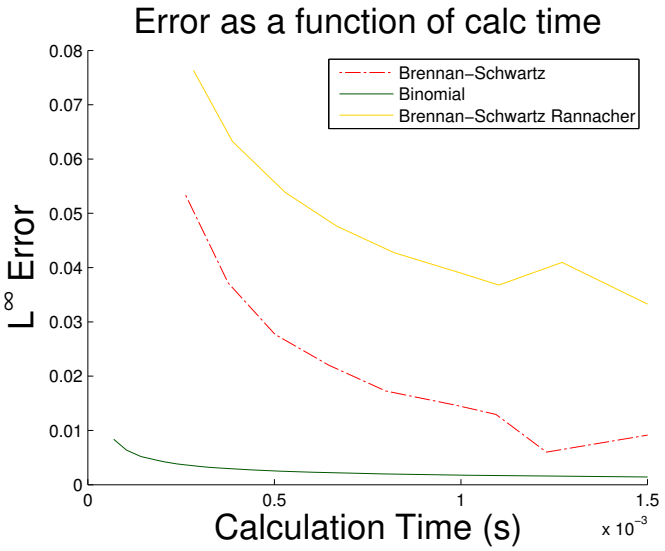

Figure 6.4. Option parameters in this plot: $r = 0.02$, $\sigma = 0.15$, $T = 1$, $K = 120$. Note that the this plot contains interpolation where necessary. The bottom line is the binomial method.

As we can see in Figure 6.4 , this led to a significant drop in accuracy rather than an improvement. Where as the difference between a normal start-up and a Rannacher start-up for the Douglas-Rachford algorithm was small (and appeared to disappear as the grid size increased), the above is anything but small, nor is it a shift in calculation time. Clearly, the Rannacher start-up degrades the accuracy of the scheme and algorithm for our problem.

Along the same vein as Rannacher time stepping, and with the same practical concerns for implementation, we attempt a transformation of the discretization in the temporal direction. As previously mentioned, one of the side effects of Crank Nicolson is an oscillatory behaviour that attenuates for the number of time steps: One theoretical approach that suggests itself is then to concentrate the time steps near expiry, which would dampen the oscillations, and decrease the number of time steps necessary.

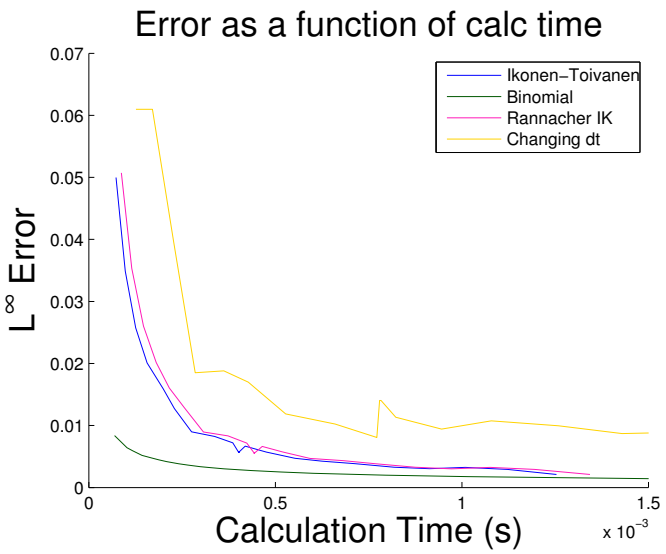

Figure 6.5. Option parameters in this plot: $r = 0.02$, $\sigma = 0.15$, $T = 1$, $K = 120$. Note that the this plot contains interpolation where necessary. The bottom line is the binomial method.

In Figure 6.5 however, the decomposition and forward substitution must be performed in *every* time step, with the predictable impact on the calculation time we see above.

So far, our investigation suggests that that unmodified versions of the operator splitting algorithms yield a better performance than attempts to refine them, though the Douglas-Rachford scheme outperforms Brennan-Schwartz.

6.1.2 Backward differentation formula of order 2

A similar plot, but instead of Crank-Nicolson with a Rannacher startup, we employ the BDF2 discretization scheme, with the first time step being fully implicit, and compare it to the Ikonen-Toivanen method with Crank-Nicolson discretization. Then we do the same comparison, but using the Brennan-Schwartz algorithm

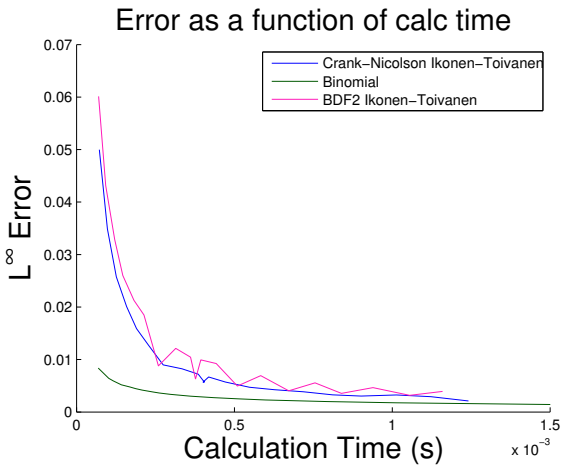

Figure 6.6. Option parameters in this plot: $r = 0.02$, $\sigma = 0.15$, $T = 1$, $K = 120$. The bottom line is the binomial method, interpolation has been used for both FDM calculations, which is the same FD method - Ikonen-Toivanen - but with different underlying discretizations: Crank-Nicolson or BDF2.

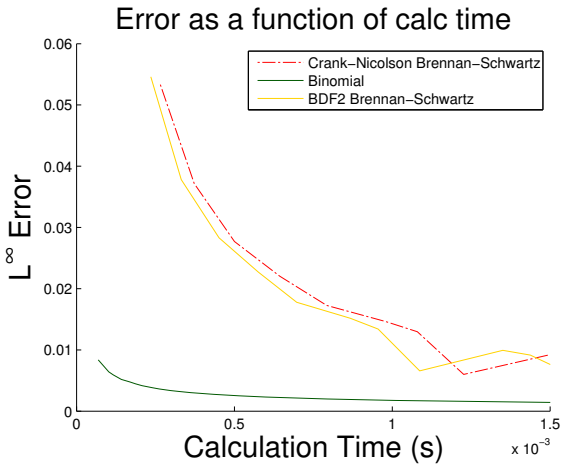

Figure 6.7. Option parameters in this plot: $r = 0.02$, $\sigma = 0.15$, $T = 1$, $K = 120$. Same as previous plot, except now with Brennan-Schwartz instead.

Interestingly, based on Figure 6.6 and Figure 6.7, we note that for Ikonen-Toivanen the BDF2 scheme performs worse, where as for Brennan-Schwartz it actually performs better, though far from enough to compete with Ikonen-Toivanen - for either discretization. We do not explore this in greater detail, but suggest that the answer is to be found in how the respective operator splitting algorithms interacts with the error terms of Crank-Nicolson and BDF2 respectively. It is this same

interaction that leads to the empirical result that lets us use a numerical parameter selection rule for Ikonen-Toivanen with such a low number of time steps.

We also note that the jagged convergence seen in Figure 6.2 is *not* solely due to oscillations introduced by Crank-Nicolson - if it were, BDF2 would be much smoother as it is immune to the type of oscillations that Crank-Nicolson can pick up. Instead, it appears to be a property of the Douglas-Rachford algorithm itself.

Next, we compare the BDF2 and Crank-Nicolson discretizations and their use with the PSOR algorithm for the same error tolerances as displayed previously.

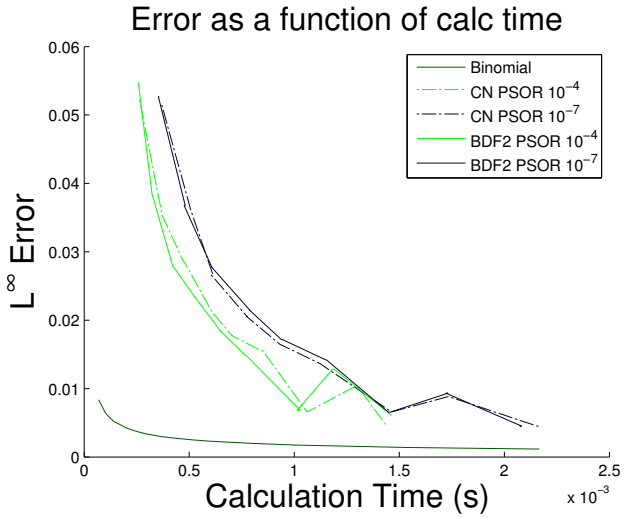

Figure 6.8. Option parameters in this plot: $r = 0.02$, $\sigma = 0.15$, $T = 1$, $K = 120$. Same as previous plot, except now with PSOR instead.

Though neither PSOR can compete with the binomial method in the interesting time frame according to Figure 6.8 , we see that the BDF2 scheme actually outperforms Crank-Nicolson in PSOR with the lower error tolerance - though we already know that we will not be able to get much better accuracy without increasing the error tolerance anyway, and for the higher error tolerance the selection of Crank-Nicolson or BDF2 appears to matter a lot less.

6.1.3 Shifting the truncation of the boundary

Next, we attempt to reduce $s_{\max}$, thus refining the grid. We test this on the Crank-Nicolson discretization.

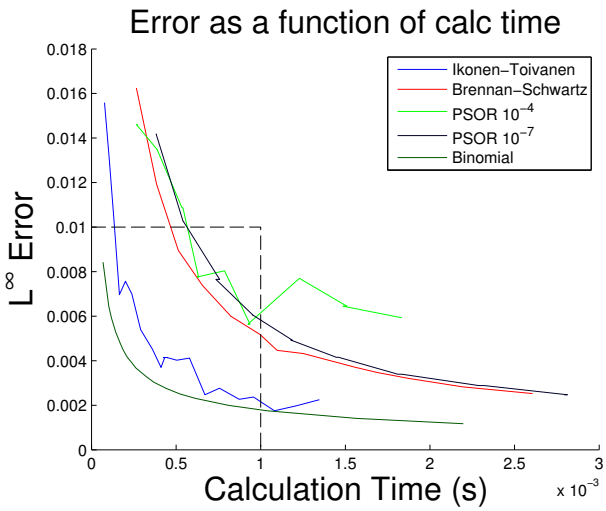

Figure 6.9. Option parameters in this plot: $r = 0.02$, $\sigma = 0.15$, $T = 1$, $K = 120$. $s_{\max}$ now set to 200, meaning that the same number of steps in the spatial discretization actually leads to a smaller step *size*, giving possibly better convergence though introducing a possibly ruinous truncation error.

Compare Figure 6.9 to Figure 6.1 or any previous plot, focusing on the position of the dashed rectangle, and which methods pass through it. It is clear that when we reduce $s_{\max}$, the refinement of the grid increases accuracy so much that the increase in truncation error is completely negligible next to the decrease in overall error. We also note that PSOR with the lower of the two error tolerances displayed visually, has had its convergence degraded by the grid refinement, and Douglas-Rachford becomes more erratic in its convergence, as seen by the increased jaggedness of the plot.

6.1.4 Peaceman-Rachford

Lastly, we perform the test of the Peaceman-Rachford algorithm. We use the parameters from the previous section, notably $s_{\max} = 200$. For clarity, we compare Peaceman-Rachford with the binomial method, we use dashed lines to mark the region of interest, the same region as in Figure 6.9.

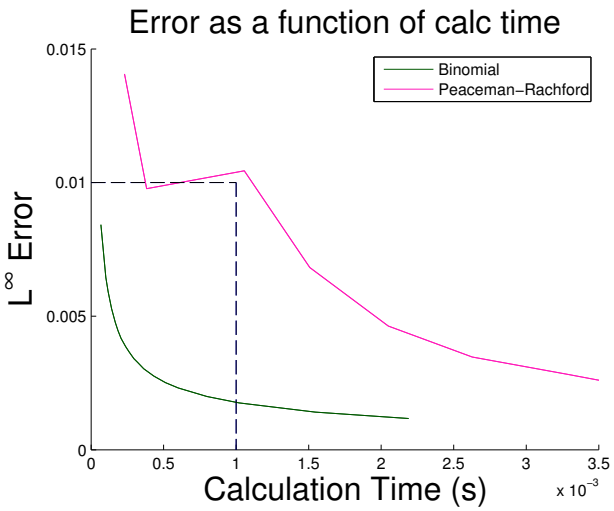

Figure 6.10. Option parameters in this plot: $r = 0.02$, $\sigma = 0.15$, $T = 1$, $K = 120$. $s_{\max}200$. The dashed lines denote the interesting region. Comparing this plot with figure 6.9 show that Peaceman-Rachford does not outperform any of the previous methods.

We note from Figure 6.10 first that the accuracy of Peaceman-Rachford has, in fact, degenerated, and second that all other methods outperforms it. As was pointed out in Chapter 5, the accuracy degrading combined with the stability properties of Peaceman-Rachford ensured that Douglas-Rachford would be the superior method of the two.

6.2 Conclusions

The first conclusion we can draw from our results is that, for the American put option, the binomial model is *not* outperformed by any of the FD methods tested - where performance is defined as the accuracy or calculation time trade off near the 1-2 millisecond range (on the equipment used, defined in A.1). This holds whether we use a Crank-Nicolson discretization or use a fully implicit BDF2 discretization, though the latter was demonstrably faster for one of the tested FD methods, the algorithm referred to as the Brennan and Schwartz algorithm in the context of pricing American option.

Out of the FD methods tested however, the algorithm known as Douglas-Rachford, suggested by Ikonen and Toivanen for use in the pricing of American options, is clearly superior on this particular model problem.

For the Douglas-Rachford algorithm, we tested a Rannacher start-up for the Crank-Nicolson discretization that did not manage to improve accuracy enough to warrant

the increase in calculation time it created, and a fixed transformation of the time steps, intending to counteract the oscillations of Crank-Nicolson, did not constitute a performance improvement either. We also note that the accuracy of the Brennan-Schwartz algorithm degrades further for an implicit start-up.

PSOR with a low enough error tolerance to make it faster than Brennan-Schwartz risks having its convergence degraded for grids fine enough to reach the interesting region of accuracy and calculation time. PSOR with higher tolerances are slower but maintain their convergence for longer, and PSOR in general cannot outperform the Douglas-Rachford operator splitting algorithm, for the model problem used here. It is however worth keeping in mind that PSOR deals with more complex models of the underlying in a fashion that suggests it may be not necessarily be worse than the other methods for those problems, and that it is certainly easier to implement the models using PSOR.

In addition, regarding the Peaceman-Rachford algorithm, we conclude that the accuracy degrades to first order. We also note that for our particular operator split, care must be taken when selecting the numerical parameters for the Peaceman-Rachford algorithm to maintain stability. These two facts render the Peaceman-Rachford algorithm worse than the Douglas-Rachford algorithm for our purposes.

6.3 Things of note

6.3.1 Improvements on the implementation

The calculation times are highly dependent on the actual coding implementation, and although care has been taken and professionals consulted to eliminate obvious inefficiencies, it is hard to rule out the idea that a given method could not be improved further by cleverly written code. For this reason, it is useful to briefly mentioned the most probably areas where such efforts should be focused[2].

For the operator splitting method, the tridiagonal solver represents the majority of the calculation effort (as expected), and work saved there - such as carrying out the decomposition and forward substitution independently of the time stepping, as used above - has a significant impact on performance. It may, for example, be possible to express the coefficient matrices as a sum of an identity matrix and a slightly different tridiagonal matrix, which is multiplied by Δt whenever needed.

For PSOR it is harder to identify such specific areas. The overrelaxation parameter (ω), could be more thoroughly investigated - [22] suggests that setting ω to 1 is the safest choice for financial application, where as we opted for the dynamic selection

[2]There are other areas, such as more detailed memory management, or even hardware changes, but the assumption is that any such inefficiencies in the current test setup has a similar impact across all methods tested.

suggested in [25] as it empirically proved itself to be better than a fixed parameter value. Both sources agree that theoretical calculation of the parameter is more time consuming than either a fixed choice or a dynamic one.
One could, however, examine whether there are better ways of making the dynamic changes, compared to the simple equal-step-in-either-direction we employed.

6.3.2 Preconditioner for PSOR

We have not opted to examine the results of preconditioning on PSOR, and previous studies (for example, [22, p. 97-99]) suggest that the selection of the proper preconditioner will be able to increase accuracy.

6.3.3 Multigrid methods

For problems with higher dimension (stochastic volatility, multiple underlyings) one could instead consider multigrid methods. Briefly, they are iterative methods similar to the Jacobi method used to explain PSOR. We do not examine these methods here as the increase in computational complexity will not pay for itself for our one-dimensional program[24, p. 871-888]

6.3.4 Theoretical error analysis

Several of the results indicate that it could be very interesting to study to the convergence rates in a theoretical fashion. While they are not useful as a replacement - degradation of accuracy occurs in our practical implementations anyway - they would still be an interesting complement, by giving us more predictive power over when the degradation occurs.

6.3.5 The selection of $s_{\max}$ - revisited

We opted to choose $s_{\max}$ at fixed values, and set the other numerical parameters against that backdrop. The advantage to do doing this is that M then completely controls to Δs, which simplifies analysis of changes in the errors when you change the size of the grid. However, once we've identified the grid sizes of interest - specifically, the Δs - we can reopen the case for another $s_{\max}$. We once again mention the suggestion given in Hull [20, p. 456], that the choice of $s_{\max}$ be made contingent on ensuring that one of the spatial discretization points actually ends up at the spot that we are interested in, making it possible to avoid interpolation[3].

For practical implementations where we are only interested in one price at one specific point (much like the query we send into the binomial model), this approach

[3]Up until we implement discrete dividends at any rate, in which case we would most likely have to interpolate inside the calculation anyway, to ensure that the price just before and after the dividend both land on discretized points.

is likely to be superior to interpolation - especially for the Douglas-Rachford algorithm, where the convergence is somewhat erratic anyway.

Appendix A

Additional comments and theory

A.1 Machine specifications

Testing was done on a machine running Windows 7 Enterprise, SP1, 64-bit OS, with 6 GB of installed RAM and with a Intel(R) Xeon(R) CPU W3565 @ 3.20GHz quad-core processor, although no code was adapted to specifically make use of additional cores.

A.2 Other models for the underlying

Briefly, we will mention some other models that also seek to model the evolution of stock prices, building on the framework of Black-Scholes. We mention these as part of the strength of the finite difference methods explored is that it is relatively easy to alter the model used, which is not true for the binomial model, though one could pursue another fully explicit method.

In Black and Scholes original paper the volatility was constant. Constant volatility does not match reality, and one extension of the Black-Scholes model with constant volatility is a model with time dependent volatility. In practical applications, the function that describes the change of this volatility is obtained by calibrating against existing options on the market to retrieve an implied, time dependent, volatility. Along the same idea, one can model the volatility as a stochastic process, instead of a deterministic function of time. One of the more famous such is referred to as the Heston Model after Steven L. Heston, who in 1993 presented a closed form solution to the pricing of standard options modelled under stochastic volatility. [8].

Another example is Kou's jump diffusion model, presented in [11], an additional stochastic term is added to create "jump diffusion" - sudden larger jumps, to more accurately model price shocks, such as the ones directly after a dividend payout. Although not the first jump diffusion model to be suggested, it allows for closed-form expressions of the more common options, making it easier to implement in

practice.

A.3 Differential inclusion

In (4.40) we use the $\ni$ sign in an equation to signify element inclusion. For clarity, we demonstrate what we mean by it. The function $f(x)$ is single-valued function and scalar and A is a set (say, the set given by the multivalued function $F(x)$ at the point x), and B is another set (say, $\{c\}$, where c is a constant). We postulate an example according to

$$f(x) + A + B \ni 0, \tag{A.1}$$

which we then take that to mean that 0 is an element in the following set

$$C = \{f(x) + a + b | a \in A, b \in B\}. \tag{A.2}$$

A.4 LU-decomposition and the TDMA

LU decomposition means we decompose a matrix into two matrices that when multiplied with each other returns the original matrix, with the additional property that one of the decomposed matrices is lower triangular and the other is upper triangular (the former has only zeros above the diagonal, the latter below the diagonal). The lower triangular matrix is traditionally denoted L and the upper triangular matrix is traditionally denoted U. The decomposition is usually done by assigning 1's to the diagonal of one of these matrices, say, L and then simply solving for the remaining matrix elements.

The purpose of the decomposition is to utilize that we can express a matrix equation, for example $Ax = b$, as $LUx = b$ or $L(Ux) = b$. If we then first solve $Lq = b$, we will get $q = Ux$ which we also solve. Because of the nature of L and U we can employ first forward substitution to get the intermediate q and then finally back substitution to get the sought values x.

In the case of tridiagonal systems, performing the necessary decomposition into L and U is easily implemented algorithmically. Recall the structure of the matrix $\boldsymbol{\mathcal{A}}$, and let us deconstruct it into an upper and lower triangular matrix. Assign 1's to the diagonal of L, and let l_{k_1} denote the elements in the subdiagonal of L, where $k_1 = 2, 3, 4, \ldots, M-1$. Further, let u_{k_2} denote the diagonal elements of U, where $k_2 = 1, 2, 3 \ldots, M-2$. The superdiagonal of $\boldsymbol{\mathcal{A}}$, the c_j coefficients, will also become the superdiagonal of U (where $j = 1, 2, 3, \ldots, M-1$) [1].

[1] For practical reasons, the non-zero elements of the tridiagonal matrix will most likely be available in vector form, allowing one to execute this algorithm without even concerning oneself with the "actual" matrix. This, obviously, reduces the amount of storage needed to contain the relevant information, which turn corresponds to significant savings in computational time.

$$\boldsymbol{\mathcal{A}} = \begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & 0 \\ 0 & a_3 & b_3 & c_3 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a_{M-2} & b_{M-2} & c_{M-2} \\ 0 & \dots & 0 & 0 & a_{M-1} & b_{M-1} \end{pmatrix}. \tag{A.3}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ l_2 & 1 & 0 & 0 & \dots & 0 \\ 0 & l_3 & 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & l_{M-2} & 1 & 0 \\ 0 & \dots & 0 & 0 & l_{M-1} & 1 \end{pmatrix}, U = \begin{pmatrix} u_1 & c_1 & 0 & 0 & \dots & 0 \\ 0 & u_2 & c_2 & 0 & \dots & 0 \\ 0 & 0 & u_3 & c_3 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 0 & u_{M-2} & c_{M-2} \\ 0 & \dots & 0 & 0 & 0 & u_{M-1} \end{pmatrix}. \tag{A.4}$$

Note that $k_1 - 1$ will always lead to a valid value of j, and that because of the size of the matrices, j can be used to enumerate the main diagonal of either L or U. Similarly, $k_2 \in j$ and $k_2 - 1 = k_1$, $\forall k_2$. Then the algorithm becomes,

$$\begin{cases} u_{M-1} = b_{M-1}, & k_2 = M-1, \\ l_{k_1} = \frac{a_j}{u_{k_1-1}}, & \\ u_{k_2} = b_{k_2} - \frac{l_{k_2-1}c_{k_2-1}}{u_{k_2-1}}, & \forall k_2 \neq M-1, \end{cases} \tag{A.5}$$

(We see that u_{k_2} characterizes all of the elements in the decomposition, as long as we remember where the 1's are.) We now have all elements belonging to the LU decomposition of a tridiagonal matrix. We denote the right hand side of the equation by Z_i, and introduce the intermediate variables $q_{i,j}$. We then recall that $\boldsymbol{\mathcal{A}}\tilde{f}_i = L(U\tilde{f}_i) = Z_i$, and we solve first $Lq = Z$ and then $U\tilde{f}_i = q$.
Note that the actual LU decomposition can be carried out once, as it remains the same for all time steps, *if we do not change* Δt. As $Z_{i,j}$ is *not* the same for each time step, certain variables need to be indexed with i to denote the changing in time. Variables with only one subscript are variables that are constant in time.[2] The actual forward and back substitution takes the following form:

$$\begin{cases} q_{i,1} = Z_{i,1}, & \\ q_{i,j} = Z_{i,j} - l_{j-1}q_{i,j-1}, & \forall j \neq 1 \\ \tilde{f}_{i,M-1} = \frac{q_{i,M-1}}{u_{M-1}}, & \\ \tilde{f}_{i,j} = \frac{q_{i,j} - c_j\tilde{f}_{i,j+1}}{u_j}, & \forall j \neq M-1. \end{cases} \tag{A.6}$$

[2]Though this is only completely true for our chosen problem and discretization, without applying Rannacher time stepping.

And we have solved the system using LU decomposition, with the answer contained in $\tilde{f}_i$. The algorithm can be used for any tridiagonal linear system, if we make the appropriate replacements of $\boldsymbol{\mathcal{A}}$, $\tilde{f}_i$ and $\boldsymbol{Z}_i$. We will reference the algorithm as TDMA, the **T**ri**D**iagonal **M**atrix **A**lgorithm.

A.5 Demonstrating that the Operator Split actually approximates the problem

For the sake of completeness, it might be interesting to briefly note that the choice of operators A_1 and A_2 as in 4.42 actually lead to $A_1 + A_2$ reconstructing the entire problem. To illustrate this we recall and restate (3.22), as below.

$$\begin{cases} \frac{\boldsymbol{f}_{i+1}-\boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}) \geq 0, \\ \boldsymbol{f}_i \geq \boldsymbol{g}, \\ (\boldsymbol{f}_i - \boldsymbol{g})^T \left(\frac{\boldsymbol{f}_{i+1}-\boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}) \right) = 0, \end{cases} \tag{A.7}$$

With that in mind, we make the following claim

Lemma A.5.1 *Recall the alternative formulation of the discrete LCP, found in* (3.22)*,*

$$\begin{cases} \frac{\boldsymbol{f}_{i+1}-\boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}, i) \geq 0, \\ \boldsymbol{f}_i \geq \boldsymbol{g}, \\ (\boldsymbol{f}_i - \boldsymbol{g})^T (\frac{\boldsymbol{f}_{i+1}-\boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}, i)) = 0. \end{cases} \tag{A.8}$$

The formulation seen in (A.8) *is equivalent to the set-valued evolution equation* (A.9) *(as seen in [5]), when we choose* $A_1 = \underline{\boldsymbol{Q}}$ *and* $A_2 = \partial I_Z$*, leading to* (A.10)

$$\frac{\boldsymbol{f}_{i+1} - \boldsymbol{f}_i}{\Delta t} + (A_1 + A_2)(\boldsymbol{f_i}) \ni 0, \tag{A.9}$$

$$\frac{\boldsymbol{f}_{i+1} - \boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}) + \partial I_Z(\boldsymbol{f}_i) \ni 0. \tag{A.10}$$

Proof We have already characterized the subdifferential of the indicator function, so we now repeat the procedure for the above set-valued equation rather than the set-valued second step of the Douglas-Rachford method.
We can then immediately conclude that we must once more demand that $f_i(s) \geq g(s)$, as otherwise the empty set returned from the subdifferential if $f_i(s) < g(s)$ will lead to an unsolvable equation, in perfect analogue to what we did for Douglas-Rachford. (Once again, the inequalities and equality must hold for all $s \in \Omega_s$).

The other two cases, when $f_i(s) = g(s)$ and $f_i(s) > g(s)$ can be expressed according to the below. Where we once again use f_i to signify an element of $\boldsymbol{f}_i$ (dropping (s) for brevity), with the understanding that the subdifferential operates on the vector

element-by-element, and thus gives rise to a system of equations. Note that this system is not uncoupled, as $\underline{\boldsymbol{Q}}$ does *not* operate on separately on each element.

$$\begin{aligned} f_i \in Z,\ f_i > g \Rightarrow \quad & \frac{\boldsymbol{f}_{i+1} - \boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}) = 0, \\ f_i \in Z,\ f_i = g \Rightarrow \quad & \frac{\boldsymbol{f}_{i+1} - \boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}) + [0, -\infty) \ni 0 \Rightarrow \\ \Rightarrow \quad & \frac{\boldsymbol{f}_{i+1} - \boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}) \geq 0 \end{aligned} \tag{A.11}$$

In both cases, the constraint $f_i(s) \geq g(s)$ is in effect. Rewriting this in a more compact form and reorganizing (dropping (s) and once more using vectors, with the understanding that each equality holds componentwise), we get

$$\begin{cases} \frac{\boldsymbol{f}_{i+1} - \boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}) \geq 0, \\ \boldsymbol{f}_i \geq \boldsymbol{g}, \\ (\boldsymbol{f}_i - \boldsymbol{g}_i)^T \left(\frac{\boldsymbol{f}_{i+1} - \boldsymbol{f}_i}{\Delta t} + \underline{\boldsymbol{Q}}(\boldsymbol{f}_i, \boldsymbol{f}_{i+1}). \right) \end{cases} \tag{A.12}$$

we have arrived back at the original discrete LCP, as seen in (3.22) and restated in (A.7). □

A.6 Yet another way to write Douglas-Rachford

A less suggestive but more readily analyzed way to write Douglas-Rachford, used by [5] is presented here. As before, A_1 and A_2 are the split operators and I is the identity operator,

$$\boldsymbol{f}_i = (I + \Delta t A_2)^{-1} \left((I + \Delta t A_1)^{-1} (I - \Delta t A_2) + \Delta t A_2 \right) \boldsymbol{f}_{i+1}. \tag{A.13}$$

Brennan and Schwartz algorithm is instead the algorithm

$$\boldsymbol{f}_i = (I + \Delta t A_2)^{-1} (I + \Delta t A_1)^{-1} \boldsymbol{f}_{i+1}. \tag{A.14}$$

Peaceman-Rachford, also as in [5], takes the following form

$$\boldsymbol{f}_i = (I + \frac{\Delta t}{2} A_2)^{-1} (I - \frac{\Delta t}{2} A_1)(I + \frac{\Delta t}{2} A_1)^{-1} (I - \frac{\Delta t}{2} A_2)^{-1} \boldsymbol{f}_{i+1}. \tag{A.15}$$

Note that formulation is equivalent to the one given in section 4.6, though this is not readily apparent.

All of these of these are closely related to what is known as alternating direction algorithms, and the reason for the name is fairly apparent when viewed in this form: An implicit step with one operator is followed by an explicit step with the other operator. Though in Douglas-Rachford, a more complicated combination of the two operators in the one of the steps (though we do alternate between implicit and explicit), and in Brennan-Schwartz both steps are implicit.

A.7 Resolvents and Projections

Define $\partial I_Z(f)$ as in the earlier parts of the thesis, and say that all s for which $f(s) < g(s)$ are in P, all s for which $f(s) = g(s)$ are in Q and all s for which $f(s) > g(s)$ are in R. Then

$$\partial I_Z(f) = \begin{cases} \emptyset, & P \neq \emptyset, \\ \{v \in L^2(\Omega_s) | v(s) = 0,\ \forall s \in R \text{ and } v(s) \leq 0,\ \forall s \in Q\}, & P = \emptyset. \end{cases} \tag{A.16}$$

and lets examine the operator $(I + \Delta t \partial I_Z)$ operating on f where I is the identity operator. Introduce $w = f + \Delta t v$, $w \in L^2(\Omega_s)$

$$(I + \Delta t \partial I_Z)(f) = \begin{cases} \emptyset, & P \neq \emptyset, \\ \{w \in L^2(\Omega_s) | w(s) = f(s),\ \forall s \in R \text{ and } w(s) \leq f(s),\ \forall s \in Q\}, & P = \emptyset. \end{cases} \tag{A.17}$$

So $(I + \Delta t \partial I_Z)$ takes $f(s)$ to a set of $f(s)$ itself when $f(s) > g(s)$, and when $f(s) = g(s)$ the function f is mapped to the set of all functions in $L^2(\Omega_s)$ that are smaller than $f(s)$ at that s. Consider the intuitive interpretation of the *inverse* of this: All functions $f(s)$ that are smaller than $g(s)$ are mapped to $g(s)$ and all functions $f(s)$ that are larger than $g(s)$ are mapped to themselves. That is

$$(I + \Delta t \partial I_Z)^{-1}(f) = \max(f(s), g(s)),\ \forall s \in \Omega_s \tag{A.18}$$

Which is precisely the projection that can be seen in both Brennan-Schwartz and PSOR.

A.8 Justifying the approximation for the reference values

To calculate an error, we need a reference value. For a European option we could compare a numerical method directly to the analytical answer. However, we need to use numerical methods because our model problem has no analytical expression. Hence, we use a binomial method with a large number of time steps to generate reference values.[3]

First, to demonstrate the validity of the apporach, we perform a calculation using a European put option, comparing it to the analytical Black-Scholes solution for the same option. We present a truncated image to illustrate the convergence to the analytical solution. The larger image, where the number of steps maximizes at 14 000, is less illustrative as the curvature where the numerical approximation approaches the analytical solution is less obvious.

[3] Although we are not interested in the calculation when calculating the reference values, it is not entirely unimportant, as generating the reference values is time consuming.

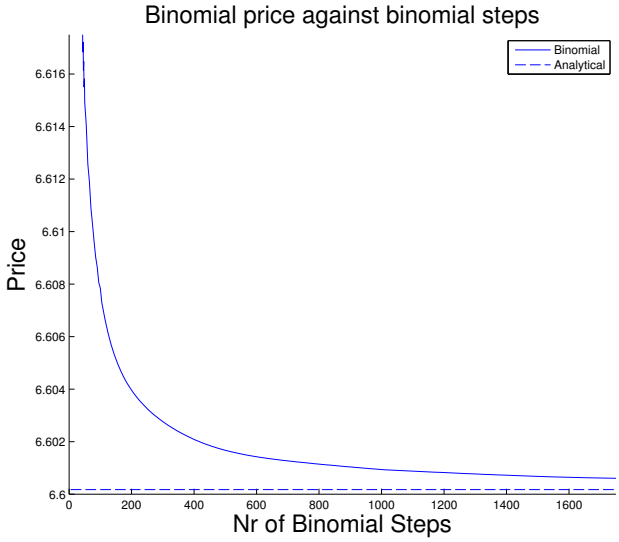

Figure A.1. A truncated picture of a *European* put option price for a strike K of 100, a spot price of 105, an interest rate of 0.03 and an underlying volatility set at the constant 0.2. The dashed line is the analytical price for the same put option using Black-Scholes.

From the same calculation, at approximately 7500 binomial steps, the error between the binomial approximation and the analytical calculation drops below 10^{-4}.

Using the same input parameters as for the European option above, but this time for an American put option, we compare the binomial method for a few different time steps with a binomial method (for an American put option still!) using 80 000 time steps. This shows that while 80 000 time steps in the binomial method gives a more accurate a reference value, the "reference error" - which is the error between the chosen reference value and the real value - is going to be insignificant when compared to our error tolerance long before that. The result is displayed in figure A.2.

And it is from figure A.2 that we elect to use 12 000 time steps when generating our standard reference values, as the error between that and the true value is well below our error tolerance, and the calculation time remains within reasonable limits.

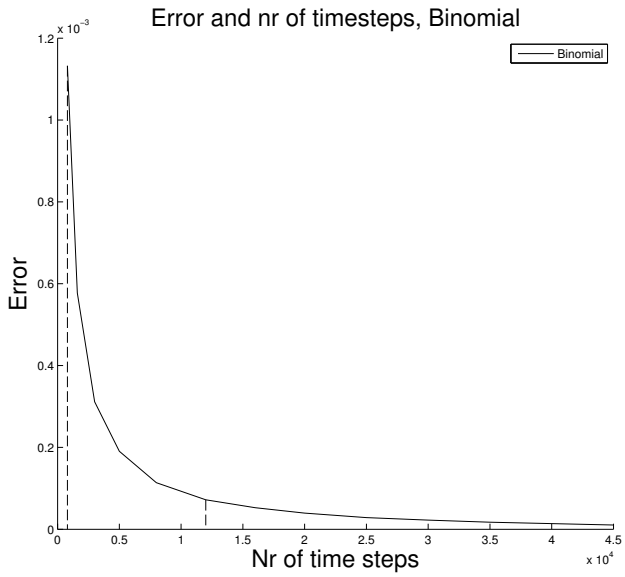

Figure A.2. Shows the difference between the price of an American put option (with the same input parameters as in A.1) calculated with the binomial method using 80000 time steps and the price of the same calculated with time steps in the range $[800, 45000]$.

A.9 Empirical tests for the numerical parameters

Both the Ikonen-Toivanen and the Brennan-Schwartz methods take two input parameters, time steps N and spatial steps M, with large effects on performance both in terms of time and accuracy.

In order to perform a useful comparison with the method, we need to examine how changes in N and M affect the accuracy of the solution as well as computational time. We note that we have no reason to believe that any of the three FD methods will attain optimum performance for, say, quadratic grids, or that the same grid shape would create an optimum performance for all methods - the result depends on the nature of the error terms that arise from both the choice of discretization for the "basic" problem as well as the algorithm employed to solve the succession of LCPs. For practical application, this knowledge does not benefit us greatly, as the optimal grid varies depending on the problem parameters - choosing bounds on error tolerance and calculation time, we will still need a different optimal grid if the underlying option changes strike or if the spot price of the underlying changes. If the option type is changed, then the changes to the optimal grid may be unpredictable without carrying out a detailed analysis of the error terms, and calculating the optimal grid shapes is computationally costly regardless of the changes made (to the tune of several orders of magnitude more expensive than using a suboptimal

near-randomly selected grid that is further refined, i.e. higher N and M). First a description of how we examine this empirically - we do not perform a detailed theoretical analysis of the problem - and then a description of why it is done despite these limitations.

How
We introduce grid size G_s, according to $G_s = (M+1) \cdot (N+1)$ which is the number of *points* for a calculation that has M space *steps* and N time *steps*. The purpose of introducing this grid size is to acquire appropriate shapes of the grid, by comparing the numerical error for various shapes of the grid. We opt to examine the L^∞-norm, that is, the maximum absolute error. [4]

Every FDM calculation will return as an answer a solution vector. Using values from the binomial method as a reference, we find the error for every element in the FDM solution vector. We then select the maximum of the absolute errors in that vector to represent the current grid shape. The shape is then changed, and the procedure repeated. For every choice of "guiding" G_s we then choose the shape that showed the smallest maximum absolute error.

Due to computational limitations, the reference values used to calculate the optimal grid shape are acquired from a binomial calculation with 800 time steps. Using the example presented in figure A.2, but putting it in numbers, the difference between the 800 steps we use and 80 000 steps is of the order 10^{-3}. By contrast, the difference between 12 000 and 80 000 is of the order 10^{-5}. Note that this means when 800 time steps is used instead of 12 000, the reference error can *not* be ignored.

Why
We have already noted that the optimal grid shapes changes with the problem parameters - it is not *solely* a property of the numerical method, although it is different depending on the method - as well as prohibitively expensive to compute. This obviously limits its use for practical comparisons. However, the existence of an optimal shape within a size means that the naive approach, where we select N and M almost at random and compare the methods on the same grid shape and size, risks producing skewed results where one of the methods is tested on shapes that are much closer to its optimal shape. By deciding on the grid shapes individually per method, our results are more valid, and easier to implement.

A.9.1 Grid size and its effects on the error

Below is a plot of the maximum absolute error (L^∞-error) against grid size for an American put option, where $r = 0.03$, $\sigma = 0.2$, $T = 2$ and $K = 100$. It was generated by first selecting a number of grid sizes, calculating the optimal grids of

[4] In the appendix A.10, a plot of the error along the entire interval of spot prices in the domain is shown, to illustrate how the maximum absolute error relates to the "typical" error.

those using reference binomials with a step size of 800, and then using the optimal grid shapes to represent their grid size. If two grid shapes have the same error, the grid shape with the highest M has been selected as the optimum grid.

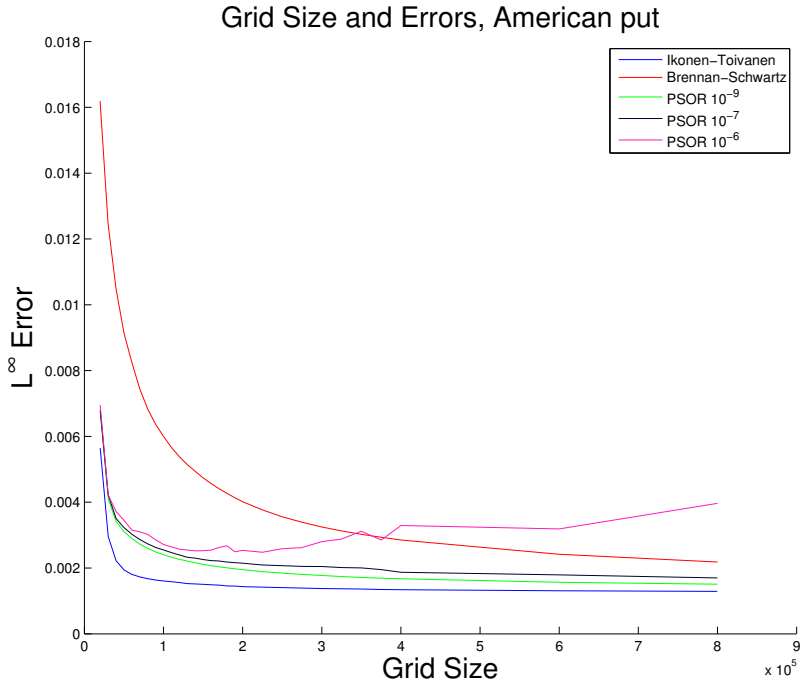

Figure A.3. Each grid size is represented by its optimal shape. Note that the optimal shape is calculated for a reference value using 800 binomial steps, which means that the reference error is visible (subtracting a bit more than 10^{-3} - the difference between the price for 800 time steps and 80 000 time steps - would remove enough of the reference error to make it insignificant, leaving the numerical error the primary contributor.)

As we can see in Figure A.3, PSOR with an error tolerance of 10^{-6} gives a very disturbing result - it doesn't appear to actually converge! Recall what was said in 4.1.5, as what we see here is an example of the behaviour described there. The grids examined later will be much smaller than the grids used above, and one of the side effects of this is that lower error tolerances are acceptable. The other two PSOR error tolerances appear to converge, faster - in relation to *grid size*, not *calculation time*! - than Brennan-Schwartz but slower than Ikonen-Toivanen. Note that the reference error is visible and non-trivial. There is a slight variation in the numerical error when comparing binomial calculations with difference time steps, which means that a straight parallel shift does not quite work to remove the reference error cleanly (leaving the graph looking significantly less smooth, as the optimal grids change).

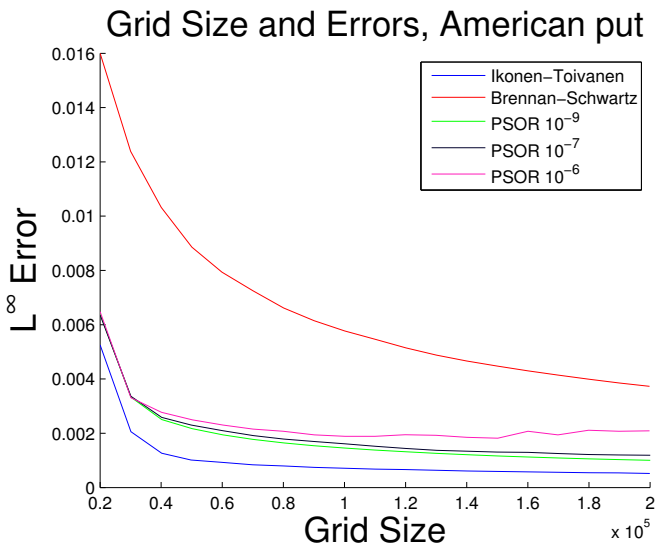

Figure A.4. Each grid size is represented by its optimal shape. This time the optimal shape is calculated for a reference value using **5000** binomial steps, reducing the reference error to below 10^{-3}, and we limit our attention to a smaller number of unknowns.

As we can see in Figure A.4, the graph is not *quite* the same as when the reference value is calculated using 800 time steps, but it has maintained the more important properties.

Remark As we do not expect to find the optimal shapes amongst grids that are extremely one-sided (e.g. M or N multiple orders of magnitude larger than the other) we save computational time by not examining such grids. In general, at least 100 points in either direction is required for a shape to be considered.

A.9.2 Within a grid size

Relevant to our examination is the behaviour of the error for non-optimal shapes within a grid size. The below plot uses an American put option with the same "standard" parameters as previously described, $r = 0.03$, $\sigma = 0.2$, $T = 2$ and $K = 100$. The jagged appearance of the plots comes from the integer rounding in selecting N and M based on a G_s - we only manage to *approximately* fixate the number of unknowns with the guiding G_s. These plots still provides an overview of what we can expect from selecting certain grid shapes, the first for Ikonen-Toivanen and Brennan-Schwartz, and the second for the three examined PSOR variants.

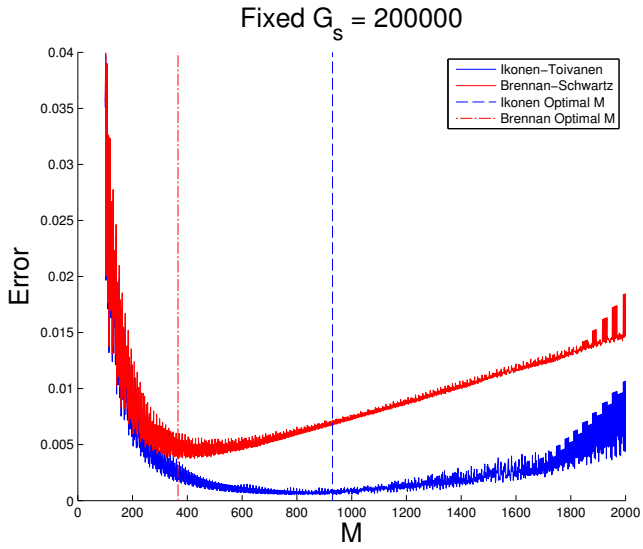

Figure A.5. The binomial reference values are calculated with 5000 time steps, the grid size is 200 000. The M belonging to the optimal grid shape has been marked.

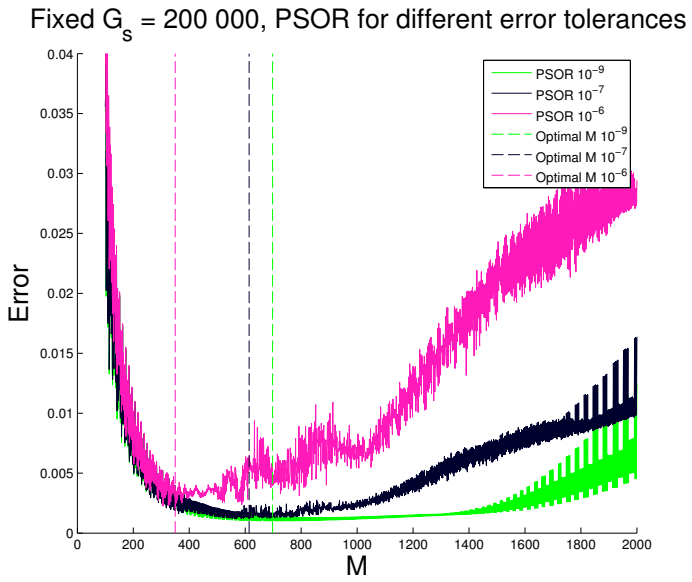

Figure A.6. The reference values are calculated with 5000 time steps, the grid size is 200 000. The M belonging to the optimal grid shape has been marked

The most interesting thing to note in Figure A.5 and Figure A.5 are the differences in how sensitive they are to changes in shape. A flat curve means that the

error increases less if we employ a non-optimal grid shape.

We can now from Figure A.4 focus our examination on smaller grid sizes, and from figures A.5 and A.6 we can constructs rules for the shapes within those grid sizes.

A.10 Motivating the L^∞ error

For practical understanding of the error measures we have chosen, it may be interesting to study the following plot of the pointwise error of the Ikonen-Toivanen method. The reason for excluding the other methods is that although the graph and the scale changes, the appearance is similar.

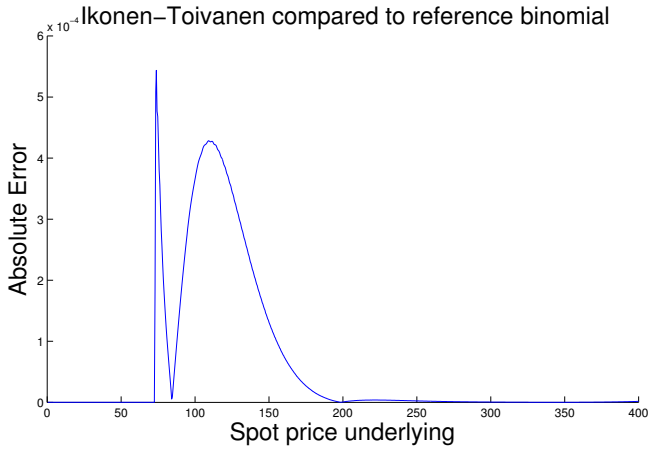

Figure A.7. The option has the parameters $r = 0.03$, $\sigma = 0.2$, $T = 2$ and $K = 100$. The Ikonen-Toivanen method is deployed with $M = 850$, $N = 235$. The binomial reference values are calculated using 12 000 time steps.

The highest peak gives the error value used, and is in some manner a worst-case scenario - though its proximity to the strike means that it is not necessarily the least *likely* scenario!

Bibliography

[1] Jim Jr. Douglas and H. H. Jr. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, 82(2):pp. 421–439, 1956.

[2] Michael J. Brennan and Eduardo S. Schwartz. The valuation of american put options. *The Journal of Finance*, 32(2):pp. 449–462, 1977.

[3] P. Lions. Une methode iterative de resolution dúne inequation variationnelle. *Israel Journal of Mathematics*, 31:204–208, 1978. 10.1007/BF02760552.

[4] John C. Cox, Stephen A. Ross, and Mark Rubinstein. Option pricing: A simplified approach. *Journal of Financial Economics*, 7(3):229 – 263, 1979.

[5] P. L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):pp. 964–979, 1979.

[6] G.I. Marchuk. Splitting and alternating direction methods. volume 1 of *Handbook of Numerical Analysis*, pages 197 – 462. Elsevier, 1990.

[7] Jonathan Eckstein and Dimitri P. Bertsekas. On the Douglas Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55:293–318, 1992.

[8] Steven L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The Review of Financial Studies*, 6(2):pp. 327–343, 1993.

[9] Mark Broadie and Jerome Detemple. American option valuation: New bounds, approximations, and a comparison of existing methods. *The Review of Financial Studies*, 9(4):pp. 1211–1250, 1996.

[10] Jonathan Eckstein and Michael C. Ferris. Operator-Splitting Methods for Monotone Affine Variational Inequalities, with a Parallel Application to Optimal Control. *INFORMS Journal on Computing*, 10:218–235, 1998.

[11] S. G. Kou. A jump-diffusion model for option pricing. *Manage. Sci.*, 48(8):1086–1101, August 2002.

[12] Roland Glowinski. Finite element methods for incompressible viscous flow. In P.G. Ciarlet and J.L. Lions, editors, *Numerical Methods for Fluids (Part 3)*, volume 9 of *Handbook of Numerical Analysis*, pages 3 – 1176. Elsevier, 2003.

[13] Samuli Ikonen and Jari Toivanen. Operator splitting methods for American option pricing. *Applied Mathematics Letters*, 17(7):809 – 814, 2004.

[14] Michael B. Giles and Rebecca Carter. Convergence analysis of Crank-Nicolson and Rannacher time-marching. *Journal of Computational Finance*, 9(4):89–112, 2006.

[15] Samuli Ikonen and Jari Toivanen. Componentwise splitting methods for pricing American options under stochastic volatility. *International Journal of Theoretical & Applied Finance*, 10(2):331 – 361, 2007.

[16] Samuli Ikonen and Jari Toivanen. Pricing American options using LU decomposition. *Applied Mathematical Sciences*, 1(51):2529 – 2551, 2007.

[17] Samuli Ikonen and Jari Toivanen. An operator splitting method for pricing american options. 16:279–292, 2008.

[18] Tomas Björk. *Arbitrage Theory in Continuous Time*. Oxford University Press, third edition, 2009.

[19] R.W. Cottle, J.S. Pang, and R.E. Stone. *The Linear Complementarity Problem*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 2009.

[20] John C. Hull. *Options, Futures and Other Derivatives*. Pearson Education Limited, 2011.

[21] R.T. Rockafellar. *Convex Analysis*. Princeton Mathematical Series. Princeton University Press, 1970.

[22] D. Tavella and C. Randall. *Pricing Financial Instruments, The Finite Difference Method*. John Wiley & Sons, 2000.

[23] R. Trémolières, J.L. Lions, and R. Glowinski. *Numerical Analysis of Variational Inequalities*. North-Holland Publishing Company, 1981.

[24] William T. Vetterling Brian P Flannery William H. Press, Saul A. Teukolsky. *Numerical recipes in C : the art of scientific computing*. Cambridge University Press, Cambridge Cambridgeshire New York, 1992.

[25] Paul Wilmott, Sam Howison, and Jeff Dewynne. *The Mathematics of Financial Derivatives*. Press Syndicate of the University of Cambridge, 1995.

TRITA-MAT-E 2013:17
ISRN-KTH/MAT/E--13/17-SE

www.kth.se