Unterschrift des Betreuers

TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

DIPLOMARBEIT

A numerical solver for the multivariate Black-Scholes problem using the multigrid method

Ausgeführt am Institut für

Analysis und Scientific Computing

der Technischen Universität Wien

unter der Anleitung von

Ao.Univ.Prof. Dipl.-Math. Dr.techn. Dirk Praetorius

durch

Claudia Satke MSc

Carl Zwillinggasse 39/2

2340 Mödling

8. Juni 2009

TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

DIPLOMA THESIS

A numerical solver for the multivariate Black-Scholes problem using the multigrid method

written at the

Institute for Analysis and Scientific Computing

of the Vienna University of Technology

supervised by

Ao.Univ.Prof. Dipl.-Math. Dr.techn. Dirk Praetorius

by

Claudia Satke MSc

Carl Zwillinggasse 39/2

2340 Mödling

Austria

June 8, 2009

Abstract

This work examines the Black-Scholes PDE for multi-asset options on an arbitrary number of underlying assets and develops a numerical solution method based on a finite-difference discretisation, involving the multigrid method for iteratively solving the resulting large sparse systems. The emphasis is on designing a numerical algorithm that involves the number of underlying assets, i.e. the spatial dimension of the PDE, as a mere parameter and is thus simultaneously applicable to any number of underlying assets. A vectorised MATLAB implementation of this algorithm is presented and tested on realistic pricing of options on one, two, and three underlyings.
Theoretical considerations at the outset involve solvability and uniqueness results for the multivariate Black-Scholes PDE and its truncation to a bounded domain and the derivation of a localisation error estimate.

Acknowledgements

First and foremost, I wish to thank Dr Dirk Praetorius who made it possible for me to pursue in my thesis the subject of my choice, who guided and supported me in the process of realising it, and who provided valuable inputs while leaving me a generous amount of freedom in developing the conception of this work.
I owe the interest in mathematical finance to Dr Simon Shaw from Brunel University, West London, where I spent an inspiring year of my studies.

I sincerely thank my family for their support at all times, moral as well as financial. My parents' and sister's patient aid, advice and encouragement have essentially helped me to achieve my goals. Special thanks go to Bernd for his love and encouragement.
Lastly, the support I got from friends and fellow students must not go unmentioned.

Contents

Introduction

Options are derivative financial products that involve the right but not the obligation to trade in one or several risky assets, i.e. assets whose future price is not deterministic, like for instance, shares, commodities, or currencies, at a fixed predefined price. We will only consider so-called *European-style* options where the option can only be exercised at *one* specified date in future, the so-called *maturity* or *expiry date*.

On the financial markets, increasingly sophisticated and complex financial instruments are being developed and need to be correctly priced. A considerable variety of derivative products can be interpreted as involving an option on several underlying assets, hence the pricing of *multi-asset options* is of both theoretical and practical interest in mathematical finance. Let us present some examples. [Margrabe, 1978] shows that performance incentive fees, margin accounts, exchange offers, and standby commitments constitute options to exchange one asset for another, i.e. options contingent on two underlying assets. [Stulz, 1982] discusses applications of options on the maximum or the minimum of two assets, including the valuation of foreign currency bonds, option-bonds, risk-sharing and incentive contracts, secured debt, and certain types of investment opportunities. Furthermore, [Boyle et al., 1989] and [Boyle and Tse, 1990] consider the so-called quality option or "cheapest to deliver" option which occurs in future contracts when the short position can deliver any one of a set of deliverable assets, and which can be expressed as an option on the minimum of this (arbitrarily large) set. [Barraquand, 1995] and [Barraquand and Martineau, 1995] list applications of multidimensional option pricing in assets and liability management, corporate capital budgeting, risk management, property/liability insurance, and the pricing of Over The Counter warrants, multidimensional interest rate term structure contingent claims, mortgage-backed securities, and life insurance policies.

In 1973, Fisher Black, Myron Scholes, and Robert Merton published a pioneering model allowing to uniquely determine the price of derivative products, which has since been known as the *Black-Scholes model*. Since then, a vast literature on option pricing theory has emerged. There are two equivalent ways to derive the option price according to the Black-Scholes model (see, e.g., [Wilmott et al., 1995]): one approach derives a partial differential equation (PDE) that governs the option price depending on the price(s) of the underlying asset(s) and time, and the other represents the option price as the discounted expected value of the option's pay-off under an equivalent martingale measure, the so-called risk neutral measure. There are only few cases where the resulting PDE or multidimensional integral, respectively, can be solved analytically, hence one must resort to numerical valuation methods.

In this thesis, we will concentrate on the PDE approach and design a numerical algorithm based on finite differences to solve the PDE for multi-asset options. In particular, our objective is that

the algorithm should be applicable to options on an *arbitrary* number of underlying assets, or in mathematical terms, to PDEs in arbitrary space dimensions.

In Chapter 1, we present the Black-Scholes PDE and its multi-dimensional generalisation, transform the problem to a more convenient form, and prove the existence and uniqueness of a solution based on theorems by [Friedman, 1964]. However, this problem is set on an unbounded domain, which is not suitable for a finite-difference method. Thus, for the numerical solution, we have to truncate the problem to a bounded computational domain. In Section 1.4, we analyse the truncated problem, its solvability, and the localisation error caused by the truncation. Based on results of [Kangro and Nicolaides, 2000] and [Hilber et al., 2004], we derive an estimate of this error in Section 1.4.3 that proves convergence of the solution of the truncated problem to the original solution and hence justifies the truncation.

In Chapter 2, we present the finite-difference discretisation that we use, and give a brief discussion of the convergence of such numerical schemes. The finite-difference method results in a set of linear systems of equations which are in general very large and sparse. In order to solve them, we use the highly efficient iterative *multigrid method* whose basic principles we explain in Chapter 3.

After that, we turn to the practical implementation of our numerical solver in Chapter 4. In particular, we show how to implement all required algorithms in MATLAB in a vectorised way and for an a-priori arbitrary number of spatial dimensions. Subsequently, we test the performance of our solver in a number of numerical experiments presented in Chapter 5.

Our findings are finally summarised in Chapter 6, and the pros and cons of our numerical method with respect to other numerical valuation strategies found in literature are briefly discussed.

Chapter 1

The Black-Scholes equation for multi-asset options

1.1 The option pricing model due to Black, Scholes and Merton

In 1973, Fischer Black and Myron Scholes published a seminal new approach to option valuation in their famous paper [Black and Scholes, 1973]. Robert C. Merton was also involved in the development and enhancements of this model. In 1997 (when Black had already died), Scholes and Merton were awarded the Nobel Prize in Economics for this option pricing model.

The pioneering merit of the Black-Scholes model is the fact that for the first time in option valuation theory, the price of an option could be determined purely from (theoretically) observable parameters and did not depend on investors' expectations about the future development of stock prices (see also the account by [Schwartz, 1977]). The fundamental underlying concept is the *no arbitrage principle*, put the following way in the abstract of [Black and Scholes, 1973]:

> *If options are correctly priced in the market, it should not be possible to make sure profits by creating portfolios of long and short positions in options and their underlying stocks.*

Thus, calculating the number of options that have to be sold short in order to create a hedged (i.e. independent of the uncertain development of the stock price) position together with a long position in one share of stock, and observing that in market equilibrium, the return on this portfolio must be equal to the return on a riskless asset, they derive a differential equation for the option price as a function of the stock price and time, based on dynamics obeying *stochastic calculus* (Itô's formula).

This model is based on a number of assumptions corresponding to "ideal market conditions"[1]:

1. The (short-term) riskless interest rate r is known and constant during the lifetime of the option (typically a few months).

[1]Some of these assumptions can be dropped. There are extensions of the original Black-Scholes model which incorporate, e.g., time-dependent deterministic or stochastic interest rates and volatilities, options on stocks that pay continuous or discrete dividends, options of American type which can be exercised at any time up to the maturity date, and more complex "exotic" options, see, e.g., [Wilmott et al., 1995], [Günther and Jüngel, 2003], or [Seydel, 2004].

2. The stock price follows a *random walk* with the instantaneous rate of return satisfying the stochastic differential equation
$$\frac{dS}{S} = \mu\, dt + \sigma\, dX \tag{1.1}$$
where S is the stock price, t is time, μ is the instantaneous rate of return on the stock (also called *drift*), $\sigma > 0$ is the instantaneous standard deviation of the rate of return (also called *volatility* of the stock price), and dX is a *Wiener process*, i.e. a continuous-time stochastic process which starts from zero and has independent increments which are distributed according to a normal distribution with mean zero and variance dt. The drift μ and volatility σ are assumed to be known and constant through time.

   A stochastic process satisfying (1.1) is said to follow a *Geometric Brownian motion*.

3. The stock pays no dividends during the life of the option.

4. The option is of *European type*, i.e. it can only be exercised at one specified maturity date (expiry date) T.

5. There are no transaction costs associated with buying or selling assets. Trading takes place in continuous time.

6. All assets are perfectly divisible, i.e. any quantity can be traded. Lending as well as borrowing of any amount is possible at the riskless interest rate.

7. Short selling is permitted without penalties and without restrictions.

Under these assumptions, [Black and Scholes, 1973] show that the value $V(S,t)$ of the option is governed by the partial differential equation known as the *Black-Scholes equation*,
$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0, \qquad S > 0,\ t \in [0,T). \tag{1.2}$$

Here, S denotes the price of the underlying asset, t is time (in years), σ is the volatility of the underlying asset, r is the riskless interest rate (more precisely, the annual rate of continuously compounded interest), and T is the maturity date of the option. This PDE needs to be supplemented by a terminal condition
$$V(S,T) = V_0(S) \tag{1.3}$$
giving the value of the option at maturity. The no arbitrage principle shows that this must be equal to the pay-off of the option. Furthermore, boundary conditions for $V(0,t)$ and $\lim_{S\to\infty} V(S,t)$ derived from economic considerations are usually prescribed. However, this derivation is often ambiguous, and boundary conditions are actually not mathematically necessary as we will see later. We will discuss this issue in detail in Section 1.2.1.

The *pay-off function* V_0 depends on the type of option. Consider for example a *European call option*, i.e. an option giving its holder the right, at the prescribed time T, to purchase one unit of the underlying asset at the preset *exercise price* (strike price) $E \geq 0$. If at the maturity date, the price of the underlying is below the exercise price, there is no gain in exercising the option, as the underlying asset can be purchased at a lower price on the market. Hence, if $S < E$ at $t = T$, the

pay-off of the option is zero. If $S > E$ at expiry, however, the buyer of the option may purchase the underlying asset at the price E and instantaneously sell it on the market at the price S, thereby making a profit of $S - E$. Thus, the pay-off function of a European call option is given by

$$V_0(S) = \max\{S - E,\, 0\}. \tag{1.4}$$

By analogous considerations, the pay-off function of a *European put option* giving the right to *sell* one unit of the underlying asset at the exercise price E at time T is found to be

$$V_0(S) = \max\{E - S,\, 0\}. \tag{1.5}$$

As for the boundary conditions, the case is rather clear for a European put option: if $S = 0$ at any time t_0, the Geometric Brownian motion model implies that $S = 0$ *with certainty* for any $t \geq t_0$, in particular at the maturity date T such that according to (1.5), the pay-off of the put option at maturity is E with certainty. The no arbitrage principle implies that the value of a *certain* future cashflow E is $E\mathrm{e}^{-r(T-t)}$ at time t, i.e. it is discounted by the riskless interest rate. Therefore, we have $V(0,t) = E\mathrm{e}^{-r(T-t)}$ for a European put option. If S is very large, on the other hand, it will almost certainly not pay to exercise the put, such that we can say that $\lim_{S\to\infty} V(S,t) = 0$.

For a European call option, if $S = 0$, the pay-off at maturity is certainly zero, such that we have $V(0,t) = 0$. In the limit $S \to \infty$, however, the case is more difficult. If S is very large, the option is almost certain to be exercised and yield a pay-off of $S - E$ at maturity. The value of the option will therefore approximately behave like $S - E\mathrm{e}^{-r(T-t)}$ as $S \to \infty$, or neglecting $E\mathrm{e}^{-r(T-t)}$ compared to S, we can say $V(S,t) \sim S$ for all t (i.e. $\lim_{S\to\infty} \frac{V(S,t)}{S} = 1$); a boundary condition of this form is used in [Wilmott et al., 1995], [Günther and Jüngel, 2003], and [Seydel, 2004]. [Hilber et al., 2004] set

$$V(S,t) \sim \mathrm{e}^{-r(T-t)}\, V_0\big(S\mathrm{e}^{r(T-t)}\big) \quad \text{at the boundaries} \tag{1.6}$$

for any type of option, which has the economic interpretation that if the stock price were deterministic, a stock price of S at time t would correspond to $S\mathrm{e}^{r(T-t)}$ at time T, and the value of the option is the discounted corresponding pay-off. This formula yields precisely the above cited boundary conditions for European puts and calls, with the limiting condition $V(S,t) \sim S - E\mathrm{e}^{-r(T-t)}$ as $S \to \infty$ for European calls. There are other ways of expressing this approximate limiting behaviour of European call options, however; [Schwartz, 1977] uses $\lim_{S\to\infty} \frac{\partial V(S,t)}{\partial S} = 1$, and [Persson and von Sydow, 2007] assume that the option price is nearly linear with respect to S at the boundaries for any type of option, hence $\frac{\partial^2 V(S,t)}{\partial S^2} = 0$ at the boundaries.

1.2 Generalisation to multi-asset options

A more general type of options does not only depend on one, but on a set of n underlying assets. This is the type we will consider from now on. Such options are called *multi-asset options*, although alternative names such as *rainbow options* (e.g. in [Seydel, 2004], [BusinessDictionary.com, 2009]) or *basket options* (in [Wilmott, 1998]) are sometimes used which may also refer to a special type of multi-asset options.

The same principles as in the case of single-asset options can be used to price multi-asset options as well. All assumptions of the Black-Scholes model given above are retained, except that we now

assume that the n underlying asset prices $S_1, \ldots, S_n$ follow an *n-dimensional Geometric Brownian motion*, i.e. the rates of return satisfy the stochastic differential equations

$$\begin{aligned} \frac{dS_1}{S_1} &= \mu_1 dt + \sigma_1 dX_1, \\ &\vdots \\ \frac{dS_n}{S_n} &= \mu_n dt + \sigma_n dX_n \end{aligned} \tag{1.7}$$

with constant drifts $\mu_1, \ldots, \mu_n$ and volatilities $\sigma_1, \ldots, \sigma_n$, respectively, and $dX_1, \ldots, dX_n$ are *correlated* Wiener processes with known constant correlation coefficients ρ_{ij} for all dX_i, dX_j. The corresponding covariance matrix

$$\Sigma := (\sigma_i \sigma_j \rho_{ij})_{i,j=1}^n \tag{1.8}$$

is required to be positive definite, i.e.

$$\boldsymbol{\xi}^T \Sigma \, \boldsymbol{\xi} = \sum_{i=1}^n \sum_{j=1}^n \sigma_i \sigma_j \rho_{ij} \, \xi_i \xi_j > 0 \quad \text{for all } \boldsymbol{\xi} \in \mathbb{R}^n \setminus \{\mathbf{0}\}. \tag{1.9}$$

Denote $\mathbf{S} := (S_1, \ldots, S_n)$ and consider the price $V(\mathbf{S}, t)$ at time t of a European-style multi-asset option with maturity date T. An analogous derivation as in the case of single-asset options, based on the no arbitrage principle, the construction of a hedged portfolio, and dynamics according to the n-dimensional version of Itô's formula (see, e.g., [Engelmann and Schwendner, 1998], or [Wilmott, 1998] for a full derivation) shows that V is then governed by the partial differential equation

$$\frac{\partial V}{\partial t} + \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \sigma_i \sigma_j \rho_{ij} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} + r \sum_{i=1}^n S_i \frac{\partial V}{\partial S_i} - rV = 0, \qquad \mathbf{S} \in (0, \infty)^n, \ t \in [0, T), \tag{1.10}$$

known as the *n-dimensional Black-Scholes PDE*, subject to the terminal condition

$$V(\mathbf{S}, T) = V_0(\mathbf{S}) \qquad \text{for all } \mathbf{S} \in [0, \infty)^n \tag{1.11}$$

where $V_0(\mathbf{S})$ is the pay-off function of the option. We will further analyse this differential problem in Section 1.3.

We will now present some important types of multi-asset options and their respective pay-off functions.

- A *European basket call* with exercise price E gives its holder the right, at time T, to purchase a predefined portfolio consisting of c_i units of each underlying asset, respectively. Conversely, a *European basket put* gives the right to sell this portfolio at the exercise price E.
- A *European call on the maximum of n assets* with exercise price E gives its holder the right to purchase the asset whose price at time T is the highest at the exercise price E. More general, individual exercise prices E_i may apply to each underlying asset S_i. Similarly, there exist calls on the *minimum* of n assets. Both types may also occur as put options.
- An *exchange option* is a European-style two-asset option which gives its holder the right, at time T, to exchange q_2 shares with price S_2 for q_1 shares with price S_1.

The pay-off functions for these types of multi-asset options are listed in Table 1.1. Note that the pay-off is always non-negative, as an option never entails any cost for the holder at expiry.

option type	pay-off function $V_0(\mathbf{S})$
European basket call	$\max\{\sum_{i=1}^n c_i S_i - E, 0\}$
European mean-value basket call	$\max\{\frac{1}{n}\sum_{i=1}^n S_i - E, 0\}$
European basket put	$\max\{E - \sum_{i=1}^n c_i S_i, 0\}$
European call on the geometric average	$\max\{(\prod_{i=1}^n S_i)^{\frac{1}{n}} - E, 0\}$
European put on the geometric average	$\max\{E - (\prod_{i=1}^n S_i)^{\frac{1}{n}}, 0\}$
European call on the maximum of n assets	$\max\{\max_{i=1,\dots,n}\{S_i - E_i\}, 0\}$
European put on the maximum of n assets	$\max\{\max_{i=1,\dots,n}\{E_i - S_i\}, 0\}$
European call on the minimum of n assets	$\max\{\min_{i=1,\dots,n}\{S_i - E_i\}, 0\}$
European put on the minimum of n assets	$\max\{\min_{i=1,\dots,n}\{E_i - S_i\}, 0\}$
Exchange option	$\max\{q_1 S_1 - q_2 S_2, 0\}$

Table 1.1: Pay-off for several types of multi-asset options.

1.2.1 A note on the boundary conditions

Concerning the boundary conditions, the case is more difficult for multi-asset options. If $\mathbf{S} = \mathbf{0}$ or $S_i \to \infty$ for all i simultaneously, similar considerations as in the one-dimensional case are possible. On the boundary parts where $S_j = 0$ or $S_j \to \infty$ for some j while the remaining S_i are arbitrary, however, there is in general no clear result. On the $S_j = 0$ boundaries, one might reduce the dimensionality of the problem by omitting S_j from the set of underlying assets and hence consider the option as one on $n-1$ assets, and iterate this procedure for the boundaries of the new domain until the one-dimensional case is reached. However, this method is in general too complex for practical application. A simpler method is to observe that $V_0(\mathbf{S})$ approximates the boundary behaviour in the one-dimensional case, and hence to set

$$V(\mathbf{S}, t) \sim V_0(\mathbf{S}) \quad \text{at the boundaries.} \tag{1.12}$$

A probably more refined approximation is obtained by carrying the economic interpretation of Formula (1.6) over to the multi-dimensional case, resulting in

$$V(\mathbf{S}, t) \sim \mathrm{e}^{-r(T-t)}\, V_0\big(\mathbf{S}\,\mathrm{e}^{r(T-t)}\big) \quad \text{at the boundaries.} \tag{1.13}$$

Alternatively, [Persson and von Sydow, 2007] assume that the option price is approximately linear with respect to each asset price at the boundaries, i.e.

$$\frac{\partial^2 V(\mathbf{S}, t)}{\partial S_i^2} = 0 \quad \text{for all } i \text{ at the boundaries.} \tag{1.14}$$

They point out in addition that financial problems are often parabolic problems with a significant diffusion part (cf. Section 1.4.1) implying that changes in the boundary conditions do not strongly affect the solution in the interior of the domain.

Even though boundary conditions are stated in all literature cited above, they are strictly speaking not necessary for the Black-Scholes problem at all (cf. [Wilmott et al., 1995]). The mathematical purpose of boundary conditions is to make the solution of a differential problem unique. We will see in Section 1.3.2, however, that the n-dimensional Black-Scholes PDE on $(0,\infty)^n$ can be transformed to a PDE on $\mathbb{R}^n$ and that this PDE together with an appropriate terminal condition has a unique solution. Not only are no specific conditions on the behaviour of the solution at the "boundaries" needed for the solvability of the problem, but imposing any conditions other than the actual behaviour of the solution determined by the terminal condition alone makes the problem unsolvable. It thus makes sense to define the Black-Scholes problem solely as the PDE (1.10) together with the terminal condition (1.11).

Nevertheless, there is a practical need for considerations regarding the behaviour of the solution near the boundaries: whereas this is not relevant for the original Black-Scholes problem on $(0,\infty)^n$, the numerical solution of this problem requires the truncation to a finite domain, and the differential problem on the finite domain does need boundary conditions. If the computational domain is chosen suitably large, it makes sense to assume these boundary data close to the boundary value of the original solution. We will discuss this issue in detail in Section 1.4.4.

1.3 Analysis of the n-dimensional Black-Scholes PDE

1.3.1 A change of variables

The problem we aim to solve is the n-dimensional Black-Scholes problem (1.10)–(1.11). This is a linear second order parabolic problem in backward time which is degenerate at $S_i = 0$ (see Definition 1 below). It is possible, however, to transform this problem to a more convenient form both for analytical and numerical treatment (see, e.g., [Brennan and Schwartz, 1978], [Wilmott et al., 1995], [Kangro and Nicolaides, 2000], [Hilber et al., 2004], [Persson and von Sydow, 2007] for slightly different transform variants).

Firstly, the backward final value problem can be transformed to a customary forward initial value problem by considering the time to expiry,

$$\tau := T - t, \tag{1.15}$$

instead of calendar time t. Furthermore, the degeneracy with respect to the price variables S_i can be eliminated by switching to the log-price variables

$$x_i := \ln S_i. \tag{1.16}$$

With the changes of variables (1.15) and (1.16) and

$$w(\mathbf{x},\tau) := V(\mathbf{S},t) \tag{1.17}$$

where we set $\mathbf{x} := (x_1,\dots,x_n)$, the problem (1.10)–(1.11) transforms to

$$\frac{\partial w}{\partial \tau} - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_i\sigma_j\rho_{ij}\frac{\partial^2 w}{\partial x_i \partial x_j} + \sum_{i=1}^{n}\Big(\frac{\sigma_i^2}{2} - r\Big)\frac{\partial w}{\partial x_i} + rw = 0, \qquad \mathbf{x}\in\mathbb{R}^n,\ \tau\in(0,T], \tag{1.18}$$

subject to the initial condition

$$w(\mathbf{x},0) = V_0(\mathrm{e}^{x_1},\dots,\mathrm{e}^{x_n}) \qquad \text{for all } \mathbf{x}\in\mathbb{R}^n. \tag{1.19}$$

This problem is uniformly parabolic in $\mathbb{R}^n$ according to the definition of [Friedman, 1964]:

Definition 1. Consider a PDE of the form

$$\frac{\partial w}{\partial \tau} - \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}(\mathbf{x},\tau)\frac{\partial^2 w}{\partial x_i \partial x_j} + \sum_{i=1}^{n} b_i(\mathbf{x},\tau)\frac{\partial w}{\partial x_i} + c(\mathbf{x},\tau)w = 0 \tag{1.20}$$

in $\overline{\Omega}\times[T_0,T_1]$ where Ω is a domain in $\mathbb{R}^n$ and $A(\mathbf{x},\tau) := (a_{ij}(\mathbf{x},\tau))_{i,j=1}^n$ is assumed to be a symmetric matrix. (1.20) can be written in operator notation as

$$\frac{\partial w}{\partial \tau} + \tilde{L}w = 0 \tag{1.21}$$

with the linear differential operator $\tilde{L}$ defined by

$$\tilde{L}w := -\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}(\mathbf{x},\tau)\frac{\partial^2 w}{\partial x_i \partial x_j} + \sum_{i=1}^{n} b_i(\mathbf{x},\tau)\frac{\partial w}{\partial x_i} + c(\mathbf{x},\tau)w. \tag{1.22}$$

The operator $\frac{\partial}{\partial \tau} + \tilde{L}$ is called *parabolic* at the point $(\mathbf{x},\tau)$ if the matrix $A(\mathbf{x},\tau)$ is positive definite (see (1.9) for the definition).

If $A(\mathbf{x},\tau)$ is positive definite at all $(\mathbf{x},\tau) \in \overline{\Omega}\times[T_0,T_1]$, the operator is called parabolic in $\overline{\Omega}\times[T_0,T_1]$. If the matrix is in addition uniformly positive definite, i.e. if there exist constants $\lambda,\Lambda > 0$ such that

$$\lambda|\boldsymbol{\xi}|^2 \le \boldsymbol{\xi}^T A(\mathbf{x},\tau)\,\boldsymbol{\xi} \le \Lambda|\boldsymbol{\xi}|^2 \quad \text{for all } \boldsymbol{\xi}\in\mathbb{R}^n \quad \text{for all } (\mathbf{x},\tau)\in\overline{\Omega}\times[T_0,T_1] \tag{1.23}$$

where $|\boldsymbol{\xi}| = \left(\sum_{i=1}^n \xi_i^2\right)^{1/2}$ is the Euclidean norm of the vector $\boldsymbol{\xi}$, then the operator $\frac{\partial}{\partial \tau} + \tilde{L}$ is called *uniformly parabolic* in $\overline{\Omega}\times[T_0,T_1]$.

Proposition 1. *The operator $\frac{\partial}{\partial \tau} + L$ of the PDE* (1.18) *where*

$$Lw := -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_i\sigma_j\rho_{ij}\frac{\partial^2 w}{\partial x_i\partial x_j} + \sum_{i=1}^{n}\Big(\frac{\sigma_i^2}{2} - r\Big)\frac{\partial w}{\partial x_i} + rw \tag{1.24}$$

is uniformly parabolic in $\mathbb{R}^n\times[0,T]$ for any $T>0$.

Proof. We want to show (1.23) for the differential operator (1.24). Here, we have $a_{ij}(\mathbf{x},\tau) = \frac{1}{2}\sigma_i\sigma_j\rho_{ij}$, i.e.

$$A(\mathbf{x},\tau) = \frac{1}{2}\Sigma \quad \text{for all } (\mathbf{x},\tau)\in\mathbb{R}^n\times[0,T]. \tag{1.25}$$

As the covariance matrix Σ is a real symmetric matrix, it has n real eigenvalues $\lambda_1 \le \ldots \le \lambda_n$ and there exists an orthonormal basis of $\mathbb{R}^n$ consisting of corresponding eigenvectors $\{\mathbf{v}_1,\ldots,\mathbf{v}_n\}$. We can therefore write any vector $\boldsymbol{\xi}\in\mathbb{R}^n$ in the form $\boldsymbol{\xi} = \sum_{i=1}^n \tilde{\xi}_i\mathbf{v}_i$ and obtain

$$\boldsymbol{\xi}^T A(\mathbf{x},\tau)\,\boldsymbol{\xi} = \frac{1}{2}\Big(\sum_{i=1}^{n}\tilde{\xi}_i\mathbf{v}_i^T\Big)\Sigma\Big(\sum_{j=1}^{n}\tilde{\xi}_j\mathbf{v}_j\Big) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\tilde{\xi}_i\tilde{\xi}_j\underbrace{\mathbf{v}_i^T\lambda_j\mathbf{v}_j}_{=\lambda_j\delta_{ij}} = \frac{1}{2}\sum_{j=1}^{n}\tilde{\xi}_j{}^2\lambda_j, \tag{1.26}$$

and consequently,

$$\frac{1}{2}\lambda_1|\boldsymbol{\xi}|^2 \le \boldsymbol{\xi}^T A(\mathbf{x},\tau)\,\boldsymbol{\xi} \le \frac{1}{2}\lambda_n|\boldsymbol{\xi}|^2 \quad \text{for all } \boldsymbol{\xi}\in\mathbb{R}^n \tag{1.27}$$

uniformly for all $(\mathbf{x},\tau)\in\mathbb{R}^n\times[0,T]$ as the matrix does not depend on $(\mathbf{x},\tau)$. We further assumed Σ to be positive definite; this implies that all eigenvalues are positive. Hence, (1.23) is satisfied with $\lambda = \frac{1}{2}\lambda_1 > 0$, $\Lambda = \frac{1}{2}\lambda_n > 0$. □

For the original Black-Scholes PDE (1.10) in $[0,\infty)^n \times [0,T]$ (after the time transformation (1.15) in order to obtain the form (1.20)), in contrast, we have $a_{ij}(\mathbf{S},\tau) = \frac{1}{2}\sigma_i\sigma_j\rho_{ij}S_iS_j$. Hence, if $S_k = 0$ for some k, taking $\boldsymbol{\xi} = \mathbf{e}_k = (\delta_{ik})_{i=1}^n$ yields

$$\mathbf{e}_k{}^T A(\mathbf{S},\tau)\,\mathbf{e}_k = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_i\sigma_j\rho_{ij}S_iS_j\,\delta_{ik}\delta_{jk} = \frac{1}{2}\sigma_k^2S_k^2 = 0, \tag{1.28}$$

i.e. the matrix $A(\mathbf{S},\tau)$ is not positive definite in this case; the problem (1.10)–(1.11) is degenerate at the boundaries of the domain. We have seen that the log transform (1.16) eliminates this degeneracy. In addition, it has the effect that the PDE (1.18) has constant coefficients in contrast to (1.10). This is advantageous in the numerical solution of the problem, cf. [Brennan and Schwartz, 1978]. Therefore, we will henceforth consider the transformed problem (1.18)–(1.19) which is a uniformly parabolic problem in forward time in the whole of $\mathbb{R}^n$.

1.3.2 Solvability of the Black-Scholes problem

In this section, we will prove that the problem

$$\frac{\partial w}{\partial \tau} + Lw = 0, \qquad \mathbf{x}\in\mathbb{R}^n,\ \tau\in(0,T], \tag{1.29}$$

$$w(\mathbf{x},0) = V_0(\mathrm{e}^{x_1},\dots,\mathrm{e}^{x_n}), \qquad \mathbf{x}\in\mathbb{R}^n \tag{1.30}$$

with the operator L defined in (1.24) has a unique classical solution for every continuous pay-off function V_0 satisfying a certain bound. A problem of the form

$$\frac{\partial w}{\partial \tau} + \tilde{L}w = f(\mathbf{x},\tau), \qquad \mathbf{x}\in\mathbb{R}^n,\ \tau\in(0,T], \tag{1.31}$$

$$w(\mathbf{x},0) = w_0(\mathbf{x}), \qquad \mathbf{x}\in\mathbb{R}^n \tag{1.32}$$

is called a *Cauchy problem* in the strip $[0,T]$. We will use several results of [Friedman, 1964] in order to prove existence and uniqueness of a solution of the Cauchy problem (1.29)–(1.30). First, we define what we understand by a "classical" solution:

Definition 2. A function $w(\mathbf{x},\tau)$ defined in $\mathbb{R}^n\times[0,T]$ is called a *classical solution* of the Cauchy problem (1.31)–(1.32) if

1. w is continuous in $\mathbb{R}^n\times[0,T]$ and satisfies the prescribed initial condition $w(\mathbf{x},0) = w_0(\mathbf{x})$ at all $\mathbf{x}\in\mathbb{R}^n$,
2. w is continuously differentiable with respect to τ for all $(\mathbf{x},\tau)\in\mathbb{R}^n\times(0,T]$,
3. w is twice continuously differentiable with respect to $\mathbf{x}$ for all $(\mathbf{x},\tau)\in\mathbb{R}^n\times(0,T]$,
4. $\frac{\partial w}{\partial\tau} + \tilde{L}w = f(\mathbf{x},\tau)$ holds at each point $(\mathbf{x},\tau)\in\mathbb{R}^n\times(0,T]$.

Theorem 2. ([Friedman, 1964, Chapter 1, Theorem 12]).
Assume that

(*A*1) *The operator* $\frac{\partial}{\partial\tau} + \tilde{L}$ *is uniformly parabolic in* $\mathbb{R}^n\times[0,T]$.

(*A*2) *The matrix* $A^{-1}(\mathbf{x},\tau)$ *is uniformly positive definite, i.e.* (1.23) *holds for* A^{-1} *for some positive constants* $\bar{\lambda}$, $\bar{\Lambda}$. *(Note that* (*A*1) *implies that the matrix* $A(\mathbf{x},\tau)$ *is invertible.)*

(*A*3) $a_{ij}(\mathbf{x},\tau)$, $b_i(\mathbf{x},\tau)$, *and* $c(\mathbf{x},\tau)$ *are continuous and bounded functions in* $\mathbb{R}^n \times [0,T]$ *for all* $i,j = 1,\ldots,n$.

(*A*4) *For some Hölder exponent* $0 < \alpha < 1$, *the functions* $b_i(\mathbf{x},\tau)$ *and* $c(\mathbf{x},\tau)$ *are Hölder continuous in* $\mathbf{x} \in \mathbb{R}^n$ *uniformly with respect to* $\tau \in [0,T]$, *and the functions* $a_{ij}(\mathbf{x},\tau)$ *satisfy*

$$|a_{ij}(\mathbf{x}_1,\tau_1) - a_{ij}(\mathbf{x}_2,\tau_2)| \le C\left(|\mathbf{x}_1 - \mathbf{x}_2|^\alpha + |\tau_1 - \tau_2|^{\alpha/2}\right) \tag{1.33}$$

for some constant $C > 0$ *for all* $(\mathbf{x}_1,\tau_1),(\mathbf{x}_2,\tau_2) \in \mathbb{R}^n \times [0,T]$.

(*A*5) $f(\mathbf{x},\tau)$ *and* $w_0(\mathbf{x})$ *are continuous in* $\mathbb{R}^n \times [0,T]$ *and* $\mathbb{R}^n$, *respectively, and there is a constant* C *such that*

$$|f(\mathbf{x},\tau)| \le C e^{h|\mathbf{x}|^2}, \tag{1.34}$$

$$|w_0(\mathbf{x})| \le C e^{h|\mathbf{x}|^2} \qquad \text{in } \mathbb{R}^n \times [0,T] \ \text{ for some } 0 \le h < \frac{\bar{\lambda}}{4T} \tag{1.35}$$

with $\bar{\lambda}$ *defined in* (*A*2).

(*A*6) $f(\mathbf{x},\tau)$ *is locally Hölder continuous of exponent* α *in* $\mathbf{x} \in \mathbb{R}^n$ *uniformly with respect to* $\tau \in [0,T]$.

Under these conditions, there exists a classical solution w *of the Cauchy problem* (1.31)–(1.32), *and this solution satisfies*

$$|w(\mathbf{x},\tau)| \le A e^{k|\mathbf{x}|^2} \quad \text{for all } (\mathbf{x},\tau) \in \mathbb{R}^n \times [0,T] \tag{1.36}$$

with some constant k *depending only on* h, $\bar{\lambda}$, *and* T.

Remark. The proof of this theorem is based on the construction of a fundamental solution using the so-called parametrix method.

Theorem 3. ([Friedman, 1964, Chapter 1, Theorem 16]).
Assume that the differential operator satisfies the conditions (*A*1), (*A*3) *and* (*A*4), *and that in addition,*

(*A*7) *The functions* $\frac{\partial a_{ij}}{\partial x_k}$, $\frac{\partial^2 a_{ij}}{\partial x_k \partial x_\ell}$, *and* $\frac{\partial b_i}{\partial x_k}$ *are continuous and bounded in* $\mathbb{R}^n \times [0,T]$ *for all* i,j, $k,\ell = 1,\ldots,n$, *and they are Hölder continuous of exponent* α *in* $\mathbf{x} \in \mathbb{R}^n$ *uniformly with respect to* $\tau \in [0,T]$.

Then there exists at most one solution of the Cauchy problem (1.31)–(1.32) *satisfying the condition*

$$\int_0^T \int_{\mathbb{R}^n} |w(\mathbf{x},\tau)|\, e^{-k|\mathbf{x}|^2}\, d\mathbf{x}\, d\tau < \infty \tag{1.37}$$

for some $k > 0$.

Using Theorems 2 and 3, we can prove an existence and uniqueness result for our problem.

Corollary 4. *The Black-Scholes Cauchy problem* (1.29)–(1.30) *with*

$$a_{ij}(\mathbf{x},\tau)=\frac{1}{2}\sigma_i\sigma_j\rho_{ij},\quad b_i(\mathbf{x},\tau)=\frac{\sigma_i^2}{2}-r,\quad c(\mathbf{x},\tau)=r\quad \text{in } \mathbb{R}^n\times[0,T] \tag{1.38}$$

has a unique classical solution satisfying (1.36) *if the pay-off function* V_0 *is continuous in* $[0,\infty)^n$ *and satisfies*

$$|V_0(\mathbf{S})|\le c\big(1+\sum_{i=1}^n S_i\big) \tag{1.39}$$

in $[0,\infty)^n$ *with some constant* $c>0$.

Remark. The pay-off functions of all types of options we presented in Table 1.1 are continuous and satisfy a bound of the form (1.39), as, e.g., the pay-off of a put option is always bounded by the exercise price, $\min_i S_i$ is bounded by $\frac{1}{n}\sum_{i=1}^n S_i$, and $\max_i S_i\le\sum_{i=1}^n S_i$.

Proof of Corollary 4. We show that the conditions (*A*1)–(*A*7) are satisfied.

(*A*1) has already been shown in Proposition 1.

For an invertible matrix A with eigenvalues $\lambda_1,\dots,\lambda_n$, the eigenvalues of A^{-1} are given by $\frac{1}{\lambda_1},\dots,\frac{1}{\lambda_n}$. Thus, the same argument as in the proof of Proposition 1 shows that

$$\frac{2}{\lambda_n}|\boldsymbol{\xi}|^2\le\boldsymbol{\xi}^T A^{-1}(\mathbf{x},\tau)\,\boldsymbol{\xi}\le\frac{2}{\lambda_1}|\boldsymbol{\xi}|^2\quad\text{for all }\boldsymbol{\xi}\in\mathbb{R}^n \tag{1.40}$$

uniformly for all $(\mathbf{x},\tau)\in\mathbb{R}^n\times[0,T]$, i.e. (*A*2) is satisfied with $\bar\lambda=\frac{2}{\lambda_n}>0$ and $\bar\Lambda=\frac{2}{\lambda_1}>0$ where λ_1 and λ_n are the smallest and the largest eigenvalue of the covariance matrix Σ, respectively.

As for (*A*3), (*A*4) and (*A*7), the functions a_{ij}, b_i and c given in (1.38) are constant in $\mathbb{R}^n\times[0,T]$ and thus continuous, bounded, and uniformly Hölder continuous of any exponent α; the same holds for their derivatives which are zero.

Let us now examine the initial function $w_0(\mathbf{x})=V_0(\mathrm{e}^{x_1},\dots,\mathrm{e}^{x_n})$. As V_0 is assumed to be continuous in $[0,\infty)^n$, so is w_0 in $\mathbb{R}^n$. Furthermore, (*A*5) demands a bound of the form $|w_0(\mathbf{x})|\le C\mathrm{e}^{h|\mathbf{x}|^2}$ for some $h<\frac{\bar\lambda}{4T}=\frac{1}{2\lambda_n T}$. We will show that the condition (1.39) implies this. First note that

$$\begin{aligned}|V_0(\mathrm{e}^{x_1},\dots,\mathrm{e}^{x_n})|\le C\mathrm{e}^{h|\mathbf{x}|^2}\quad&\Leftrightarrow\quad|V_0(\mathbf{S})|\le C\mathrm{e}^{h\sum_{i=1}^n(\ln S_i)^2}\\ \text{for all }\mathbf{x}\in\mathbb{R}^n\quad&\qquad\text{for all }\mathbf{S}\in(0,\infty)^n.\end{aligned} \tag{1.41}$$

By assumption (1.39), we have $|V_0(\mathbf{S})|\le c\big(1+\sum_{i=1}^n S_i\big)$, i.e. we can prove the required bound if we can show that

$$g_h(\mathbf{S}):=\frac{c+c\sum_{i=1}^n S_i}{\mathrm{e}^{h\sum_{i=1}^n(\ln S_i)^2}} \tag{1.42}$$

is bounded for all $\mathbf{S}\in(0,\infty)^n$ for some sufficiently small h. We estimate

$$g_h(\mathbf{S})\le\underbrace{\frac{c}{\mathrm{e}^{h\sum_{i=1}^n(\ln S_i)^2}}}_{=:g_{0,h}(\mathbf{S})}+c\sum_{i=1}^n\underbrace{\frac{S_i}{\mathrm{e}^{h(\ln S_i)^2}}}_{=:\bar g_h(S_i)} \tag{1.43}$$

and immediately see that $g_{0,h}(\mathbf{S})$ is bounded by c, as $\mathrm{e}^{h\sum_{i=1}^n(\ln S_i)^2}\ge 1$ for any $h\ge 0$. As for the function $\bar g_h$, basic calculus shows that $\lim_{x\to0+}\bar g_h(x)=\lim_{x\to\infty}\bar g_h(x)=0$, that $\bar g_h(x)\ge 0$ for $x>0$,

and that $\bar{g}_h$ assumes its maximum at $x = \mathrm{e}^{1/(2h)}$. Thus, $\bar{g}_h$ is bounded by $\bar{g}_h(\mathrm{e}^{1/(2h)}) = \mathrm{e}^{1/(4h)}$. This holds for any $h > 0$, such that we can choose, e.g., $h^* := \frac{\bar{\lambda}}{8T}$ and thus prove (1.35) with $h = h^* < \frac{\bar{\lambda}}{4T}$ and $C = c(1 + n\mathrm{e}^{1/(4h^*)})$.

As $f(\mathbf{x}, \tau) = 0$ in our case, f is continuous, uniformly Hölder continuous of any exponent, and obviously satisfies the same bound as w_0 uniformly in τ such that all requirements of $(A5)$ and $(A6)$ are met.

Theorem 2 thus ensures the existence of a classical solution w. This solution moreover satisfies $|w(\mathbf{x}, \tau)| \leq A\mathrm{e}^{k|\mathbf{x}|^2}$ for some $k > 0$ and hence satisfies (1.37) with $k + \epsilon$, $\epsilon > 0$. By Theorem 3, w is the only solution with this property. □

Note that we have only shown uniqueness in terms of *solutions satisfying the constraint* (1.37), i.e. solutions which do not grow too fast at infinity. As we have restricted our analysis to pay-off functions which grow at most linearly (condition (1.39)), this condition seems to be economically reasonable and thus not to constitute a true constraint.

We have shown that there exists a unique classical solution $w(\mathbf{x}, \tau)$ to the transformed Black-Scholes problem (1.18)–(1.19) in $\mathbb{R}^n \times (0, T]$. We can transform this solution back to the $(\mathbf{S}, t)$ variables via

$$V(\mathbf{S}, t) := w\left(\ln S_1, \ldots, \ln S_n, T - t\right) \tag{1.44}$$

and thus obtain a unique solution to the original Black-Scholes problem (1.10)–(1.11) in the domain $(0, \infty)^n \times [0, T)$. This solution is as smooth as w in the interior of the domain and therefore satisfies the Black-Scholes PDE in the classical sense. However, we cannot infer that V can be continuously extended to $[0, \infty)^n \times [0, T]$, except for $t = T$: the back transform yields $V(\mathbf{S}, T) = V_0(\mathbf{S})$ for all $\mathbf{S} \in (0, \infty)^n$. The continuity of V_0 that we assumed in Corollary 4 then implies that $V(\mathbf{S}, T)$ can be continuously extended to $[0, \infty)^n$ and that the initial condition is thus in fact satisfied in $[0, \infty)^n$ as required by (1.11). On the whole of $[0, \infty)^n \times [0, T]$, however, we obtain a continuous function only if $\lim_{x_i \to -\infty} w(\mathbf{x}, \tau)$ exists for all $\mathbf{x} \in \mathbb{R}^n$, for all $\tau \in [0, T]$ and for all i and is continuous. We cannot infer this from our assumptions.

We can, however, derive more accurate bounds on the solution w of (1.29)–(1.30), using a positivity theorem for parabolic PDEs as done in [Kangro and Nicolaides, 2000]. We will make use of those estimates in Section 1.4.4.

Theorem 5. ([Friedman, 1964, Chapter 2, Theorem 9]).

Consider the Cauchy problem (1.31)–(1.32) *and assume that the operator* $\frac{\partial}{\partial \tau} + \tilde{L}$ *is parabolic with continuous coefficient functions* $a_{ij}(\mathbf{x}, \tau)$, $b_i(\mathbf{x}, \tau)$, *and* $c(\mathbf{x}, \tau)$ *in* $\mathbb{R}^n \times (0, T]$. *Furthermore, assume that the following bounds hold in* $\mathbb{R}^n \times (0, T]$ *for some* $M > 0$ *for all* $i, j = 1, \ldots, n$:

$$|a_{ij}(\mathbf{x}, \tau)| \leq M, \quad |b_i(\mathbf{x}, \tau)| \leq M\left(|\mathbf{x}| + 1\right), \quad |c(\mathbf{x}, \tau)| \leq M\left(|\mathbf{x}|^2 + 1\right). \tag{1.45}$$

Let w *be a continuous function in* $\mathbb{R}^n \times [0, T]$ *which is continuously differentiable with respect to* τ *and twice continuously differentiable with respect to* $\mathbf{x}$, *and let*

$$w(\mathbf{x}, \tau) \geq -Ae^{k|\mathbf{x}|^2} \tag{1.46}$$

hold in $\mathbb{R}^n \times [0, T]$ *for some positive constants* A *and* k. *If*

$$\frac{\partial w}{\partial \tau} + \tilde{L}w \geq 0 \quad \text{in } \mathbb{R}^n \times (0, T] \tag{1.47}$$

and

$$w(\mathbf{x}, 0) \geq 0 \quad \text{in } \mathbb{R}^n, \tag{1.48}$$

it follows that

$$w(\mathbf{x},\tau) \geq 0 \quad \textit{in } \mathbb{R}^n \times [0,T]. \tag{1.49}$$

Hence, this theorem allows to infer non-negativity of the solution from non-negativity of the initial data. If there are linear bounds for the pay-off function V_0, it can be applied to derive bounds for the option price at any point in time:

Corollary 6. *Under the assumptions of Corollary 4, assume that there are constants ν_i, κ_i, $i = 0, \dots, n$ such that*

$$\nu_0 + \sum_{i=1}^{n} \nu_i S_i \leq V_0(\mathbf{S}) \leq \kappa_0 + \sum_{i=1}^{n} \kappa_i S_i \tag{1.50}$$

for all $\mathbf{S} \in (0,\infty)^n$. Then, the solution w according to Corollary 4 satisfies

$$\nu_0 e^{-r\tau} + \sum_{i=1}^{n} \nu_i e^{x_i} \leq w(\mathbf{x},\tau) \leq \kappa_0 e^{-r\tau} + \sum_{i=1}^{n} \kappa_i e^{x_i} \tag{1.51}$$

for all $(\mathbf{x},\tau) \in \mathbb{R}^n \times [0,T]$.

Proof. We proceed by applying Theorem 5 to the functions

$$u_1(\mathbf{x},\tau) := w(\mathbf{x},\tau) - \big(\nu_0 \mathrm{e}^{-r\tau} + \sum_{i=1}^{n} \nu_i \mathrm{e}^{x_i}\big), \tag{1.52}$$

$$u_2(\mathbf{x},\tau) := \big(\kappa_0 \mathrm{e}^{-r\tau} + \sum_{i=1}^{n} \kappa_i \mathrm{e}^{x_i}\big) - w(\mathbf{x},\tau). \tag{1.53}$$

By Corollary 4, w is continuous and sufficiently smooth such that L can be applied to u_1 and u_2, and w satisfies $|w(\mathbf{x},\tau)| \leq A\mathrm{e}^{k|\mathbf{x}|^2}$, hence $w(\mathbf{x},\tau) \geq -A\mathrm{e}^{k|\mathbf{x}|^2}$. We can further estimate

$$\big|\nu_0 \mathrm{e}^{-r\tau} + \sum_{i=1}^{n} \nu_i \mathrm{e}^{x_i}\big| \leq |\nu_0| + \sum_{i=1}^{n} |\nu_i| \mathrm{e}^{\sqrt{n}\max\{1,|\mathbf{x}|^2\}} \quad \text{in } \mathbb{R}^n \times [0,T] \tag{1.54}$$

and thus see that u_1 satisfies (1.46) with suitable constants A and k. Analogously, (1.46) can be shown for u_2.

As we have shown earlier, the Black-Scholes differential operator is uniformly parabolic with continuous coefficients. As the coefficient functions are constant, they obviously satisfy (1.45) as well.

Furthermore, we have $\frac{\partial w}{\partial \tau} + Lw = 0$ and

$$\begin{aligned}\big(\frac{\partial}{\partial \tau} + L\big)\big(\nu_0 \mathrm{e}^{-r\tau} + \sum_{i=1}^{n} \nu_i \mathrm{e}^{x_i}\big) &= -r\nu_0 \mathrm{e}^{-r\tau} - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \sigma_i \sigma_j \rho_{ij}\, \delta_{ij}\, \nu_i \mathrm{e}^{x_i} + \sum_{i=1}^{n} \Big(\frac{\sigma_i^2}{2} - r\Big)\nu_i \mathrm{e}^{x_i} \\ &\quad + r\nu_0 \mathrm{e}^{-r\tau} + r\sum_{i=1}^{n} \nu_i \mathrm{e}^{x_i} = \sum_{i=1}^{n} \Big(-\frac{\sigma_i^2}{2} + \frac{\sigma_i^2}{2} - r + r\Big)\nu_i \mathrm{e}^{x_i} = 0, \end{aligned} \tag{1.55}$$

hence $\frac{\partial u_1}{\partial \tau} + Lu_1 = 0$, and analogously, $\frac{\partial u_2}{\partial \tau} + Lu_2 = 0$. Finally, $u_1(\mathbf{x},0) \geq 0$ and $u_2(\mathbf{x},0) \geq 0$ hold in $\mathbb{R}^n$ by the assumption (1.50) transformed to the $(\mathbf{x},\tau)$ variables. Thus, Theorem 5 yields (1.51). □

1.4 Numerical solution of the Black-Scholes problem

We have proved in the previous section that the transformed Black-Scholes problem (1.29)–(1.30) possesses a classical solution; however, an analytical solution is only available in some special cases, see Section 5.1. In the general case, the problem cannot be solved analytically but only numerically.

1.4.1 Truncation of the domain

We will use a finite-difference method (see Chapter 2) to calculate a numerical solution. However, such a method can only be applied on a *bounded* domain whereas the spatial domain of the problem (1.29)–(1.30) is the whole of $\mathbb{R}^n$. We therefore need to *localise* the PDE to a bounded computational domain, i.e to *truncate* the original domain. For some $R > 0$, define

$$\Omega_R := (-R, R)^n. \tag{1.56}$$

In the numerical solution, we will consider the truncated version of problem (1.29)–(1.30) in the spatial domain Ω_R,

$$\frac{\partial \tilde{w}}{\partial \tau} + L\tilde{w} = 0, \qquad \mathbf{x} \in \Omega_R,\ \tau \in (0, T], \tag{1.57}$$

$$\tilde{w}(\mathbf{x}, 0) = V_0(\mathrm{e}^{x_1}, \ldots, \mathrm{e}^{x_n}) \qquad \text{for all } \mathbf{x} \in \overline{\Omega_R}, \tag{1.58}$$

$$\tilde{w}(\mathbf{x}, \tau) = \tilde{g}(\mathbf{x}, \tau) \qquad \text{for all } (\mathbf{x}, \tau) \in \partial\Omega_R \times (0, T]. \tag{1.59}$$

The truncation of the domain introduces the new "artificial" boundary $\partial\Omega_R \times (0, T]$ where we have to impose suitable boundary conditions (1.59) which we do not yet specify here. Due to this localisation, the function $\tilde{w}$ will in general be different from the exact solution w of (1.29)–(1.30) restricted to $\overline{\Omega_R} \times [0, T]$; it can be hoped, however, that $\tilde{w}$ is a good approximation of w if the domain Ω_R is "large enough". In Section 1.4.3, we will analyse the localisation error in detail and subsequently discuss the choice of the boundary function $\tilde{g}$ in Section 1.4.4.

Problem (1.57)–(1.59) is of the *convection-diffusion* type (cf., e.g, [Günther and Jüngel, 2003], [Seydel, 2004]), as we see writing (1.57) in the form

$$\frac{\partial w}{\partial \tau} - \operatorname{div}(A\nabla w) + \mathbf{b} \cdot \nabla w + rw = 0 \tag{1.60}$$

with the constant matrix $A = \frac{1}{2}\Sigma \in \mathbb{R}^{n\times n}_{\mathrm{sym}}$ and the constant vector $\mathbf{b} \in \mathbb{R}^n$ defined by $b_i = \left(\frac{\sigma_i^2}{2} - r\right)$. The term $\operatorname{div}(A\nabla w)$ then physically corresponds to a diffusion, $\mathbf{b} \cdot \nabla w$ to a convection, and rw to a reaction. The size of the convection term compared to the diffusion term is crucial when using a finite-difference method to numerically solve such a problem. A dominant convection term can cause stability problems and spurious oscillations in the numerical solution, whereas the diffusion term has a smoothing property. In our case, problems may arise if there is an asset whose volatility is much lower than the square root of twice the interest rate. This is not the case in most realistic settings (cf. [Persson and von Sydow, 2007, (17)]). We will come across a constraint involving this parameter in Chapter 2, Proposition 13. Note that in the special case $\sigma_i^2 = 2r$ for all i, the convection term vanishes, and we have a pure diffusion-reaction problem.

1.4.2 Solvability of the truncated Black-Scholes problem

Before we proceed to numerically solve the problem (1.57)–(1.59), we make sure that this problem actually has a unique solution. As in the case of the Cauchy problem, this can be done with the aid of results by [Friedman, 1964].

Consider an initial-boundary value problem of the form

$$\frac{\partial w}{\partial \tau} + \tilde{L}w = f(\mathbf{x}, \tau) \qquad \text{in } \Omega \times (0, T], \tag{1.61}$$

$$w = w_0 \qquad \text{in } \overline{\Omega} \times \{\tau = 0\}, \tag{1.62}$$

$$w = g \qquad \text{on } \partial\Omega \times (0, T] \tag{1.63}$$

where Ω is a bounded domain in $\mathbb{R}^n$. By a *classical solution* of (1.61)–(1.63), we understand a function w which is continuous in $\overline{\Omega} \times [0, T]$, continuously differentiable with respect to τ and twice continuously differentiable with respect to $\mathbf{x}$ in $\Omega \times (0, T]$, and which satisfies the equations (1.61)–(1.63) point-wise. Denote $B := \Omega \times \{\tau = 0\}$ and $S := \partial\Omega \times (0, T]$. $\overline{B}$ is the part of the boundary where initial data need to be prescribed, and S is the manifold where boundary data have to be prescribed. A continuous solution can only be expected if the combination of the initial function w_0 and the boundary function g forms a *continuous* function on $\overline{B} \cup S$, i.e. if the transition at $\partial\Omega \times \{\tau = 0\}$ is continuous. We will consider this compatibility condition in our choice of boundary conditions for the problem (1.57)–(1.59) in Section 1.4.4.

The following existence, uniqueness, and regularity results can be shown.

Theorem 7. (adapted from [Friedman, 1964, Chapter 3, Corollary 1 to Theorem 9 and Corollary 2 to Theorem 11]).

Let Ω be a bounded domain in $\mathbb{R}^n$ and consider the initial-boundary value problem

$$\frac{\partial w}{\partial \tau} + \tilde{L}w = f \qquad \textit{in } \Omega \times (0, T], \tag{1.64}$$

$$w = \psi \qquad \textit{on } \overline{B} \cup S \tag{1.65}$$

where (1.65) represents a combined initial-boundary condition. Let $\frac{\partial}{\partial \tau} + \tilde{L}$ be a parabolic operator with constant coefficients in $\overline{\Omega} \times [0, T]$, and let f be infinitely differentiable in $\Omega \times (0, T]$.

Furthermore, assume that S has the outside strong sphere property*, i.e. for every $P = (\bar{\mathbf{x}}, \bar{\tau}) \in S$ there exist a closed ball K with centre $(\mathbf{x}_0, \tau_0)$ and a constant $\varepsilon > 0$ independent of P such that*

$$K \cap \left(\overline{\Omega} \times [0, T]\right) = \{P\} \quad \textit{and} \tag{1.66}$$

$$|\mathbf{x}_0 - \mathbf{x}| \geq c(P) > 0 \quad \textit{for all } (\mathbf{x}, \tau) \in \overline{\Omega} \times [0, T] \textit{ with } |\tau - \bar{\tau}| < \varepsilon. \tag{1.67}$$

Then for every function ψ which is continuous on $\overline{B} \cup S$, there exists a unique classical solution w of (1.64)–(1.65). This solution is infinitely differentiable in $\Omega \times (0, T]$.

Corollary 8. *The truncated Black-Scholes problem (1.57)–(1.59) posed in $\Omega_R \times (0, T]$ for any $R > 0$ has a unique classical solution $\tilde{w}$ provided that V_0 and $\tilde{g}$ are continuous with a continuous transition at $\partial\Omega_R \times \{\tau = 0\}$. $\tilde{w}$ is then infinitely differentiable in $\Omega_R \times (0, T]$.*

Proof. It only remains to show that $S = \partial\Omega_R \times (0, T]$ has the outside strong sphere property. For $P = (\bar{\mathbf{x}}, \bar{\tau}) \in S$ with $\bar{x}_i = R$ ($\bar{x}_i = -R$), choose any $\rho > 0$ and define K as the closed ball with radius ρ and centre $(\bar{\mathbf{x}} + \rho\,\mathbf{e}_i, \bar{\tau})$ ($(\bar{\mathbf{x}} - \rho\,\mathbf{e}_i, \bar{\tau})$). Then, clearly, (1.66) and (1.67) are satisfied with $c(P) = \rho$ and ε arbitrary.

Theorem 7 thus proves the result. □

1.4.3 Estimation of the localisation error

We remind of the definitions that $w(\mathbf{x},\tau)$ denotes the solution of the Black-Scholes problem (1.29)–(1.30) on $\mathbb{R}^n \times [0,T]$, and that $\tilde{w}(\mathbf{x},\tau)$ denotes the solution of the truncated problem (1.57)–(1.59) on $[-R,R]^n \times [0,T]$. We will now show that $\tilde{w}$ is a good approximation of w provided that the computational domain is suitably large; more specifically, the localisation error $|w-\tilde{w}|$ at each fixed point $(\mathbf{x},\tau) \in \Omega_R \times (0,T]$ can be estimated in terms of the maximum error on the boundary of the domain, i.e. the maximum error in the boundary conditions $\tilde{g}$ prescribed for $\tilde{w}$. Based on this estimate, we will address the issue of choosing suitable boundary conditions in the subsequent section. For certain types of options and boundary conditions, we will finally be able to show that the localisation error tends to zero point-wise in $\Omega_R \times (0,T]$ as R tends to infinity, albeit at a rate of at best $O(\frac{1}{R})$. For single-asset options, we will present an alternative approach at the end of this section that shows that the localisation error in fact decays exponentially in R. However, this result refers to a different norm, and it cannot be generalised to higher dimensions.

To estimate the localisation error in the n-dimensional case, we closely follow the ideas of [Kangro and Nicolaides, 2000], yet with some differences: Kangro and Nicolaides examine the *untransformed* Black-Scholes problem on $[0,\infty)^n \times [0,T)$ and thus only have to truncate the domain at large values of $\mathbf{S}$, introducing an artificial *far field boundary*. Hence, they derive a bound for the localisation error in terms of the far field boundary error. They stress that they do not recommend to transform the problem to the whole of $\mathbb{R}^n$, as they regard the additional artificial boundary which then has to be introduced for small asset price values as an unnecessary complication. In our analysis, however, we always examine the *transformed* problem on the whole of $\mathbb{R}^n$ because we wish to take advantage of the numerical convenience of a constant coefficient problem. Consequently, we have to truncate the computational domain at both large *and* small values of the asset prices; Ω_R with $\mathbf{x} \in (-R,R)^n$ corresponds to $\mathbf{S} \in \left(\mathrm{e}^{-R},\mathrm{e}^{R}\right)^n =: \left(\frac{1}{S_{\max}},S_{\max}\right)^n$. From the numerical point of view, the truncation near the zero boundary is not a big problem, for asset prices near zero are seldom of practical interest, and the existence of the solution on the zero boundary is questionable anyway as we have seen in Section 1.3.2. However, we do have to consider the additional artificial boundary in the analysis of the localisation error and therefore cannot proceed exactly like [Kangro and Nicolaides, 2000]. We will show how to modify their arguments to apply them to our case and still derive a convergence result, though a much weaker one than the result by [Kangro and Nicolaides, 2000].

The central argument is the following weak maximum principle for initial-boundary value problems.

Theorem 9. ([Friedman, 1964, Chapter 2, Theorem 6]).
Let Ω be a bounded domain in $\mathbb{R}^n$. For any point $(\mathbf{x}_0,\tau_0) \in \Omega \times (0,T)$, define

$$S\big((\mathbf{x}_0,\tau_0)\big) := \{(\mathbf{x},\tau) \in \Omega \times (0,T) : (\mathbf{x},\tau) \textit{ can be connected to } (\mathbf{x}_0,\tau_0) \textit{ by a simple continuous curve in } \Omega \times (0,T) \textit{ along which the } \tau \textit{ coordinate is nondecreasing.}\} \quad (1.68)$$

Assume that the operator $\frac{\partial}{\partial\tau} + \tilde{L}$ is parabolic and has continuous coefficients in $\Omega \times (0,T)$. Furthermore, assume that

$$c(\mathbf{x},\tau) \geq 0 \quad \textit{in } \Omega \times (0,T). \quad (1.69)$$

Let u be a continuous function in $\overline{\Omega} \times [0,T]$ which is continuously differentiable with respect to τ and twice continuously differentiable with respect to $\mathbf{x}$ and for which

$$\frac{\partial u}{\partial \tau} + \tilde{L}u \geq 0 \quad \text{in } \Omega \times (0,T). \tag{1.70}$$

Then for each $P := (\mathbf{x}_0, \tau_0)$ in $\Omega \times (0,T)$, there holds that if u has a negative minimum in $\overline{S(P)}$, this minimum is (also) obtained at some point in $\overline{S(P)} \setminus S(P)$.

For Ω being a domain, i.e. an open connected set in $\mathbb{R}^n$ and hence a path-wise connected set, it is easy to see that

$$S\big((\mathbf{x}_0, \tau_0)\big) = \{(\mathbf{x}, \tau) \in \Omega \times (0,T) : \tau \leq \tau_0\} = \Omega \times (0, \tau_0], \tag{1.71}$$

and hence that

$$\overline{S\big((\mathbf{x}_0, \tau_0)\big)} \setminus S\big((\mathbf{x}_0, \tau_0)\big) = \overline{\Omega} \times \{\tau = 0\} \ \cup \ \partial\Omega \times (0, \tau_0). \tag{1.72}$$

Using this, the contraposition of Theorem 9 yields the following very useful version of the maximum principle.

Theorem 10. *Under the assumptions of Theorem 9, if*

$$\frac{\partial u}{\partial \tau} + \tilde{L}u \geq 0 \quad \text{in } \Omega \times (0, \tau_0), \tag{1.73}$$

$$u(\mathbf{x}, 0) \geq 0 \quad \text{for all } \mathbf{x} \in \overline{\Omega} \tag{1.74}$$

$$\text{and} \qquad u(\mathbf{x}, \tau) \geq 0 \quad \text{on } \partial\Omega \times (0, \tau_0] \tag{1.75}$$

for some $\tau_0 > 0$, then

$$u(\mathbf{x}, \tau) \geq 0 \quad \text{in } \overline{\Omega} \times [0, \tau_0]. \tag{1.76}$$

Like [Kangro and Nicolaides, 2000], we now use Theorem 10 to obtain an a priori estimate for the localisation error in the interior of the domain in terms of its maximum value on the boundary.

Lemma 11. *Let z be any continuous function in $\overline{\Omega_R} \times [0,T]$ which is continuously differentiable with respect to τ and twice continuously differentiable with respect to $\mathbf{x}$ and which satisfies*

$$z(\mathbf{x}, \tau) \geq 0 \quad \text{in } \overline{\Omega_R} \times [0,T], \tag{1.77}$$

$$z(\mathbf{x}, \tau) > 0 \quad \text{on } \partial\Omega_R \times [0,T] \tag{1.78}$$

$$\text{and} \qquad \frac{\partial z}{\partial \tau} + Lz \geq 0 \quad \text{in } \Omega_R \times (0,T]. \tag{1.79}$$

Denote $\Gamma := \partial\Omega_R$. Then the localisation error for the Black-Scholes problem can be estimated like

$$|\tilde{w}(\mathbf{x}, \tau) - w(\mathbf{x}, \tau)| \leq \sup_{(\mathbf{x}_0, \tau_0) \in \ \Gamma \times (0, \tau]} \frac{|\tilde{w}(\mathbf{x}_0, \tau_0) - w(\mathbf{x}_0, \tau_0)|}{z(\mathbf{x}_0, \tau_0)} \ z(\mathbf{x}, \tau) \qquad \text{for all } (\mathbf{x}, \tau) \in \Omega_R \times (0,T]. \tag{1.80}$$

Proof. For any fixed $\tau^* \in (0,T]$, define

$$f_{\tau^*}(\mathbf{x},\tau) := \underbrace{\sup_{(\mathbf{x}_0,\tau_0)\in\ \Gamma\times(0,\tau^*]} \frac{|\tilde{w}(\mathbf{x}_0,\tau_0) - w(\mathbf{x}_0,\tau_0)|}{z(\mathbf{x}_0,\tau_0)}}_{=:\ C_{\tau^*}} \ z(\mathbf{x},\tau). \tag{1.81}$$

Here, C_{τ^*} is a well-defined non-negative constant; assumption (1.78) ensures that z is bounded away from zero on $\Gamma \times (0,\tau^*]$. We will now apply Theorem 10 to the functions f_+ and f_- defined by

$$f_\pm(\mathbf{x},\tau) := f_{\tau^*}(\mathbf{x},\tau) \pm (\tilde{w}(\mathbf{x},\tau) - w(\mathbf{x},\tau)) \quad \text{in } \overline{\Omega_R} \times [0,\tau^*]. \tag{1.82}$$

The Black-Scholes differential operator satisfies all requirements of Theorem 10, in particular, $c(\mathbf{x},\tau) = r \geq 0$. Furthermore,

$$\begin{aligned}\frac{\partial f_\pm}{\partial \tau} + L f_\pm &= \frac{\partial f_{\tau^*}}{\partial \tau} + L f_{\tau^*} \pm \Big(\underbrace{\frac{\partial \tilde{w}}{\partial \tau} + L\tilde{w}}_{\substack{=0 \text{ in } \Omega_R\times(0,T] \\ \text{by (1.57)}}} - \underbrace{(\frac{\partial w}{\partial \tau} + Lw)}_{\substack{=0 \text{ in } \mathbb{R}^n\times(0,T] \\ \text{by (1.29)}}} \Big) = \underbrace{C_{\tau^*}}_{\geq 0} \Big(\underbrace{\frac{\partial z}{\partial \tau} + Lz}_{\substack{\geq 0 \text{ in } \Omega_R\times(0,T] \\ \text{by (1.79)}}} \Big) \\ &\geq 0 \quad \text{in } \Omega_R \times (0,\tau^*]\end{aligned} \tag{1.83}$$

for any $\tau^* \in (0,T]$, i.e. (1.73) is satisfied for f_+ and f_-. We further have

$$\begin{aligned} f_\pm(\mathbf{x},0) = f_{\tau^*}(\mathbf{x},0) \pm \Big(\underbrace{\tilde{w}(\mathbf{x},0)}_{\substack{=V_0(e^{x_1},\ldots,e^{x_n}) \\ \text{in } \overline{\Omega_R} \text{ by (1.58)}}} - \underbrace{w(\mathbf{x},0)}_{\substack{=V_0(e^{x_1},\ldots,e^{x_n}) \\ \text{in } \mathbb{R}^n \text{ by (1.30)}}} \Big) = \underbrace{C_{\tau^*}}_{\geq 0} \underbrace{z(\mathbf{x},0)}_{\substack{\geq 0 \text{ in } \overline{\Omega_R} \\ \text{by (1.77)}}} \\ \geq 0 \quad \text{in } \overline{\Omega_R} \end{aligned} \tag{1.84}$$

such that (1.74) is satisfied. Finally, in order to show the requirement (1.75), note that for all $(\mathbf{x},\tau) \in \Gamma \times (0,\tau^*]$, there holds

$$C_{\tau^*} = \sup_{(\mathbf{x}_0,\tau_0)\in\ \Gamma\times(0,\tau^*]} \frac{|\tilde{w}(\mathbf{x}_0,\tau_0) - w(\mathbf{x}_0,\tau_0)|}{z(\mathbf{x}_0,\tau_0)} \geq \frac{|\tilde{w}(\mathbf{x},\tau) - w(\mathbf{x},\tau)|}{z(\mathbf{x},\tau)} \tag{1.85}$$

so that we obtain

$$\begin{aligned} f_\pm(\mathbf{x},\tau) &= C_{\tau^*}\, z(\mathbf{x},\tau) \pm (\tilde{w}(\mathbf{x},\tau) - w(\mathbf{x},\tau)) \\ &\geq \frac{|\tilde{w}(\mathbf{x},\tau) - w(\mathbf{x},\tau)|}{z(\mathbf{x},\tau)}\ z(\mathbf{x},\tau) \pm (\tilde{w}(\mathbf{x},\tau) - w(\mathbf{x},\tau)) \\ &= \big|\tilde{w}(\mathbf{x},\tau) - w(\mathbf{x},\tau)\big| \pm \big(\tilde{w}(\mathbf{x},\tau) - w(\mathbf{x},\tau)\big) \quad \geq 0 \end{aligned} \tag{1.86}$$

for all $(\mathbf{x},\tau) \in \Gamma \times (0,\tau^*]$.

Thus, Theorem 10 yields $f_\pm \geq 0$ in $\overline{\Omega_R} \times [0,\tau^*]$, i.e.

$$|\tilde{w}(\mathbf{x},\tau) - w(\mathbf{x},\tau)| \leq C_{\tau^*}\, z(\mathbf{x},\tau) \quad \text{for all } (\mathbf{x},\tau) \in \overline{\Omega_R} \times [0,\tau^*], \quad \text{for all } \tau^* \in (0,T]. \tag{1.87}$$

Setting $\tau = \tau^*$ in (1.87), we obtain (1.80). □

Setting $z(\mathbf{x},\tau)=1$ in Lemma 11, we immediately obtain as a first result that

$$|\tilde{w}(\mathbf{x},\tau)-w(\mathbf{x},\tau)| \le \sup_{(\mathbf{x}_0,\tau_0)\in\ \Gamma\times(0,\tau]} |\tilde{w}(\mathbf{x}_0,\tau_0)-w(\mathbf{x}_0,\tau_0)| \quad \text{for all } (\mathbf{x},\tau)\in\Omega_R\times(0,T], \tag{1.88}$$

i.e. that the localisation error within the domain is always bounded by the maximum error on the boundary up to time τ (the initial data being assumed to be exact). It is a general result that the solution of a linear homogeneous parabolic problem with a continuous parabolic operator with $c\ge 0$ is bounded by the maximum of the initial and boundary data (see, e.g., [Friedman, 1964, Chapter 2]). With another choice of z, however, we can even show the stronger result that the localisation error decreases with the size of the domain: we now set

$$z(\mathbf{x},\tau) := \sum_{i=1}^n x_i^2 + B_R\,\tau \quad \text{with } B_R := \sum_{i=1}^n \sigma_i^2 + R\sum_{i=1}^n \big|\sigma_i^2-2r\big| > 0. \tag{1.89}$$

This function clearly satisfies (1.77). On $\Gamma\times[0,\tau]$, the smallest value is assumed at the points $(\pm R\mathbf{e}_j,0)$ where $z\left(\pm R\mathbf{e}_j,0\right)=R^2>0$, such that (1.78) is satisfied as well. Finally,

$$\begin{aligned}
\frac{\partial z}{\partial\tau}+Lz &= B_R - \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \sigma_i\sigma_j\rho_{ij}\,2\delta_{ij} + \sum_{i=1}^n\Big(\frac{\sigma_i^2}{2}-r\Big)2x_i + rz \\
&= B_R - \sum_{i=1}^n \sigma_i^2 + \sum_{i=1}^n \underbrace{\left(\sigma_i^2-2r\right)x_i}_{\le\left|\sigma_i^2-2r\right||x_i|} + \underbrace{rz}_{\ge 0} \\
&\ge B_R - \sum_{i=1}^n \sigma_i^2 - \sum_{i=1}^n \big|\sigma_i^2-2r\big|R \quad = 0,
\end{aligned} \tag{1.90}$$

i.e. (1.79) is satisfied. Lemma 11 now gives us

$$\begin{aligned}
|\tilde{w}(\mathbf{x},\tau)-w(\mathbf{x},\tau)| &\le \sup_{(\mathbf{x}_0,\tau_0)\in\ \Gamma\times(0,\tau]} \frac{|\tilde{w}(\mathbf{x}_0,\tau_0)-w(\mathbf{x}_0,\tau_0)|}{z(\mathbf{x}_0,\tau_0)}\; z(\mathbf{x},\tau) \\
&\le \frac{\sup\limits_{(\mathbf{x}_0,\tau_0)\in\ \Gamma\times(0,\tau]} |\tilde{w}(\mathbf{x}_0,\tau_0)-w(\mathbf{x}_0,\tau_0)|}{\inf\limits_{(\mathbf{x}_0,\tau_0)\in\ \Gamma\times(0,\tau]} z(\mathbf{x}_0,\tau_0)}\; z(\mathbf{x},\tau) \\
&= \|\tilde{w}-w\|_{L^\infty(\Gamma\times(0,\tau])}\,\frac{\sum_{i=1}^n x_i^2 + B_R\tau}{R^2} \qquad \text{for all } (\mathbf{x},\tau)\in\Omega_R\times(0,T].
\end{aligned} \tag{1.91}$$

For each fixed interior point $(\mathbf{x}^*,\tau^*)$, the factor $\frac{\sum_{i=1}^n (x_i^*)^2+B_R\tau^*}{R^2}$ is a constant of order $O(1/R)$ (or $O(1/R^2)$ in the special case that $\sigma_i^2=2r$ for all i, i.e. in the case of a pure diffusion problem). This proves the following convergence result:

Theorem 12. *If the maximum boundary error $\|\tilde{w}-w\|_{L^\infty(\Gamma\times(0,\tau])}$ can be bounded for any $R>0$ by a bound of order $O(R^\alpha)$ with $\alpha<1$ for all $\tau\in(0,T]$, then the localisation error $|\tilde{w}-w|$ of the Black-Scholes problem truncated to $\Omega_R\times(0,T]$ tends to zero point-wise for each $(\mathbf{x},\tau)\in\mathbb{R}^n\times(0,T]$ as R tends to infinity. The convergence order is at least $O(1/(R^{1-\alpha}))$.*

It is an important question in practical numerical finance how large the computational domain must be in order to keep the localisation error small (compared to other discretisation errors due to the numerical method, modelling errors, rounding errors etc.) [Wilmott et al., 1995] and [Seydel, 2004] state that the computational domain should be chosen "suitably large" without giving a more precise quantification; [Günther and Jüngel, 2003] refer to [Kangro and Nicolaides, 2000] for the proof that the localisation error is small if the computational domain is sufficiently large. [Kangro and Nicolaides, 2000] and [Persson and von Sydow, 2007] mention the general "rule of thumb" for numerical computations,

$$S_{\max} \approx 3 \text{ to } 4 \text{ times the exercise price of the option,} \tag{1.92}$$

where the exercise price usually corresponds to a discontinuity in the first derivative of the pay-off function (cf. Table 1.1). Additionally, however, [Kangro and Nicolaides, 2000] show how their estimate of the localisation error can be used to explicitly determine the size of the computational domain necessary to ensure a desired accuracy within some domain of interest. They thus find in numerical experiments that the above mentioned rule of thumb is sufficient if not too conservative.

A similar estimate is possible based on our localisation error bound (1.91): if one is interested in the option price for asset prices lying in some fixed domain $\Omega_{\tilde{R}}$, say (e.g. a neighbourhood of the current asset price), the necessary size of the computational domain such that the localisation error within $\Omega_{\tilde{R}} \times [0,T]$ is (uniformly) lower than some tolerance ε is obtained by solving the inequality

$$\|\tilde{w} - w\|_{L^\infty(\Gamma\times(0,T])} \frac{n\tilde{R}^2 + \left(\sum_{i=1}^n \sigma_i^2 + R\sum_{i=1}^n \left|\sigma_i^2 - 2r\right|\right) T}{R^2} \leq \varepsilon \tag{1.93}$$

for R. This estimate requires a bound for $\|\tilde{w} - w\|_{L^\infty(\Gamma\times(0,T])}$ which must be of lower order than $O(R)$, i.e. $O(\ln S_{\max})$; otherwise, decay of the localisation error cannot be inferred from (1.91). Even if a suitable bound for $\|\tilde{w} - w\|_{L^\infty(\Gamma\times(0,T])}$ is available, however, the resulting domain size may be much larger than the rule of thumb suggests; this is due to the low decay rate of $R^{\alpha-1}$, i.e. $(\ln S_{\max})^{\alpha-1}$, obtained in Theorem 12. This is a much weaker result than the point-wise decay rate of (at best) $\mathrm{e}^{-O(R^2)}$ found by [Kangro and Nicolaides, 2000], where convergence of the localisation error is still guaranteed if $\|\tilde{w} - w\|_{L^\infty(\Gamma\times(0,T])}$ grows like $S_{\max} = \mathrm{e}^R$. However, as already mentioned above, this result refers to a different truncation strategy.

In the one-dimensional case, an alternative approach proves that the localisation error in our problem indeed decays like $\mathrm{e}^{-O(R)}$. We can proceed as in [Hilber et al., 2004, Theorem 3.7], where the localisation error for the PDE pricing a European single-asset option under stochastic volatility is analysed. Due to the assumption of stochastic volatility, an additional spatial variable is involved in the PDE analysed there; however, the approach carries over to the Black-Scholes problem (1.18)–(1.19) with $n = 1$ in a straightforward way.

[Hilber et al., 2004] consider the differential problem in a variational setting in weighted Sobolev spaces that enforce decay of the solution at infinity; this assumption is justified by considering the excess to pay-off,

$$\bar{w}(x,\tau) := w(x,\tau) - \mathrm{e}^{-r\tau} V_0(\mathrm{e}^{x+r\tau}), \tag{1.94}$$

which tends to zero as $|x| \to \infty$ when the boundary behaviour (1.6) is assumed, instead of the option price w. The new unknown function $\bar{w}$ satisfies the PDE

$$L\bar{w}\,(x,\tau) = \frac{\sigma^2}{2}\,\mathrm{e}^{2x+r\tau}\,\frac{d^2V_0}{dS^2}\,(\mathrm{e}^{x+r\tau}), \qquad x \in \mathbb{R},\ \tau \in (0,T], \tag{1.95}$$

subject to the initial condition

$$\bar{w}(x,0) = 0 \qquad \text{for all } x \in \mathbb{R} \tag{1.96}$$

and "boundary conditions",

$$\lim_{|x|\to\infty} \bar{w}(x,\tau) = 0 \qquad \text{for all } \tau \in (0,T]. \tag{1.97}$$

The operator L in (1.95) is the transformed Black-Scholes operator defined in (1.24) with $n = 1$; in contrast to the PDE for the option price w, the initial and boundary conditions are homogeneous now, but the right-hand side is inhomogeneous. The right-hand side function

$$f(x,\tau) := \frac{\sigma^2}{2}\, \mathrm{e}^{2x+r\tau}\, \frac{d^2V_0}{dS^2}(\mathrm{e}^{x+r\tau}) \tag{1.98}$$

involves the second derivative of the option's pay-off V_0. As Table 1.1 demonstrates, typical pay-off function are continuous but not continuously differentiable. For instance, the pay-off (1.4) of a call option can be written as the product of a smooth function and the Heaviside step function,

$$V_0(S) = \max\{S-E,\,0\} = (S-E)\cdot H_E(S) \qquad \text{where } H_E(S) = \begin{cases} 0, & S \le E, \\ 1, & S > E. \end{cases} \tag{1.99}$$

The second derivative of H_E exists only in the distributional sense and is given by the delta distribution defined by

$$\langle \delta_E,\, \phi\rangle = \phi(E) \quad \text{for } \phi \in C_0^\infty(\mathbb{R}). \tag{1.100}$$

The right-hand side function (1.98) thus has to be understood as a distribution, and the strong formulation (1.95)–(1.97) of the problem makes no sense. However, the problem can be solved in an appropriate variational setting. In our case, we can use the evolution triple $H^1_\varphi(\mathbb{R}) \hookrightarrow L^2_\varphi(\mathbb{R}) \hookrightarrow (H^1_\varphi(\mathbb{R}))^*$, where

$$L^2_\varphi(\mathbb{R}) := \{v : v\varphi \in L^2(\mathbb{R})\} \quad \text{with } \varphi \in W^{1,\infty}_{\mathrm{loc}}(\mathbb{R}),\ \varphi(x) \ge 0 \tag{1.101}$$

and $H^1_\varphi(\mathbb{R})$ denotes the space of functions in $L^2_\varphi(\mathbb{R})$ which are weakly differentiable in $L^2_\varphi(\mathbb{R})$, equipped with the norm

$$\|v\|^2_{H^1_\varphi(\mathbb{R})} := \|v\|^2_{L^2_\varphi(\mathbb{R})} + \Big\|\frac{dv}{dx}\Big\|^2_{L^2_\varphi(\mathbb{R})} = \|v\varphi\|^2_{L^2(\mathbb{R})} + \Big\|\frac{dv}{dx}\varphi\Big\|^2_{L^2(\mathbb{R})}. \tag{1.102}$$

The weight function φ imposes the homogeneous "boundary conditions"; it turns out that

$$\varphi(x) = \mathrm{e}^{\mu|x|} \tag{1.103}$$

with $\mu > 0$ is an appropriate choice for our problem. The variational formulation of (1.95)–(1.97) then is,

Find $\bar{w} \in L^2(0,T;H^1_\varphi(\mathbb{R})) \cap H^1(0,T;(H^1_\varphi(\mathbb{R}))^*)$ such that

$$\Big(\frac{d}{d\tau}\,\bar{w}(\,\cdot\,,\tau),\, v\Big)_{L^2_\varphi(\mathbb{R})} + a^\varphi\big(\bar{w}(\,\cdot\,,\tau),\, v\big) = \langle f(\,\cdot\,,\tau),\, v\rangle_{(H^1_\varphi(\mathbb{R}))^*\times H^1_\varphi(\mathbb{R})} \qquad \text{for all } v \in H^1_\varphi(\mathbb{R})$$
$$\text{for all } \tau \in (0,T], \tag{1.104}$$

$$\bar{w}(\,\cdot\,,0) = 0, \tag{1.105}$$

with the bilinear form $a^\varphi : H^1_\varphi(\mathbb{R}) \times H^1_\varphi(\mathbb{R}) \to \mathbb{R}$ defined by

$$a^\varphi(u,v) := \int_{\mathbb{R}} \Big(\frac{\sigma^2}{2} \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} + \sigma^2 \frac{\partial u}{\partial x}\, v \frac{\varphi'(x)}{\varphi(x)} + \Big(\frac{\sigma^2}{2} - r \Big) \frac{\partial u}{\partial x}\, v + r\, uv \Big) \varphi(x)^2 \, dx. \tag{1.106}$$

By [Hilber et al., 2004, Proposition 3.1], this problem has a unique solution if

$$f \in L^2(0,T; (H^1_\varphi(\mathbb{R}))^*) \tag{1.107}$$

and if the bilinear form $a^\varphi(\,\cdot\,,\,\cdot\,)$ is continuous and satisfies the Gårding inequality

$$a^\varphi(v,v) \geq c_1 \|v\|^2_{H^1_\varphi(\mathbb{R})} - c_2 \|v\|^2_{L^2_\varphi(\mathbb{R})} \quad \text{for some constants } c_1 > 0,\ c_2 > 0. \tag{1.108}$$

For our problem, it can be shown that there exists $\bar\mu > 0$ such that the conditions on a^φ are satisfied for every weight function of the form (1.103) with $\mu \leq \bar\mu$. $f(\,\cdot\,,\tau)$ from (1.98) represents a continuous functional on $H^1_\varphi(\mathbb{R})$ for each τ if $\frac{d^2 V_0}{dS^2}(\mathrm{e}^{x+r\tau})$ does. Without explicitly stating it, [Hilber et al., 2004] use that the delta distribution satisfies the requirements. Hence, for typical pay-off functions, the problem (1.104)–(1.105) has a unique weak solution $\bar w \in L^2(0,T; H^1_\varphi(\mathbb{R})) \cap H^1(0,T; (H^1_\varphi(\mathbb{R}))^*)$. If a classical solution according to Corollary 4 exists as well, they must coincide, hence the weak solution of (1.104)–(1.105) is in fact a classical solution.

The truncated Black-Scholes problem (1.57)–(1.59) in the domain $\Omega_R \times (0,T] = (-R,R) \times (0,T]$ has the variational formulation

Find $\tilde{\bar w} \in L^2(0,T; H^1_0(\Omega_R)) \cap H^1(0,T; (H^1_0(\Omega_R))^*)$ such that

$$\Big(\frac{d}{d\tau}\, \tilde{\bar w}(\,\cdot\,,\tau),\, v \Big)_{L^2(\Omega_R)} + a_R\big(\tilde{\bar w}(\,\cdot\,,\tau),\, v\big) = \langle f(\,\cdot\,,\tau),\, v \rangle_{(H^1_0(\Omega_R))^* \times H^1_0(\Omega_R)} \quad \text{for all } v \in H^1_0(\Omega_R)$$

$$\text{for all } \tau \in (0,T], \tag{1.109}$$

$$\tilde{\bar w}(\,\cdot\,,0) = 0 \qquad \text{in } \Omega_R, \tag{1.110}$$

where the bilinear form a_R is the restriction of a^1 as given in (1.108) to $H^1_0(\Omega_R) \times H^1_0(\Omega_R)$. Here, the homogeneous boundary conditions are included in the definition of spaces.

Using the same arguments as in [Hilber et al., 2004, Theorem 3.7], one can now prove that there exists a constant C independent of R and of τ such that

$$\|(\tilde{\bar w} - \bar w)(\,\cdot\,,\tau)\|^2_{L^2(\Omega_{R/2})} + \int_0^\tau \|(\tilde{\bar w} - \bar w)(\,\cdot\,,s)\|^2_{H^1(\Omega_{R/2})}\, ds \leq C \mathrm{e}^{-\bar\mu R} \quad \text{for all } \tau \in [0,T], \tag{1.111}$$

i.e. the localisation error within the domain $\Omega_{R/2}$ is bounded by a function that decays exponentially with the domain size R. As $H^1(\Omega_{R/2})$ is continuously embedded in $C^0([-\frac{R}{2}, \frac{R}{2}])$ by the *Sobolev embedding theorem*, (1.111) implies that

$$\Big(\int_0^T \sup_{x \in [-\frac{R}{2}, \frac{R}{2}]} \big| (\tilde{\bar w} - \bar w)(x,s) \big|^2 \, ds \Big)^{\frac{1}{2}} \leq C \mathrm{e}^{-\frac{\bar\mu}{2} R}. \tag{1.112}$$

This estimate is much sharper than (1.91), albeit with respect to the weaker $L^2(0,T; C^0([-\frac{R}{2}, \frac{R}{2}]))$-norm rather than the $C^0([-\frac{R}{2}, \frac{R}{2}] \times [0,T])$-norm.

In any case, the proof does not carry over to the multi-dimensional case: for an n-asset option, the excess to pay-off satisfies the PDE

$$L\bar{w}\,(\mathbf{x},\tau) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_i\sigma_j\rho_{ij}\,\mathrm{e}^{x_i+x_j+r\tau}\,\frac{\partial^2 V_0}{\partial S_i\partial S_j}\,(\mathrm{e}^{x_1+r\tau},\ldots,\mathrm{e}^{x_n+r\tau}), \qquad \mathbf{x}\in\mathbb{R}^n,\ \tau\in(0,T], \tag{1.113}$$

with homogeneous initial and boundary conditions. For the existence proof of the variational formulation of the problem, we need the right-hand side function to be in $L^2(0,T;(H^1_\varphi(\mathbb{R}))^*)$ (see (1.107)). However, the delta distributions typically appearing in the second derivatives of option pay-offs are *not* continuous functionals on $H^1_0(\Omega_R)$ if $n>1$.

1.4.4 Possible choices of boundary conditions

As the error on the artificial boundary Γ, $\|\tilde{w}-w\|_{L^\infty(\Gamma\times(0,\tau])} = \|\tilde{g}-w\|_{L^\infty(\Gamma\times(0,\tau])}$, appears in our estimate (1.91) of the localisation error in the whole computational domain, it is obviously preferable to prescribe a boundary function $\tilde{g}$ in (1.59) which is a good approximation of the true solution w on Γ; however, as we have already seen in Section 1.2.1, the limiting behaviour of w is not always obvious. Conditions on the derivatives of $\tilde{g}$ such as (1.14) are not appropriate for the desired estimate, we need Dirichlet boundary conditions instead. [Kangro and Nicolaides, 2000] use

$$\tilde{g}_1(\mathbf{x},\tau) := V_0(\mathrm{e}^{x_1},\ldots,\mathrm{e}^{x_n}), \quad \mathbf{x}\in\Gamma,\ \tau\in(0,T] \tag{1.114}$$

corresponding to (1.12). Based upon (1.13), an alternative choice is

$$\tilde{g}_2(\mathbf{x},\tau) := \mathrm{e}^{-r\tau}\,V_0\big(\mathrm{e}^{x_1+r\tau},\ldots,\mathrm{e}^{x_n+r\tau}\big), \quad \mathbf{x}\in\Gamma,\ \tau\in(0,T]; \tag{1.115}$$

this is assumed in the one-dimensional variational model leading to the estimate (1.112). As $(\mathbf{x},\tau)\to(\bar{\mathbf{x}},0)$, we obviously have $\tilde{g}_i(\mathbf{x},\tau)\to V_0\big(\mathrm{e}^{\bar{x}_1},\ldots,\mathrm{e}^{\bar{x}_n}\big)$ for $i=1,2$, i.e. $\tilde{g}_1$ and $\tilde{g}_2$ are compatible with the initial condition. Let us now examine what we can say about the error $\|\tilde{g}_i-w\|_{L^\infty(\Gamma\times(0,\tau])}$ for several option types. We will make use of Corollary 6 in order to estimate the solution w.

Example 1. Consider a European basket call with exercise price $E\ge 0$. The corresponding pay-off function $\max\big\{\sum_{i=1}^n c_iS_i - E,\,0\big\}$ can be estimated like

$$V_0(\mathbf{S})\ge 0 \quad\text{and}\quad \sum_{i=1}^{n} c_iS_i - E \;\le\; V_0(\mathbf{S}) \;\le\; \sum_{i=1}^{n} c_iS_i, \tag{1.116}$$

so we obtain by Corollary 6

$$\underbrace{\max\Big\{\sum_{i=1}^{n} c_i\mathrm{e}^{x_i} - E\mathrm{e}^{-r\tau},\,0\Big\}}_{=\tilde{g}_2(\mathbf{x},\tau)} \le w(\mathbf{x},\tau) \le \sum_{i=1}^{n} c_i\mathrm{e}^{x_i} \tag{1.117}$$

and consequently,

$$w(\mathbf{x},\tau)-\tilde{g}_1(\mathbf{x},\tau) \le \sum_{i=1}^{n} c_i\mathrm{e}^{x_i} - \Big(\sum_{i=1}^{n} c_i\mathrm{e}^{x_i} - E\Big) = E, \tag{1.118}$$

$$0\le\; w(\mathbf{x},\tau)-\tilde{g}_2(\mathbf{x},\tau) \le \sum_{i=1}^{n} c_i\mathrm{e}^{x_i} - \Big(\sum_{i=1}^{n} c_i\mathrm{e}^{x_i} - E\mathrm{e}^{-r\tau}\Big) = E\mathrm{e}^{-r\tau}. \tag{1.119}$$

Although we obtain a more accurate point-wise bound for $w - \tilde{g}_2$ than for $w - \tilde{g}_1$, the resulting bound on the maximum error in $\Gamma \times (0, \tau]$ is

$$\|\tilde{g}_i - w\|_{L^\infty(\Gamma\times(0,\tau])} \leq E \tag{1.120}$$

in both cases. Hence, for a basket call, we find a bound independent of R which implies convergence of the localisation error like $1/R$.

Example 2. For any type of put option, the pay-off function is bounded by the exercise price (E or $\max_i E_i$), so we a obtain similar result as in the case of basket calls.

Example 3. For a call on the minimum of n assets, analogous estimates as in Example 1 yield $\|\tilde{g}_i - w\|_{L^\infty(\Gamma\times(0,\tau])} \leq \max_i E_i$. For a call on the maximum of n assets, however, linear estimates like in Corollary 6 only yield a bound of order e^R so that we cannot infer convergence in this case.

These examples show that $\tilde{g}_1$ and $\tilde{g}_2$ are appropriate choices of boundary functions for a number of option types. We will examine in numerical experiments in Chapter 5 which is the better choice.

Chapter 2

Finite-difference discretisation of the model problem

2.1 The computational grid

We will use a finite-difference method to approximate the solution of the truncated transformed Black-Scholes problem (1.57)–(1.59), i.e. we discretise the domain $\Omega_R \times (0,T]$ of the problem, approximate the PDE (1.57) at the grid points of the discrete mesh using finite differences instead of derivatives, and solve the resulting systems of equations in order to obtain an approximation of the solution $\tilde{w}(\mathbf{x},\tau)$ at the grid points.

We first have to define meshes in space and time. As for the time interval $[0,T]$, we choose a suitable number K defining the time step $k := \frac{T}{K} > 0$ and thus introduce the equidistant time grid

$$\Omega_k^\tau := \{\tau_0, \ldots, \tau_K\} \quad \text{with } \tau_\ell := \ell k,\ \ell = 0, \ldots, K \tag{2.1}$$

where $\tau_0 = 0$ and $\tau_K = T$.

As for the truncated spatial domain Ω_R, we choose a suitably large number $M = 2^m, m \geq 0$, and hence obtain on $\overline{\Omega_R}$ the equidistant spatial grid of mesh width $h = \frac{R}{M}$,

$$\Omega_{R,h}^{\mathbf{x}} := \{\mathbf{x}_{\boldsymbol{\alpha}} : \boldsymbol{\alpha} \in \mathbb{Z}^n,\ -M \leq \alpha_i \leq M,\ i = 1, \ldots, n\} \quad \text{with } \mathbf{x}_{\boldsymbol{\alpha}} := h\boldsymbol{\alpha} = (h\alpha_1, \ldots, h\alpha_n). \tag{2.2}$$

The mesh $\Omega_{R,h}^{\mathbf{x}}$ consists of a total of $(2M+1)^n$ grid points. Each of them is characterised by a multiindex $\boldsymbol{\alpha}$ which we will sometimes simply call its index. Grid points with at least one component α_i of their index being M or $-M$ lie on the boundary of the computational domain where we prescribe Dirichlet boundary values. The number of interior, i.e. "free" grid points is $(2M-1)^n$.

It will be important to identify "neighbouring" grid points. The neighbours in the i-th direction of a grid point $\mathbf{x}_{\boldsymbol{\alpha}}$ are those grid points with the smallest distance from $\mathbf{x}_{\boldsymbol{\alpha}}$ in the x_i direction. Clearly, every interior grid point $\mathbf{x}_{\boldsymbol{\alpha}}$ has two neighbours in the i-th direction with indices $\boldsymbol{\alpha} - \mathbf{e}_i$ and $\boldsymbol{\alpha} + \mathbf{e}_i$, respectively. The "two-step neighbours" reached from $\mathbf{x}_{\boldsymbol{\alpha}}$ by one step in the i-th direction and one in the j-th direction, with $i \neq j$, are the four grid points with indices $\boldsymbol{\alpha} \pm \mathbf{e}_i \pm \mathbf{e}_j$.

2.2 Finite-difference discretisation

Our goal is to approximate the solution $\tilde{w}(\mathbf{x},\tau)$ of problem (1.57)–(1.59) by a set of discrete vectors $\mathbf{W}^\ell \in \mathbb{R}^{(2M+1)^n}$, $\ell = 0,\dots,K$ representing approximations of the values of the function $\tilde{w}$ at the grid points of the spatial mesh $\Omega^{\mathbf{x}}_{R,h}$ at each time step τ_ℓ of Ω^τ_k. To simplify notation, we write $\tilde{w}(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) =: \tilde{w}^\ell_{\boldsymbol{\alpha}}$ and denote by $W^\ell_{\boldsymbol{\alpha}}$ the corresponding component of the approximation vector $\mathbf{W}^\ell$. Furthermore, the vector of all $\tilde{w}^\ell_{\boldsymbol{\alpha}}$ is denoted by $\tilde{\mathbf{w}}^\ell$.

The basic idea of the finite-difference method is to replace the derivatives in the PDE (1.57) by finite differences based on the grid points. Different types of difference quotients (forward, backward or central) can be used, yielding a variety of possible finite-difference schemes.

For a sufficiently smooth function $w : \overline{\Omega_R} \times [0,T] \to \mathbb{R}$, Taylor series expansion yields

$$w(\mathbf{x},\tau \pm k) = w(\mathbf{x},\tau) \pm k\frac{\partial w}{\partial \tau}(\mathbf{x},\tau) + \frac{k^2}{2!}\frac{\partial^2 w}{\partial \tau^2}(\mathbf{x},\tau) + O(k^3), \tag{2.3}$$

$$w(\mathbf{x} \pm h\mathbf{e}_i,\tau) = w(\mathbf{x},\tau) \pm h\frac{\partial w}{\partial x_i}(\mathbf{x},\tau) + \frac{h^2}{2!}\frac{\partial^2 w}{\partial x_i^2}(\mathbf{x},\tau) \pm \frac{h^3}{3!}\frac{\partial^3 w}{\partial x_i^3}(\mathbf{x},\tau) + O(h^4), \tag{2.4}$$

$$\begin{aligned} w(\mathbf{x} + h_i\mathbf{e}_i + h_j\mathbf{e}_j,\tau) &= w(\mathbf{x},\tau) + h_i\frac{\partial w}{\partial x_i}(\mathbf{x},\tau) + h_j\frac{\partial w}{\partial x_j}(\mathbf{x},\tau) + \frac{h_i^2}{2!}\frac{\partial^2 w}{\partial x_i^2}(\mathbf{x},\tau) \\ &+2\,\frac{h_i h_j}{2!}\frac{\partial^2 w}{\partial x_i \partial x_j}(\mathbf{x},\tau) + \frac{h_j^2}{2!}\frac{\partial^2 w}{\partial x_j^2}(\mathbf{x},\tau) + \frac{h_i^3}{3!}\frac{\partial^3 w}{\partial x_i^3}(\mathbf{x},\tau) + 3\,\frac{h_i^2 h_j}{3!}\frac{\partial^3 w}{\partial x_i^2 \partial x_j}(\mathbf{x},\tau) \\ &+3\,\frac{h_i h_j^2}{3!}\frac{\partial^3 w}{\partial x_i \partial x_j^2}(\mathbf{x},\tau) + \frac{h_j^3}{3!}\frac{\partial^3 w}{\partial x_j^3}(\mathbf{x},\tau) + O(h^4). \end{aligned} \tag{2.5}$$

If $\frac{\partial^3 w}{\partial \tau^3}$ and all $\mathbf{x}$-partial derivatives of the fourth order exist and are uniformly bounded in $\overline{\Omega_R}\times[0,T]$, the higher order terms in these formulae can be uniformly bounded for all $(\mathbf{x},\tau) \in \overline{\Omega_R} \times [0,T]$ and all $i,j \in \{1,\dots,n\}$.

The above formulae imply the finite-difference approximations for the τ-partial derivative,

$$\begin{aligned} \frac{\partial w}{\partial \tau}(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) &= \frac{1}{k}\Big(w(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell + k) - w(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) + O(k^2)\Big) \\ &= \frac{w^{\ell+1}_{\boldsymbol{\alpha}} - w^\ell_{\boldsymbol{\alpha}}}{k} + O(k) \quad \text{(forward difference)} \end{aligned} \tag{2.6}$$

or

$$\begin{aligned} \frac{\partial w}{\partial \tau}(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) &= \frac{1}{k}\Big(w(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) - w(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell - k) + O(k^2)\Big) \\ &= \frac{w^\ell_{\boldsymbol{\alpha}} - w^{\ell-1}_{\boldsymbol{\alpha}}}{k} + O(k) \quad \text{(backward difference)}, \end{aligned} \tag{2.7}$$

and for the first and second $\mathbf{x}$-partial derivatives,

$$\begin{aligned} \frac{\partial w}{\partial x_i}(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) &= \frac{1}{2h}\Big(w(\mathbf{x}_{\boldsymbol{\alpha}+\mathbf{e}_i},\tau_\ell) - w(\mathbf{x}_{\boldsymbol{\alpha}-\mathbf{e}_i},\tau_\ell) + O(h^3)\Big) \\ &= \frac{w^\ell_{\boldsymbol{\alpha}+\mathbf{e}_i} - w^\ell_{\boldsymbol{\alpha}-\mathbf{e}_i}}{2h} + O(h^2) \quad \text{(central difference)}, \end{aligned} \tag{2.8}$$

$$\frac{\partial^2 w}{\partial x_i^2}(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) = \frac{1}{h^2}\Big(w(\mathbf{x}_{\boldsymbol{\alpha}+\mathbf{e}_i},\tau_\ell) - 2w(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) + w(\mathbf{x}_{\boldsymbol{\alpha}-\mathbf{e}_i},\tau_\ell) + O(h^4)\Big)$$
$$= \frac{w^\ell_{\boldsymbol{\alpha}+\mathbf{e}_i} - 2w^\ell_{\boldsymbol{\alpha}} + w^\ell_{\boldsymbol{\alpha}-\mathbf{e}_i}}{h^2} + O(h^2) \quad \text{(central difference)}, \tag{2.9}$$

$$\frac{\partial^2 w}{\partial x_i \partial x_j}(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) = \frac{1}{4h^2}\Big(w\left(\mathbf{x}_{\boldsymbol{\alpha}+\mathbf{e}_i+\mathbf{e}_j},\tau_\ell\right) - w\left(\mathbf{x}_{\boldsymbol{\alpha}+\mathbf{e}_i-\mathbf{e}_j},\tau_\ell\right)$$
$$- w\left(\mathbf{x}_{\boldsymbol{\alpha}-\mathbf{e}_i+\mathbf{e}_j},\tau_\ell\right) + w\left(\mathbf{x}_{\boldsymbol{\alpha}-\mathbf{e}_i-\mathbf{e}_j},\tau_\ell\right) + O(h^4)\Big)$$
$$= \frac{w^\ell_{\boldsymbol{\alpha}+\mathbf{e}_i+\mathbf{e}_j} - w^\ell_{\boldsymbol{\alpha}+\mathbf{e}_i-\mathbf{e}_j} - w^\ell_{\boldsymbol{\alpha}-\mathbf{e}_i+\mathbf{e}_j} + w^\ell_{\boldsymbol{\alpha}-\mathbf{e}_i-\mathbf{e}_j}}{4h^2} + O(h^2) \quad \text{(central difference)}. \tag{2.10}$$

Hence, we obtain the central difference approximation of the differential operator L on the grid $\Omega^{\mathbf{x}}_{R,h} \times \Omega^\tau_k$,

$$L\tilde{w}(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) = -\frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \sigma_i\sigma_j\rho_{ij}\frac{\partial^2\tilde{w}}{\partial x_i\partial x_j}(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) + \sum_{i=1}^n\Big(\frac{\sigma_i^2}{2} - r\Big)\frac{\partial\tilde{w}}{\partial x_i}(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) + r\tilde{w}(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell)$$
$$\approx -\frac{1}{2}\sum_{i\neq j}\sigma_i\sigma_j\rho_{ij}\ \frac{\tilde{w}^\ell_{\boldsymbol{\alpha}+\mathbf{e}_i+\mathbf{e}_j} - \tilde{w}^\ell_{\boldsymbol{\alpha}+\mathbf{e}_i-\mathbf{e}_j} - \tilde{w}^\ell_{\boldsymbol{\alpha}-\mathbf{e}_i+\mathbf{e}_j} + \tilde{w}^\ell_{\boldsymbol{\alpha}-\mathbf{e}_i-\mathbf{e}_j}}{4h^2}$$
$$-\frac{1}{2}\sum_{i=1}^n\sigma_i^2\ \frac{\tilde{w}^\ell_{\boldsymbol{\alpha}+\mathbf{e}_i} - 2\tilde{w}^\ell_{\boldsymbol{\alpha}} + \tilde{w}^\ell_{\boldsymbol{\alpha}-\mathbf{e}_i}}{h^2} + \sum_{i=1}^n\Big(\frac{\sigma_i^2}{2} - r\Big)\frac{\tilde{w}^\ell_{\boldsymbol{\alpha}+\mathbf{e}_i} - \tilde{w}^\ell_{\boldsymbol{\alpha}-\mathbf{e}_i}}{2h} + r\tilde{w}^\ell_{\boldsymbol{\alpha}} \tag{2.11}$$

or, rearranging terms and observing the symmetry with respect to i and j,

$$L\tilde{w}(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell) \approx \sum_{i<j}\frac{\sigma_i\sigma_j\rho_{ij}}{4h^2}\left(-\tilde{w}^\ell_{\boldsymbol{\alpha}+\mathbf{e}_i+\mathbf{e}_j} + \tilde{w}^\ell_{\boldsymbol{\alpha}+\mathbf{e}_i-\mathbf{e}_j} + \tilde{w}^\ell_{\boldsymbol{\alpha}-\mathbf{e}_i+\mathbf{e}_j} - \tilde{w}^\ell_{\boldsymbol{\alpha}-\mathbf{e}_i-\mathbf{e}_j}\right)$$
$$+\sum_{i=1}^n\Big(-\frac{\sigma_i^2}{2h^2} + \frac{\sigma_i^2}{4h} - \frac{r}{2h}\Big)\tilde{w}^\ell_{\boldsymbol{\alpha}+\mathbf{e}_i} + \sum_{i=1}^n\Big(-\frac{\sigma_i^2}{2h^2} - \frac{\sigma_i^2}{4h} + \frac{r}{2h}\Big)\tilde{w}^\ell_{\boldsymbol{\alpha}-\mathbf{e}_i}$$
$$+\Big(r + \sum_{i=1}^n\frac{\sigma_i^2}{h^2}\Big)\tilde{w}^\ell_{\boldsymbol{\alpha}} \tag{2.12}$$

at all *interior* grid points $\mathbf{x}_{\boldsymbol{\alpha}}$. This discretisation can be compactly written in the form

$$L\tilde{w}\big|_{\Omega^{\mathbf{x}}_{R,h}\times\{\tau_\ell\}} \approx A_h\tilde{\mathbf{w}}^\ell \tag{2.13}$$

with the "space discretisation matrix" $A_h \in \mathbb{R}^{(2M+1)^n\times(2M+1)^n}$ which is independent of τ. If we characterise the matrix entries by the multiindices associated with the mesh $\Omega^{\mathbf{x}}_{R,h}$, the matrix A_h

is given by

$$(A_h)_{\boldsymbol{\alpha},\boldsymbol{\beta}} = \begin{cases} r + \sum_{i=1}^{n} \frac{\sigma_i^2}{h^2} & \text{if } \boldsymbol{\beta} = \boldsymbol{\alpha} \\ & \quad \text{and } \mathbf{x}_{\boldsymbol{\alpha}} \text{ is an interior grid point,} \\ -\frac{\sigma_i^2}{2h^2} + \frac{\sigma_i^2}{4h} - \frac{r}{2h} & \text{if } \boldsymbol{\beta} = \boldsymbol{\alpha} + \mathbf{e}_i,\ i = 1, \ldots, n, \\ & \quad \text{and } \mathbf{x}_{\boldsymbol{\alpha}} \text{ is an interior grid point,} \\ -\frac{\sigma_i^2}{2h^2} - \frac{\sigma_i^2}{4h} + \frac{r}{2h} & \text{if } \boldsymbol{\beta} = \boldsymbol{\alpha} - \mathbf{e}_i,\ i = 1, \ldots, n, \\ & \quad \text{and } \mathbf{x}_{\boldsymbol{\alpha}} \text{ is an interior grid point,} \\ -\frac{\sigma_i \sigma_j \rho_{ij}}{4h^2} & \text{if } \boldsymbol{\beta} = \boldsymbol{\alpha} + \mathbf{e}_i + \mathbf{e}_j \text{ or } \boldsymbol{\beta} = \boldsymbol{\alpha} - \mathbf{e}_i - \mathbf{e}_j, \\ & \quad i \neq j, \text{ and } \mathbf{x}_{\boldsymbol{\alpha}} \text{ is an interior grid point,} \\ \frac{\sigma_i \sigma_j \rho_{ij}}{4h^2} & \text{if } \boldsymbol{\beta} = \boldsymbol{\alpha} + \mathbf{e}_i - \mathbf{e}_j \text{ or } \boldsymbol{\beta} = \boldsymbol{\alpha} - \mathbf{e}_i + \mathbf{e}_j, \\ & \quad i \neq j, \text{ and } \mathbf{x}_{\boldsymbol{\alpha}} \text{ is an interior grid point,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.14}$$

Due to the local character of the finite-difference discretisation, the matrix is *sparse* with at most $2n^2 + 1$ non-zero entries per row. We stress that the matrix is *not symmetric* except for the special case that the convection component in the PDE vanishes and that all assets are completely uncorrelated, i.e. in the case that $\frac{\sigma_i^2}{2} = r$ for all i and $\rho_{ij} = 0$ for all $i \neq j$.

Note that the matrix A_h has to include columns corresponding to boundary grid points in order to provide all neighbours of interior grid points. At the boundary of the spatial domain, however, the differential operator L is actually not defined. Therefore, we cannot interpret $\left(A_h \tilde{\mathbf{w}}^\ell\right)_{\boldsymbol{\alpha}}$ as an approximation of $L\tilde{w}\left(\mathbf{x}_{\boldsymbol{\alpha}}, \tau_\ell\right)$ if $\mathbf{x}_{\boldsymbol{\alpha}}$ is a boundary grid point. Instead, the rows of A_h corresponding to boundary grid points may be assigned arbitrary values. In (2.14), we define those rows to contain exclusively zeros, such that $\left(A_h \tilde{\mathbf{w}}^\ell\right)|_{\partial\Omega^{\mathbf{x}}_{R,h}} = \mathbf{0}$. We will later see that this choice is convenient for our numerical solution scheme. Independently of the components of A_h in the rows corresponding to boundary grid points, (2.8)–(2.10) show that

$$L\tilde{w}\big|_{\mathrm{int}\left(\Omega^{\mathbf{x}}_{R,h}\right)\times\{\tau_\ell\}} = A_h \tilde{\mathbf{w}}^\ell\big|_{\mathrm{int}\left(\Omega^{\mathbf{x}}_{R,h}\right)} + O(h^2) \tag{2.15}$$

provided that the fourth order $\mathbf{x}$-partial derivatives of $\tilde{w}$ are uniformly bounded, i.e. the finite-difference approximation of L by the matrix A_h then is of second order in h (cf., e.g., [Smith, 1978, Chapter 1]).

In contrast, the one-sided finite-difference approximations (2.6) and (2.7) of the τ-partial derivative are only of first order in k; nevertheless, they yield satisfactory numerical schemes, whereas central differences with respect to τ are useless in practice as the resulting numerical schemes are inherently unstable according to [Wilmott et al., 1995, Chapter 8] (see Section 2.3 for the definition of stability). In the following sections, we will briefly introduce a family of frequently used finite-difference schemes and discuss their properties when applied to our problem (1.57)–(1.59).

2.2.1 The θ-method

The so-called θ-method is presented in almost any reference on the numerical solution of parabolic PDEs, e.g., [Smith, 1978], [Morton and Mayers, 1994], [Günther and Jüngel, 2003], [Seydel, 2004],

or [Wilmott et al., 1995]. Consider the initial-boundary value problem

$$\frac{\partial w}{\partial \tau} + \tilde{L}w = f(\mathbf{x},\tau), \qquad \mathbf{x} \in \Omega,\ \tau \in (0,T], \tag{2.16}$$

$$w(\mathbf{x},0) = w_0(\mathbf{x}) \qquad \text{for all } \mathbf{x} \in \overline{\Omega}, \tag{2.17}$$

$$w(\mathbf{x},\tau) = g(\mathbf{x},\tau) \qquad \text{for all } (\mathbf{x},\tau) \in \partial\Omega \times (0,T], \tag{2.18}$$

where Ω is a bounded open region in $\mathbb{R}^n$, $\tilde{L}$ is a second order differential operator in the space variable $\mathbf{x}$ without explicit dependence on τ, and f, w_0 and g are given functions. We will restrict our considerations to the case where $\tilde{L}$ is linear and has constant coefficients as in our model problem; however, this is not essential for the application of the θ-method. Taking $\Omega = \Omega_R$ and the grid $\Omega^{\mathbf{x}}_{R,h} \times \Omega^{\tau}_{k}$ as defined in (2.1)–(2.2), the PDE (2.16) may be discretised at the interior grid point $(\mathbf{x}_{\boldsymbol{\alpha}}, \tau_\ell)$ using a forward difference (2.6) with respect to τ, and a discretisation matrix $\tilde{A}_h$ for $\tilde{L}$ based on central differences as in (2.11)–(2.15), yielding

$$\frac{w^{\ell+1}_{\boldsymbol{\alpha}} - w^{\ell}_{\boldsymbol{\alpha}}}{k} + \big(\tilde{A}_h \mathbf{w}^{\ell}\big)_{\boldsymbol{\alpha}} = f(\mathbf{x}_{\boldsymbol{\alpha}}, \tau_\ell) + O(k) + O(h^2)$$
$$\text{for all } \boldsymbol{\alpha} \text{ corresponding to interior grid points, } \ell = 0,\ldots,K-1, \tag{2.19}$$

where the $O(k)$ and $O(h^2)$ terms can be understood uniformly bounded in $\overline{\Omega} \times [0,T]$ provided that w is sufficiently smooth.

Alternatively, one might use a backward difference (2.7) at the time step $\tau_{\ell+1}$. This results in

$$\frac{w^{\ell+1}_{\boldsymbol{\alpha}} - w^{\ell}_{\boldsymbol{\alpha}}}{k} + \big(\tilde{A}_h \mathbf{w}^{\ell+1}\big)_{\boldsymbol{\alpha}} = f(\mathbf{x}_{\boldsymbol{\alpha}}, \tau_{\ell+1}) + O(k) + O(h^2)$$
$$\text{for all } \boldsymbol{\alpha} \text{ corresponding to interior grid points, } \ell = 0,\ldots,K-1. \tag{2.20}$$

The idea of the θ-method is to take a convex combination of these two equations. With $\theta \in [0,1]$, we obtain for sufficiently smooth solutions w,

$$\frac{w^{\ell+1}_{\boldsymbol{\alpha}} - w^{\ell}_{\boldsymbol{\alpha}}}{k} + (1-\theta)\big(\tilde{A}_h \mathbf{w}^{\ell}\big)_{\boldsymbol{\alpha}} + \theta\big(\tilde{A}_h \mathbf{w}^{\ell+1}\big)_{\boldsymbol{\alpha}} = (1-\theta) f(\mathbf{x}_{\boldsymbol{\alpha}}, \tau_\ell) + \theta f(\mathbf{x}_{\boldsymbol{\alpha}}, \tau_{\ell+1}) + O(k) + O(h^2)$$
$$\text{for all } \boldsymbol{\alpha} \text{ corresponding to interior grid points, } \ell = 0,\ldots,K-1. \tag{2.21}$$

The θ-method for the model problem (2.16)–(2.18) now defines the approximation vectors $\mathbf{W}^\ell$ as the solution of the system of linear equations

$$\frac{W^{\ell+1}_{\boldsymbol{\alpha}} - W^{\ell}_{\boldsymbol{\alpha}}}{k} + (1-\theta)\big(\tilde{A}_h \mathbf{W}^{\ell}\big)_{\boldsymbol{\alpha}} + \theta\big(\tilde{A}_h \mathbf{W}^{\ell+1}\big)_{\boldsymbol{\alpha}} = (1-\theta) f(\mathbf{x}_{\boldsymbol{\alpha}}, \tau_\ell) + \theta f(\mathbf{x}_{\boldsymbol{\alpha}}, \tau_{\ell+1})$$
$$\text{for all } \boldsymbol{\alpha} \text{ corresponding to interior grid points, } \ell = 0,\ldots,K-1, \tag{2.22}$$

$$W^0_{\boldsymbol{\alpha}} = w_0(\mathbf{x}_{\boldsymbol{\alpha}}) \qquad \text{for all } \boldsymbol{\alpha}, \tag{2.23}$$

$$W^\ell_{\boldsymbol{\alpha}} = g(\mathbf{x}_{\boldsymbol{\alpha}}, \tau_\ell) \qquad \text{for all } \boldsymbol{\alpha} \text{ corresponding to boundary grid points, } \ell = 1,\ldots,K. \tag{2.24}$$

Hence, all $\mathbf{W}^{\ell+1}$ can be successively computed, starting from the vector $\mathbf{W}^0$ determined by the initial condition (2.17).

To obtain a clear matrix notation of this method, we decompose the set of all multiindices $\boldsymbol{\alpha}$ of the mesh $\Omega^{\mathrm{x}}_{R,h}$ into the set $\boldsymbol{\alpha}_{\mathrm{int}}$ of indices corresponding to interior grid points and the set $\boldsymbol{\alpha}_{\mathrm{bd}}$ of indices corresponding to boundary grid points, and decompose the vectors $\mathbf{W}^\ell$ like

$$\mathbf{W}^\ell = \mathbf{W}^{\ell,\mathrm{int}} + \mathbf{W}^{\ell,\mathrm{bd}} \tag{2.25}$$

where

$$W^{\ell,\mathrm{int}}_{\boldsymbol{\alpha}} := \begin{cases} W^\ell_{\boldsymbol{\alpha}}, & \text{if } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{int}}, \\ 0, & \text{if } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{bd}}, \end{cases} \qquad W^{\ell,\mathrm{bd}}_{\boldsymbol{\alpha}} := \begin{cases} 0, & \text{if } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{int}}, \\ W^\ell_{\boldsymbol{\alpha}}, & \text{if } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{bd}}. \end{cases} \tag{2.26}$$

Furthermore, we define the right-hand side discretisation vectors $\mathbf{f}^\ell$,

$$f^\ell_{\boldsymbol{\alpha}} := \begin{cases} f(\mathbf{x}_{\boldsymbol{\alpha}}, \tau_\ell), & \text{if } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{int}}, \\ 0, & \text{if } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{bd}}, \end{cases} \tag{2.27}$$

and the vectors $\mathbf{g}^\ell$ containing the boundary values (2.24),

$$g^\ell_{\boldsymbol{\alpha}} := \begin{cases} 0, & \text{if } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{int}}, \\ g(\mathbf{x}_{\boldsymbol{\alpha}}, \tau_\ell), & \text{if } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{bd}}. \end{cases} \tag{2.28}$$

Thus, the boundary components of the unknown approximations $\mathbf{W}^\ell$ can be immediately defined by

$$\mathbf{W}^{\ell,\mathrm{bd}} = \mathbf{g}^\ell \quad \text{for } \ell = 1, \dots, K. \tag{2.29}$$

Hence, the only true unknowns are the components $\mathbf{W}^{\ell,\mathrm{int}}$ for $\ell = 1, \dots, K$. Using the decomposition (2.25), the θ-method equation (2.22) becomes

$$\begin{aligned} \frac{1}{k}\mathbf{W}^{\ell+1,\mathrm{int}} - \frac{1}{k}\mathbf{W}^{\ell,\mathrm{int}} + (1-\theta)\big(\tilde{A}_h \mathbf{W}^{\ell,\mathrm{int}}\big)^{\mathrm{int}} + \theta\big(\tilde{A}_h \mathbf{W}^{\ell+1,\mathrm{int}}\big)^{\mathrm{int}} \\ + (1-\theta)\big(\tilde{A}_h \mathbf{W}^{\ell,\mathrm{bd}}\big)^{\mathrm{int}} + \theta\big(\tilde{A}_h \mathbf{W}^{\ell+1,\mathrm{bd}}\big)^{\mathrm{int}} = (1-\theta)\mathbf{f}^\ell + \theta\,\mathbf{f}^{\ell+1}. \end{aligned} \tag{2.30}$$

Now using (2.29) and recalling that according to our definition of the matrix $\tilde{A}_h$, $\big(\tilde{A}_h \mathbf{x}\big)^{\mathrm{bd}} = \mathbf{0}$ and thus $\big(\tilde{A}_h \mathbf{x}\big)^{\mathrm{int}} = \tilde{A}_h \mathbf{x}$ for any vector $\mathbf{x}$, (2.30) can be rewritten as

$$\Big(\frac{1}{k} I + \theta \tilde{A}_h\Big)\mathbf{W}^{\ell+1,\mathrm{int}} = \Big(\frac{1}{k} I - (1-\theta)\tilde{A}_h\Big)\mathbf{W}^{\ell,\mathrm{int}} + (1-\theta)\mathbf{f}^\ell + \theta\,\mathbf{f}^{\ell+1} - (1-\theta)\tilde{A}_h \mathbf{g}^\ell - \theta \tilde{A}_h \mathbf{g}^{\ell+1}. \tag{2.31}$$

This is a $(2M+1)^n \times (2M+1)^n$ linear system for each $\mathbf{W}^{\ell+1,\mathrm{int}}$. Note, however, that all lines of this system which correspond to a boundary index reduce to "$0 = 0$" and are thus redundant. The only sought-after components of $\mathbf{W}^{\ell+1,\mathrm{int}}$ are those corresponding to interior indices, anyway. The system (2.31) can thus be reduced to a $(2M-1)^n \times (2M-1)^n$ linear system involving exclusively the "interior" components:

Let `intind` and `bdind` denote vectors listing the interior and boundary indices, respectively, in the same order as they appear in the vectors $\mathbf{W}^\ell$. For vectors $\mathbf{x}$ and $\mathbf{y}$, we will use the MATLAB syntax $\mathbf{W}(\mathbf{x})$ and $A(\mathbf{x}, \mathbf{y})$ to denote, respectively, the vector made up of the components listed in $\mathbf{x}$ of the vector $\mathbf{W}$, and the matrix made up of the lines listed in $\mathbf{x}$ and the columns listed in $\mathbf{y}$ of the matrix A. The unknown components of $\mathbf{W}^{\ell+1,\mathrm{int}}$ can therefore be written as $\mathbf{W}^{\ell+1,\mathrm{int}}$(`intind`),

or equivalently, as $\mathbf{W}^{\ell+1}(\texttt{intind})$, which is a $(2M-1)^n$-component vector. In components, the system (2.31) reads

$$\frac{1}{k}\,W_{\boldsymbol{\alpha}}^{\ell+1,\mathrm{int}} + \theta\big(\tilde{A}_h\mathbf{W}^{\ell+1,\mathrm{int}}\big)_{\boldsymbol{\alpha}} = \frac{1}{k}\,W_{\boldsymbol{\alpha}}^{\ell,\mathrm{int}} - (1-\theta)\big(\tilde{A}_h\mathbf{W}^{\ell,\mathrm{int}}\big)_{\boldsymbol{\alpha}} + (1-\theta)f_{\boldsymbol{\alpha}}^{\ell} + \theta\, f_{\boldsymbol{\alpha}}^{\ell+1} - (1-\theta)\big(\tilde{A}_h\mathbf{g}^{\ell}\big)_{\boldsymbol{\alpha}} - \theta\big(\tilde{A}_h\mathbf{g}^{\ell+1}\big)_{\boldsymbol{\alpha}}. \quad (2.32)$$

For components corresponding to interior grid points, i.e. for $\boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{int}}$, we have

$$\big(\tilde{A}_h\mathbf{W}^{\ell,\mathrm{int}}\big)_{\boldsymbol{\alpha}} = \sum_{\boldsymbol{\beta}}(\tilde{A}_h)_{\boldsymbol{\alpha}\boldsymbol{\beta}} \underbrace{W_{\boldsymbol{\beta}}^{\ell,\mathrm{int}}}_{=0 \text{ if } \boldsymbol{\beta}\in\boldsymbol{\alpha}_{\mathrm{bd}}} = \sum_{\boldsymbol{\beta}\in\boldsymbol{\alpha}_{\mathrm{int}}} (\tilde{A}_h)_{\boldsymbol{\alpha}\boldsymbol{\beta}}\, W_{\boldsymbol{\beta}}^{\ell,\mathrm{int}}, \quad (2.33)$$

hence

$$\big(\tilde{A}_h\mathbf{W}^{\ell,\mathrm{int}}\big)(\texttt{intind}) = \tilde{A}_h\,(\texttt{intind},\texttt{intind})\,\mathbf{W}^{\ell,\mathrm{int}}(\texttt{intind}) \quad (2.34)$$

for any ℓ. Analogously, we find that

$$\big(\tilde{A}_h\mathbf{g}^{\ell}\big)(\texttt{intind}) = \tilde{A}_h\,(\texttt{intind},\texttt{bdind})\,\mathbf{g}^{\ell}(\texttt{bdind}). \quad (2.35)$$

We introduce the short notations

$$\tilde{A}_h^{(\mathrm{ii})} := \tilde{A}_h\,(\texttt{intind},\texttt{intind}) \qquad \in \mathbb{R}^{(2M-1)^n\times(2M-1)^n}, \quad (2.36)$$

$$\tilde{A}_h^{(\mathrm{ib})} := \tilde{A}_h\,(\texttt{intind},\texttt{bdind}) \qquad \in \mathbb{R}^{(2M-1)^n\times((2M+1)^n-(2M-1)^n)}. \quad (2.37)$$

Now extracting only the lines corresponding to interior indices from the linear system (2.31) and using (2.34) and (2.35), we obtain the $(2M-1)^n \times (2M-1)^n$ systems,

$$\Big(\frac{1}{k}I + \theta\tilde{A}_h^{(\mathrm{ii})}\Big)\mathbf{W}^{\ell+1,\mathrm{int}}(\texttt{intind}) = \Big(\frac{1}{k}I - (1-\theta)\tilde{A}_h^{(\mathrm{ii})}\Big)\mathbf{W}^{\ell,\mathrm{int}}(\texttt{intind}) + (1-\theta)\mathbf{f}^{\ell}(\texttt{intind}) + \theta\,\mathbf{f}^{\ell+1}(\texttt{intind}) - (1-\theta)\tilde{A}_h^{(\mathrm{ib})}\mathbf{g}^{\ell}(\texttt{bdind}) - \theta\tilde{A}_h^{(\mathrm{ib})}\mathbf{g}^{\ell+1}(\texttt{bdind}) \quad \text{for } \ell = 0,\ldots,K-1, \quad (2.38)$$

or in short (following the notation of [Morton and Mayers, 1994, Chapter 5]),

$$B_1\ \mathbf{W}^{\ell+1}(\texttt{intind}) = B_0\ \mathbf{W}^{\ell}(\texttt{intind}) + \mathbf{F}^{\ell}, \qquad \ell = 0,\ldots,K-1, \quad (2.39)$$

with the matrices

$$B_1 := \frac{1}{k}I + \theta\tilde{A}_h^{(\mathrm{ii})} \qquad \in \mathbb{R}^{(2M-1)^n\times(2M-1)^n}, \quad (2.40)$$

$$B_0 := \frac{1}{k}I - (1-\theta)\tilde{A}_h^{(\mathrm{ii})} \qquad \in \mathbb{R}^{(2M-1)^n\times(2M-1)^n}, \quad (2.41)$$

and the vectors

$$\mathbf{F}^{\ell} := (1-\theta)\mathbf{f}^{\ell}(\texttt{intind}) + \theta\,\mathbf{f}^{\ell+1}(\texttt{intind}) - (1-\theta)\tilde{A}_h^{(\mathrm{ib})}\mathbf{g}^{\ell}(\texttt{bdind}) - \theta\tilde{A}_h^{(\mathrm{ib})}\mathbf{g}^{\ell+1}(\texttt{bdind}). \quad (2.42)$$

Equation (2.39) represents the pure matrix formulation of the θ-method, comprising solely the values at interior grid points as unknowns, the boundary conditions (2.24) being incorporated in the

right-hand side vectors $\mathbf{F}^\ell$.

For $\theta = 0$, we obtain the so-called *explicit* or *explicit Euler method*,

$$\frac{1}{k}\mathbf{W}^{\ell+1}(\texttt{intind}) = \Big(\frac{1}{k}I - \tilde{A}_h^{(\mathrm{ii})}\Big)\mathbf{W}^{\ell}(\texttt{intind}) + \mathbf{f}^{\ell}(\texttt{intind}) - \tilde{A}_h^{(\mathrm{ib})}\mathbf{g}^{\ell}(\texttt{bdind}) \tag{2.43}$$

providing an explicit formula to compute each interior component of $\mathbf{W}^{\ell+1}$ individually. This method is for instance used for the one-dimensional untransformed Black-Scholes equation in [Schwartz, 1977], one of the first publications on the use of finite-difference methods in the financial literature. However, although the explicit method is very intuitive, its practical use is often limited due to stability deficiencies; specifically, unless the mesh ratio $\frac{k}{h^2}$ is smaller than some constant depending on the problem, the explicit method is unstable and hence magnifies rounding errors at each time step to such an extent that the result may become meaningless (cf. [Morton and Mayers, 1994, Sections 2.4 and 3.1], [Wilmott et al., 1995, Section 8.4], and Section 2.3).

This problem can be avoided by choosing different values of θ, yet at the cost of higher computational complexity. For $\theta > 0$, we obtain *implicit methods* where the components of $\mathbf{W}^{\ell+1}$ cannot be computed individually, but where a simultaneous system of equations has to be solved to obtain the vector $\mathbf{W}^{\ell+1}(\texttt{intind})$.

For $\theta = 1$, we obtain the so-called *fully implicit* or *implicit Euler method*

$$\Big(\frac{1}{k}I + \tilde{A}_h^{(\mathrm{ii})}\Big)\mathbf{W}^{\ell+1}(\texttt{intind}) = \frac{1}{k}\mathbf{W}^{\ell}(\texttt{intind}) + \mathbf{f}^{\ell+1}(\texttt{intind}) - \tilde{A}_h^{(\mathrm{ib})}\mathbf{g}^{\ell+1}(\texttt{bdind}), \tag{2.44}$$

or, formally,

$$\mathbf{W}^{\ell+1}(\texttt{intind}) = \Big(\frac{1}{k}I + \tilde{A}_h^{(\mathrm{ii})}\Big)^{-1}\left[\frac{1}{k}\mathbf{W}^{\ell}(\texttt{intind}) + \mathbf{f}^{\ell+1}(\texttt{intind}) - \tilde{A}_h^{(\mathrm{ib})}\mathbf{g}^{\ell+1}(\texttt{bdind})\right]. \tag{2.45}$$

This method turns out to have no stability limitations.

However, an even more sophisticated choice is $\theta = \frac{1}{2}$ yielding the so-called the *Crank-Nicolson method*

$$\begin{aligned}\Big(\frac{1}{k}I + \frac{1}{2}\tilde{A}_h^{(\mathrm{ii})}\Big)\mathbf{W}^{\ell+1}(\texttt{intind}) &= \Big(\frac{1}{k}I - \frac{1}{2}\tilde{A}_h^{(\mathrm{ii})}\Big)\mathbf{W}^{\ell}(\texttt{intind}) \\ &+ \frac{1}{2}\Big(\mathbf{f}^{\ell}(\texttt{intind}) + \mathbf{f}^{\ell+1}(\texttt{intind})\Big) - \frac{1}{2}\tilde{A}_h^{(\mathrm{ib})}\Big(\mathbf{g}^{\ell}(\texttt{bdind}) + \mathbf{g}^{\ell+1}(\texttt{bdind})\Big),\end{aligned} \tag{2.46}$$

or, formally,

$$\begin{aligned}\mathbf{W}^{\ell+1}(\texttt{intind}) = \Big(I + \frac{k}{2}\tilde{A}_h^{(\mathrm{ii})}\Big)^{-1}\Bigg[&\Big(I - \frac{k}{2}\tilde{A}_h^{(\mathrm{ii})}\Big)\mathbf{W}^{\ell}(\texttt{intind}) \\ &+\frac{k}{2}\Big(\mathbf{f}^{\ell}(\texttt{intind}) + \mathbf{f}^{\ell+1}(\texttt{intind})\Big) - \frac{k}{2}\tilde{A}_h^{(\mathrm{ib})}\Big(\mathbf{g}^{\ell}(\texttt{bdind}) + \mathbf{g}^{\ell+1}(\texttt{bdind})\Big)\Bigg].\end{aligned} \tag{2.47}$$

This method has a higher order of accuracy than the other θ-methods, cf. Section 2.3.1.

2.3 On the convergence of numerical schemes

The exact solution vectors $\mathbf{W}^\ell$ of the system of linear equations (2.39) are only approximations of the exact solution w of the differential problem (2.16)–(2.18). Natural requirements for a numerical scheme to provide "reasonable" approximations are

1. that the scheme is uniquely solvable, and
2. that the $\mathbf{W}^\ell$ converge to the exact solution w as the mesh is refined (in a sense to be defined more precisely below).

We can prove the first property for the θ-method for our model problem, the truncated transformed Black-Scholes problem (1.57)–(1.59). Following [Morton and Mayers, 1994, Chapter 5], we will also define some properties closely linked to the convergence of a numerical scheme in this section, even if we will ultimately not be able to apply them to our problem for lack of regularity.

Note that for the rest of this chapter, we will use the notation $\mathbf{W}^\ell$ that formerly denoted the vector of approximations at *all* grid points of the spatial mesh at time τ_ℓ and that hence included the known boundary values, to denote the vector of approximations *at the interior grid points only*. This is the true set of unknowns, hence the output of the numerical scheme, which was denoted as $\mathbf{W}^\ell(\texttt{intind})$ in the previous section.

In order to be able to measure errors and define convergence, we need to fix a norm. Common norms in this context are the *maximum norm*,

$$\|\mathbf{W}^\ell\|_\infty := \max_{\boldsymbol{\alpha}\in\boldsymbol{\alpha}_{\text{int}}} |W^\ell_{\boldsymbol{\alpha}}|, \tag{2.48}$$

and the *discrete ℓ^2-norm* in $\mathbb{R}^n$ associated with mesh width h,

$$\|\mathbf{W}^\ell\|_2 := h^{\frac{n}{2}} \Big(\sum_{\boldsymbol{\alpha}\in\boldsymbol{\alpha}_{\text{int}}} |W^\ell_{\boldsymbol{\alpha}}|^2 \Big)^{\frac{1}{2}}. \tag{2.49}$$

These discrete norms can also be defined for continuous functions $u \in C(\overline{\Omega_R}\times[0,T])$ by identifying $u(\,.\,,\tau_\ell)$ with the vector u^ℓ of mesh point values and the vector of root cell averages, respectively,

$$u^\ell := \big(u(\mathbf{x}_{\boldsymbol{\alpha}},\tau_\ell)\big)_{\boldsymbol{\alpha}\in\boldsymbol{\alpha}_{\text{int}}} \tag{2.50}$$

$$\text{and} \qquad u^\ell := \Big(h^{-\frac{n}{2}} \Big(\int_{V_{\boldsymbol{\alpha}}} |u(\mathbf{x},\tau_\ell)|^2\, d\mathbf{x}\Big)^{\frac{1}{2}}\Big)_{\boldsymbol{\alpha}\in\boldsymbol{\alpha}_{\text{int}}}, \tag{2.51}$$

respectively, where $V_{\boldsymbol{\alpha}}$ denotes the axis-aligned cube of side length h centered at $\mathbf{x}_{\boldsymbol{\alpha}}$ (cf. [Morton and Mayers, 1994, Chapter 5]). The discrete norms $\|u^\ell\|$ then approximate the L^∞ and the integral L^2 norm of $u(\,.\,,\tau_\ell)$ on $\overline{\Omega_R}$, respectively; more precisely, we have

$$\|u^\ell\|_\infty \le \|u(\,.\,,\tau_\ell)\|_{C(\overline{\Omega_R})} \text{ for all } h>0 \quad \text{and} \quad \|u^\ell\|_\infty \to \|u(\,.\,,\tau_\ell)\|_{C(\overline{\Omega_R})} \text{ as } h\to 0, \tag{2.52}$$

$$\|u^\ell\|_2 \le \|u(\,.\,,\tau_\ell)\|_{L^2(\Omega_R)} \text{ for all } h>0 \quad \text{and} \quad \|u^\ell\|_2 \to \|u(\,.\,,\tau_\ell)\|_{L^2(\Omega_R)} \text{ as } h\to 0. \tag{2.53}$$

Note that for any *fixed* spatial mesh, the two discrete norms are equivalent according to

$$h^{\frac{n}{2}}\|\mathbf{W}\|_\infty \le \|\mathbf{W}\|_2 \le \big(2R\big)^{\frac{n}{2}}\|\mathbf{W}\|_\infty; \tag{2.54}$$

however, the equivalence is not preserved in the limit $h\to 0$, as the left bound tends to zero.

We will now examine the solvability of numerical schemes.

Definition 3. We call a numerical scheme of the form (2.39),

$$B_1\ \mathbf{W}^{\ell+1} = B_0\ \mathbf{W}^{\ell} + \mathbf{F}^{\ell} \qquad \text{for } \ell = 0, \dots, K-1, \tag{2.55}$$

uniformly solvable with respect to the norm $\|\cdot\|$ if there exists a constant C such that

$$\|B_1^{-1}\| \leq C\, k \tag{2.56}$$

for all k sufficiently small, independently of the dimension of the matrix.

Remark. When applied to a matrix like in (2.56), $\|\cdot\|$ is understood to be the matrix norm induced by the vector norm $\|\cdot\|$, i.e. for $A \in \mathbb{R}^{N\times N}$,

$$\|A\| := \sup_{\mathbf{x}\in\mathbb{R}^N\setminus\{\mathbf{0}\}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}. \tag{2.57}$$

Proposition 13. (generalised from a result in [Morton and Mayers, 1994, Section 5.3]).

For the explicit Euler method for the truncated transformed Black-Scholes problem (1.57)–(1.59), (2.56) *holds with* $C = 1$ *for* $\|\cdot\| = \|\cdot\|_\infty$, *i.e. the method is always uniformly solvable with respect to the maximum norm. For* $\theta > 0$, *the* θ*-method for* (1.57)–(1.59) *is uniformly solvable with respect to the maximum norm if*

$$h \leq \min_{i=1,\dots,n} \frac{\sigma_i^2}{\left|\frac{\sigma_i^2}{2} - r\right|} \qquad \textit{and} \qquad \frac{k}{h^2} < \frac{1}{\theta\left(\sum_{i<j}\sigma_i\sigma_j|\rho_{ij}|\right)}. \tag{2.58}$$

Proof. In order to prove (2.56), we need to show that there is a constant C such that

$$\frac{\|\mathbf{W}\|_\infty}{\|\mathbf{F}\|_\infty} \leq C\, k \quad \text{for all } \mathbf{F} \in \mathbb{R}^{(2M-1)^n} \setminus \{\mathbf{0}\},\ \mathbf{W} = B_1^{-1}\mathbf{F}. \tag{2.59}$$

To this end, we estimate $\|\mathbf{W}\|_\infty$ in terms of $\|\mathbf{F}\|_\infty$ where $\mathbf{F} = B_1\mathbf{W} = \frac{1}{k}\mathbf{W} + \theta A_h^{(\mathrm{ii})}\mathbf{W}$.

If $\theta = 0$, i.e. for the explicit method, we have $\mathbf{F} = \frac{1}{k}\mathbf{W}$ for all components and thus immediately see that (2.59) holds with $C = 1$.

For arbitrary θ, we have to take a closer look at $A_h^{(\mathrm{ii})}\mathbf{W}$. Regarding (2.14), we have for a vector $\tilde{\mathbf{W}} \in \mathbb{R}^{(2M+1)^n}$ that

$$\begin{aligned}\left(A_h\tilde{\mathbf{W}}\right)_{\boldsymbol{\alpha}} = \Big(r + \sum_{i=1}^n \frac{\sigma_i^2}{h^2}\Big)\tilde{W}_{\boldsymbol{\alpha}} + \sum_{i=1}^n \Big(-\frac{\sigma_i^2}{2h^2} + \frac{\sigma_i^2}{4h} - \frac{r}{2h}\Big)\tilde{W}_{\boldsymbol{\alpha}+\mathbf{e}_i} + \sum_{i=1}^n \Big(-\frac{\sigma_i^2}{2h^2} - \frac{\sigma_i^2}{4h} + \frac{r}{2h}\Big)\tilde{W}_{\boldsymbol{\alpha}-\mathbf{e}_i} \\ + \sum_{i<j} \frac{\sigma_i\sigma_j\rho_{ij}}{4h^2}\left(-\tilde{W}_{\boldsymbol{\alpha}+\mathbf{e}_i+\mathbf{e}_j} + \tilde{W}_{\boldsymbol{\alpha}+\mathbf{e}_i-\mathbf{e}_j} + \tilde{W}_{\boldsymbol{\alpha}-\mathbf{e}_i+\mathbf{e}_j} - \tilde{W}_{\boldsymbol{\alpha}-\mathbf{e}_i-\mathbf{e}_j}\right)\end{aligned} \tag{2.60}$$

for all $\boldsymbol{\alpha}$ corresponding to interior grid points. The components of $A_h^{(\mathrm{ii})}\mathbf{W}$ are of the same form, except that summands corresponding to neighbouring grid points are omitted if these neighbours lie on the boundary. In any case, the estimate

$$\begin{aligned}\left|\left(A_h^{(\mathrm{ii})}\mathbf{W}\right)_{\boldsymbol{\beta}} - \Big(r + \sum_{i=1}^n \frac{\sigma_i^2}{h^2}\Big)W_{\boldsymbol{\beta}}\right| \leq \sum_{i=1}^n \Big|-\frac{\sigma_i^2}{2h^2} + \frac{\sigma_i^2}{4h} - \frac{r}{2h}\Big|\|\mathbf{W}\|_\infty + \sum_{i=1}^n \Big|-\frac{\sigma_i^2}{2h^2} - \frac{\sigma_i^2}{4h} + \frac{r}{2h}\Big|\|\mathbf{W}\|_\infty \\ + 4\sum_{i<j}\frac{\sigma_i\sigma_j|\rho_{ij}|}{4h^2}\|\mathbf{W}\|_\infty\end{aligned} \tag{2.61}$$

holds for all components of $A_h^{(\mathrm{ii})}\mathbf{W}$. The first constraint of (2.58) is equivalent to

$$\frac{\sigma_i^2}{2h^2} - \frac{\sigma_i^2}{4h} + \frac{r}{2h} \geq 0 \quad \text{and} \quad \frac{\sigma_i^2}{2h^2} + \frac{\sigma_i^2}{4h} - \frac{r}{2h} \geq 0 \quad \text{for all } i, \tag{2.62}$$

so that (2.61) reduces to

$$\begin{aligned}\Big|\big(A_h^{(\mathrm{ii})}\mathbf{W}\big)_{\boldsymbol{\beta}} - \Big(r + \sum_{i=1}^n \frac{\sigma_i^2}{h^2}\Big)W_{\boldsymbol{\beta}}\Big| &\leq \sum_{i=1}^n \Big(\frac{\sigma_i^2}{2h^2} - \frac{\sigma_i^2}{4h} + \frac{r}{2h}\Big)\|\mathbf{W}\|_\infty + \sum_{i=1}^n \Big(\frac{\sigma_i^2}{2h^2} + \frac{\sigma_i^2}{4h} - \frac{r}{2h}\Big)\|\mathbf{W}\|_\infty \\ &\quad + \sum_{i<j} \frac{\sigma_i\sigma_j|\rho_{ij}|}{h^2}\|\mathbf{W}\|_\infty \\ &= \Big(\sum_{i=1}^n \frac{\sigma_i^2}{h^2} + \sum_{i<j} \frac{\sigma_i\sigma_j|\rho_{ij}|}{h^2}\Big)\|\mathbf{W}\|_\infty. \end{aligned} \tag{2.63}$$

From

$$F_{\boldsymbol{\beta}} = \frac{1}{k}W_{\boldsymbol{\beta}} + \theta\big(A_h^{(\mathrm{ii})}\mathbf{W}\big)_{\boldsymbol{\beta}}, \tag{2.64}$$

it follows that

$$\Big(1 + k\theta\Big(r + \sum_{i=1}^n \frac{\sigma_i^2}{h^2}\Big)\Big)W_{\boldsymbol{\beta}} = kF_{\boldsymbol{\beta}} - k\theta\Big(\big(A_h^{(\mathrm{ii})}\mathbf{W}\big)_{\boldsymbol{\beta}} - \Big(r + \sum_{i=1}^n \frac{\sigma_i^2}{h^2}\Big)W_{\boldsymbol{\beta}}\Big) \tag{2.65}$$

for all $\boldsymbol{\beta}$. Taking absolute values, using (2.63) and passing to the supremum, we finally obtain

$$\underbrace{\Big(1 + k\theta r - k\theta \sum_{i<j} \frac{\sigma_i\sigma_j|\rho_{ij}|}{h^2}\Big)}_{=:\tilde{C}} \|\mathbf{W}\|_\infty \leq k\|\mathbf{F}\|_\infty. \tag{2.66}$$

The second condition in (2.58) now implies that

$$\tilde{C} \quad = \quad 1 + \underbrace{k\theta r}_{\geq 0} - \underbrace{\frac{k}{h^2}}_{< \frac{1}{\theta\left(\sum_{i<j}\sigma_i\sigma_j|\rho_{ij}|\right)}} \theta \sum_{i<j} \sigma_i\sigma_j|\rho_{ij}|\Big) \quad > \quad 0, \tag{2.67}$$

i.e. (2.59) is satisfied for $C := \frac{1}{\tilde{C}}$. □

We are now ready to define in which sense we wish the numerical approximations to converge:

Definition 4. A numerical scheme for the problem (2.16)–(2.18) with exact solution w is called *convergent* in the norm $\|\cdot\|$ if

$$\|\mathbf{W}^\ell - w^\ell\| \to 0 \quad \text{as } h \to 0, k \to 0,\ \ell k \to \tau \in (0, T] \tag{2.68}$$

for all initial data w_0 for which the problem is *well-posed* in $\|\cdot\|$. As h and k both tend to zero, they may be required to satisfy certain relations.

Remark. For the exact definition of well-posedness, we refer to [Morton and Mayers, 1994, Section 5.5]; broadly speaking, a differential problem is well-posed if for a sufficiently large set of initial functions, there exists a unique solution which is sufficiently regular and depends continuously on the data.

For well-posed linear differential problems, the following convergence theorem due to Lax applies:

Theorem 14 (Lax Equivalence Theorem)**.** ([Morton and Mayers, 1994, Theorem 5.1]).

A numerical scheme that is consistent *with a linear well-posed parabolic differential problem and that is uniformly solvable in the sense of* (2.56) *is convergent if and only if it is* stable *(where all properties have to hold with respect to the same norm).*

Remark. The proof of this theorem in [Morton and Mayers, 1994] reveals that under the above conditions, the error $\mathbf{W}^\ell - w^\ell$ for a smooth solution w is of the same order as the truncation error defined in (2.69) below.

The key properties of the numerical scheme are hence

1. *consistency*, which ensures that the numerical scheme in fact approximates the correct differential equation, and

2. *stability*, which is analogous to the requirement of continuous dependence on the data for the differential problem and ensures that errors (e.g., rounding errors) are not unduly amplified in each step of the numerical scheme.

The definitions of these properties are not entirely consistent in the literature; we adopt those by [Morton and Mayers, 1994, Chapter 5] which read as follows:

Definition 5. A numerical scheme is said to be *consistent* with the differential problem (2.16)–(2.18), if the *truncation error* $\mathbf{T}^\ell$ defined by

$$T^\ell_{\boldsymbol{\alpha}} := \left(B_1\ w^{\ell+1}\right)_{\boldsymbol{\alpha}} - \left(B_0\ w^{\ell}\right)_{\boldsymbol{\alpha}} - F^\ell_{\boldsymbol{\alpha}} \quad \text{for all } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\text{int}}, \ell = 0, \ldots, K-1 \tag{2.69}$$

satisfies

$$T^\ell_{\boldsymbol{\alpha}} \to 0 \quad \text{as } h \to 0, k \to 0 \ \text{ for all } \boldsymbol{\alpha} \text{ and } \ell, \tag{2.70}$$

for all sufficiently smooth solutions w of the differential problem.

Hence, a numerical scheme is consistent with a PDE if in the limit, the exact solution solves the numerical scheme. Depending on the order of convergence of the truncation error, we define the order of accuracy of the numerical scheme:

Definition 6. A numerical scheme is said to have *order of accuracy p in k and q in h*, if p and q are the largest integers for which there exists a constant C such that

$$|T^\ell_{\boldsymbol{\alpha}}| \leq C\,(k^p + h^q) \quad \text{as } h \to 0, k \to 0 \ \text{ for all } \boldsymbol{\alpha} \text{ and } \ell \tag{2.71}$$

for all sufficiently smooth solutions w of the differential problem.

Definition 7. Let $\mathbf{V}^\ell$ and $\mathbf{W}^\ell$ be two solutions of the numerical scheme (2.55) starting from different initial data $\mathbf{V}^0$ and $\mathbf{W}^0$ but having the same right-hand side $\mathbf{F}^\ell$. We then say that the scheme is *stable* in the norm $\|\cdot\|$ if there exists a constant C such that

$$\|\mathbf{V}^\ell - \mathbf{W}^\ell\| \leq C\,\|\mathbf{V}^0 - \mathbf{W}^0\| \quad \text{for all } \ell k \leq T. \tag{2.72}$$

Remark. In the case of a linear problem, the difference between two solutions with the same right-hand side $\mathbf{F}^\ell$ satisfies the same linear equation with *homogeneous* right-hand side. In this case, (2.72) is therefore equivalent to the requirement

$$\|(B_1^{-1}B_0)^\ell\| \leq C \quad \text{for all } \ell k \leq T. \tag{2.73}$$

2.3.1 Convergence of the θ-method

Let us now turn to the analysis of the θ-method for the Black-Scholes problem (1.57)–(1.59) that we are examining here.

The θ-method with $\theta < \frac{1}{2}$ is usually only stable if a constraint of the form $\frac{k}{h^2} \leq C$ is met (see, e.g., [Morton and Mayers, 1994, Section 2.10]), i.e. the time step has to be very small compared to the spatial mesh size, entailing a great amount of computational effort. In contrast, θ-methods with $\theta \geq \frac{1}{2}$ are usually unconditionally stable which can make them ultimately even more efficient than the explicit method. For the problem (1.57)–(1.59), local Fourier analysis (cf. [Morton and Mayers, 1994, Sections 2.7 and 5.6], [Smith, 1978, Chapter 3]) can be used to prove unconditional practical stability with respect to the discrete ℓ^2-norm.

The comparison of (2.21) and (2.22) indicates that the θ-method is consistent with the Black-Scholes problem. For sufficiently smooth solutions $\tilde{w}$, all components of the truncation error are uniformly $O(h^2) + O(k)$, i.e. the method is of second order in h and of first order in k. For the Crank-Nicolson method, it can even be shown to be of second order in both h and k ([Morton and Mayers, 1994, Section 2.10], [Wilmott et al., 1995, Section 8.7]). Regarding the Remark to Theorem 14, the numerical method converges of the same order as the truncation error if the solution is sufficiently smooth.

The problem lies with well-posedness, however. The assumption of well-posedness in Theorem 14 ensures convergence of the numerical scheme in the sense of (2.68) even for generalised solutions which are not smooth enough to solve the problem in the classical sense but which may be obtained as limits of smooth solutions (cf. [Morton and Mayers, 1994, Section 5.5]). In our case, Corollary 8 in Section 1.4.2 ensures that the truncated Black-Scholes problem has a unique solution provided that the initial and boundary data are continuous, and this solution is even infinitely differentiable in the interior of the computational domain. However, the initial function involves the option's pay-off which is usually continuous but not continuously differentiable. Thus, we cannot expect a solution whose derivatives are uniformly bounded in $\overline{\Omega_R} \times [0,T]$, and consequently, the truncation error for the concrete problem is *not* uniformly bounded. This would not constitute a problem if we could approximate its solution arbitrarily well by solutions for smoother initial and boundary data which have uniformly bounded derivatives. However, even for arbitrarily smooth initial and boundary conditions, the existence of such smooth solutions to the problem (1.57)–(1.59) is open. [Friedman, 1964, Chapter 3] proves regularity results for initial-boundary value problems including differentiability of the solution up to the boundary of the domain, albeit under the condition that the boundary of the domain is accordingly smooth. In our case, however, $\partial\Omega_R \times [0,T]$ has "corners" and is thus not continuously differentiable.

Hence, we cannot a priori prove convergence of the θ-method for the Black-Scholes problem (1.57)–(1.59); we can only observe the behaviour of the error for special types of options where the exact solution is analytically known.

For stability reasons, we will use a θ-method with $\theta \geq \frac{1}{2}$ to numerically solve our model problem. We will present results obtained with the Crank-Nicolson method as well as the implicit Euler method.

Chapter 3

The multigrid method

All implicit θ-methods for the computation of the approximation vectors $\mathbf{W}^{\ell+1}(\texttt{intind})$ require the solution of a linear system of the form

$$B_1\ \mathbf{W}^{\ell+1}(\texttt{intind}) = B_0\ \mathbf{W}^{\ell}(\texttt{intind}) + \mathbf{F}^{\ell} \tag{3.1}$$

at each time step, with $B_1 = \frac{1}{k} I + \theta A_h^{(\mathrm{ii})}$ and $\theta \in (0, 1]$, or equivalently,

$$kB_1\ \mathbf{W}^{\ell+1}(\texttt{intind}) = kB_0\ \mathbf{W}^{\ell}(\texttt{intind}) + k\,\mathbf{F}^{\ell} \tag{3.2}$$

with the system matrix $kB_1 = I + k\,\theta A_h^{(\mathrm{ii})}$. The dimension of this system is $(2M-1)^n \times (2M-1)^n$ with n being the space dimension or number of underlying assets, and $2M+1$ the number of grid points in each direction. Hence the linear system to be solved will in general be very large, yet we have already observed in Section 2.2 that the matrix $A_h^{(\mathrm{ii})}$ is sparse, and hence so is B_1. For large sparse systems, iterative methods are in general preferable to direct solution methods (cf., e.g., [Smith, 1978, Chapter 5]). A particularly effective solution method is the *multigrid method* which is based on classical iterative (relaxation) methods but which has considerably better convergence properties at a comparable computational cost. In fact, the effort required by the so-called *full multigrid V-cycle scheme* to solve an $N \times N$ system up to discretisation accuracy is $O(N)$ (see, e.g., [Briggs, 1987, Chapter 4]) which is the optimal effort that can be achieved.

This is the method that we will use for our problem. In the following, we will briefly describe the basic ideas and techniques that we need; for a comprehensive treatment of multigrid methods, we refer to [Briggs, 1987], [Hackbusch, 1985] or [Hackbusch, 1993]. Broadly speaking, the idea of the multigrid method is to apply an iterative method (in our case, the *weighted Jacobi method*) on a sequence of meshes which are coarser than the original one.

3.1 The weighted Jacobi method

Suppose we want to solve the linear system

$$A\mathbf{W} = \mathbf{RHS} \tag{3.3}$$

for a given matrix $A \in \mathbb{R}^{N\times N}$ and right-hand side vector $\mathbf{RHS} \in \mathbb{R}^N$. The objective of the weighted Jacobi method, as for any iterative method, is to produce a sequence $\mathbf{W}_0, \mathbf{W}_1, \mathbf{W}_2, \ldots$ of vectors

which approximate the exact solution $\mathbf{W}$ while being less costly to compute. Let us introduce some further notation. The error in the m-th approximation is

$$\mathbf{E}_m := \mathbf{W} - \mathbf{W}_m, \tag{3.4}$$

and by the corresponding *residual* $\mathbf{R}_m$, we understand the difference between $\mathbf{RHS}$ and $A\mathbf{W}_m$,

$$\mathbf{R}_m := \mathbf{RHS} - A\mathbf{W}_m = A\mathbf{W} - A\mathbf{W}_m. \tag{3.5}$$

Note that the error satisfies the so-called *residual equation*

$$A\,\mathbf{E}_m = \mathbf{R}_m \tag{3.6}$$

which will be of importance in the multigrid method.

Let D denote the diagonal matrix made up of the diagonal elements of A. Starting from an arbitrary initial guess $\mathbf{W}_0$, the weighted Jacobi method then defines the approximations $\mathbf{W}_m$ recursively by

$$\begin{aligned} \mathbf{W}_{m+1} &:= \left(I - \omega D^{-1} A\right) \mathbf{W}_m + \omega D^{-1} \mathbf{RHS} \\ &= \mathbf{W}_m + \omega D^{-1} \mathbf{R}_m \end{aligned} \tag{3.7}$$

where $\omega \in (0, 1]$ is a weighting factor; for $\omega = 1$, we obtain the classical (unweighted) Jacobi method. If A is a sparse matrix with $O(N)$ non-zero elements, then the residual can be computed in $O(N)$ operations, hence each step of the weighted Jacobi method is of effort $O(N)$ in this case.

3.1.1 Convergence properties of the weighted Jacobi method

Iterative methods for the system (3.3) that are of the form

$$\mathbf{W}_0 \in \mathbb{R}^N \text{ arbitrary}, \quad \mathbf{W}_{m+1} := M\ \mathbf{W}_m + N\ \mathbf{RHS} \tag{3.8}$$

with matrices $M, N \in \mathbb{R}^{N \times N}$ are called *linear iterative methods*; M is called the corresponding *iteration matrix* (see, e.g., [Hackbusch, 1993, Chapter 3]). Clearly, the weighted Jacobi method (3.7) is of this form with

$$M = I - \omega D^{-1} A, \qquad N = \omega D^{-1}. \tag{3.9}$$

A necessary condition for the convergence of an iterative method is *consistency* with the linear system, i.e. that the solution $\mathbf{W}$ be a *fixed point* of the method for any right-hand side vector $\mathbf{RHS}$. For a linear iterative method, this is equivalent to the condition

$$M = I - NA \tag{3.10}$$

([Hackbusch, 1993, Theorem 3.2.2]). Thus, comparing (3.9) and (3.10), we immediately see that the weighted Jacobi method is consistent.

An iterative method is called *convergent* if for any right-hand side vector $\mathbf{RHS}$, the iterates $\mathbf{W}_m$ converge to a limit that is independent of the initial guess $\mathbf{W}_0$ as $m \to \infty$. [Hackbusch, 1993, Theorem 3.2.7] states that a linear iterative method is convergent if and only if

$$\rho(M) < 1 \tag{3.11}$$

where $\rho(M)$ denotes the spectral radius of the iteration matrix and is called the *convergence rate* of the method. If the method is additionally consistent, (3.11) implies that the iterates converge to the unique solution $\mathbf{W}$ of the linear system (3.3) ([Hackbusch, 1993, Supplement 3.2.8]).

A number of convergence criteria are known for the weighted Jacobi method applied to linear systems whose system matrix A is symmetric positive definite, an M-matrix, strictly diagonally dominant or irreducibly diagonally dominant (e.g., [Hackbusch, 1993, Sections 4.4 and 6.4]). The system matrix $kB_1 = I + k\,\theta A_h^{(\mathrm{ii})}$ of the linear system (3.2) with $\theta > 0$ is in general of neither of those types, nor is it straightforward how to analytically calculate the eigenvalues of B_1. We can, however, prove convergence of the weighted Jacobi method for the problem (3.2) by estimating the norm of the corresponding iteration matrix and making use of the relation

$$\rho(M) \leq \|M\| \tag{3.12}$$

for all matrix norms induced by a vector norm according to (2.57) (e.g., [Hackbusch, 1993, Section 2.9]). We will consider the matrix norm induced by the maximum norm (2.48); it can be shown (see, e.g., [Hackbusch, 1993, Section 2.6.3]) that the resulting matrix norm is given by

$$\| A \|_\infty = \max_{\boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{int}}} \sum_{\boldsymbol{\beta} \in \boldsymbol{\beta}_{\mathrm{int}}} |A_{\boldsymbol{\alpha}\boldsymbol{\beta}}|. \tag{3.13}$$

Proposition 15. *Under the conditions*

$$h \leq \min_{i=1,\ldots,n} \frac{\sigma_i^2}{\left|\frac{\sigma_i^2}{2} - r\right|} \qquad \text{and} \qquad \frac{k}{h^2} < \frac{1}{\theta\left(\sum_{i<j} \sigma_i \sigma_j |\rho_{ij}|\right)}, \tag{3.14}$$

the weighted Jacobi method for the problem (3.2) *with* $\theta \in (0, 1]$ *(i.e. any implicit* θ*-scheme discretisation of the Black-Scholes problem) converges for any* $\omega \in (0, 1]$.

Proof. We aim to show (3.11). As we have already verified that the weighted Jacobi method is a consistent linear iterative method, this implies that it converges to the solution of the linear system.

The diagonal entries of the system matrix $kB_1 = I_{(2M-1)^n} + k\,\theta A_h^{(\mathrm{ii})}$ of (3.2) are all equal to

$$d := 1 + k\,\theta\left(r + \sum_{i=1}^{n} \frac{\sigma_i^2}{h^2}\right) \tag{3.15}$$

regarding (2.14), hence the iteration matrix of the corresponding weighted Jacobi method is

$$M = I_{(2M-1)^n} - \omega\,(d\,I)^{-1}\left(I_{(2M-1)^n} + k\,\theta A_h^{(\mathrm{ii})}\right) = \left(1 - \frac{\omega}{d}\right) I_{(2M-1)^n} - \frac{\omega}{d}\,k\,\theta\,A_h^{(\mathrm{ii})} \tag{3.16}$$

by (3.9). According to (3.13), the norm $\|M\|_\infty$ of this matrix is equal to its maximum absolute row sum, i.e.

$$\begin{aligned} \|M\|_\infty = \Big|1 - \frac{\omega}{d} - \frac{\omega}{d}\,k\,\theta\left(r + \sum_{i=1}^{n} \frac{\sigma_i^2}{h^2}\right)\Big| + \frac{\omega}{d}\,k\,\theta\left(\sum_{i=1}^{n}\Big|-\frac{\sigma_i^2}{2h^2} + \frac{\sigma_i^2}{4h} - \frac{r}{2h}\Big|\right. \\ \left. + \sum_{i=1}^{n}\Big|-\frac{\sigma_i^2}{2h^2} - \frac{\sigma_i^2}{4h} + \frac{r}{2h}\Big| + 4\sum_{i<j}\frac{\sigma_i\sigma_j|\rho_{ij}|}{4h^2}\right) \end{aligned} \tag{3.17}$$

regarding (2.14). The first condition in (3.14) implies $\frac{\sigma_i^2}{2h^2} \mp \frac{\sigma_i^2}{4h} \pm \frac{r}{2h} \geq 0$ (cf. (2.58) and (2.62)). Hence, (3.17) reduces to

$$\begin{aligned}
\|M\|_\infty &= \Big|1 - \frac{\omega}{d}\Big(1 + k\,\theta\big(r + \sum_{i=1}^n \frac{\sigma_i^2}{h^2}\big)\Big)\Big| + \frac{\omega}{d}\,k\,\theta\left(\sum_{i=1}^n \frac{\sigma_i^2}{h^2} + \sum_{i<j} \frac{\sigma_i\sigma_j|\rho_{ij}|}{h^2}\right) \\
&= |1-\omega| + \frac{\omega}{d}\,\theta\,\frac{k}{h^2}\Big(\sum_{i=1}^n \sigma_i^2 + \sum_{i<j}\sigma_i\sigma_j|\rho_{ij}|\Big) \\
&= 1 - \omega\,\Big[1 - \frac{\theta}{d}\,\frac{k}{h^2}\Big(\sum_{i=1}^n \sigma_i^2 + \sum_{i<j}\sigma_i\sigma_j|\rho_{ij}|\Big)\Big]. \qquad (3.18)
\end{aligned}$$

Thus, we obtain $\|M\|_\infty < 1$ if and only if

$$1 - \frac{\theta}{d}\,\frac{k}{h^2}\Big(\sum_{i=1}^n \sigma_i^2 + \sum_{i<j}\sigma_i\sigma_j|\rho_{ij}|\Big) > 0, \qquad (3.19)$$

or equivalently, if

$$\frac{k}{h^2} < \frac{\frac{1}{\theta} + kr}{\sum_{i<j}\sigma_i\sigma_j|\rho_{ij}|}. \qquad (3.20)$$

The second condition in (3.14) ensures this for any k, $r \geq 0$. Thus, under the conditions (3.14), (3.12) yields

$$\rho(M) \leq \|M\|_\infty < 1. \qquad (3.21)$$

□

Remark. Note that the conditions (3.14) are the same as the conditions (2.58) for uniform solvability of implicit θ-schemes with respect to the maximum norm (Proposition 13).

However, even if the weighted Jacobi method converges, it turns out that its performance (like that of most classical iterative methods) when applied to systems arising from the finite-difference discretisation of PDEs is not satisfactory: If one plots the size of the error $\mathbf{E}_m$ (measured in some norm) against the number of iteration steps (as is done, e.g., in [Briggs, 1987, Chapter 2]), one will in general observe a fast decrease of the error during the first few steps, but as the number of iterations grows, the error decays more and more slowly, and an acceptable accuracy can in general only be reached after a very large number of iteration steps.

This slowing down convergence is due to the so-called *smoothing property* of the iterative method. This effect is best seen in the Fourier mode decomposition of the error. The (complex) *Fourier modes* which are distinguishable on the grid points of the mesh $\Omega^{\mathbf{x}}_{R,h}$ are represented by the set of vectors $\hat{\mathbf{W}}_{\boldsymbol{k}} \in \mathbb{R}^{(2M+1)^n}$, $\boldsymbol{k} \in \{-(M-1), -(M-2), \ldots, -1, 0, 1, \ldots, M\}^n$, with components

$$(\hat{\mathbf{W}}_{\boldsymbol{k}})_{\boldsymbol{\alpha}} = \mathrm{e}^{i\frac{\pi}{R}\boldsymbol{k}\cdot\mathbf{x}_{\boldsymbol{\alpha}}} \qquad (3.22)$$

(see, e.g., [Morton and Mayers, 1994, Section 5.6]). The n-component vector $\boldsymbol{k}$ can be understood as a *wavenumber* vector which describes how oscillatory the mode is; $\boldsymbol{k} = \mathbf{0}$ corresponds to a very smooth (in fact, constant) mode, whereas the mode with $\boldsymbol{k} = (M, M, \ldots, M)$ is the most oscillatory one on the mesh. In the following analysis, we will distinguish between *smooth modes* and *high-frequency modes* on the respective grid; here, we understand by a smooth mode a Fourier

mode where every single wave number component is less than $\frac{M}{2}$ in modulus (cf. [Briggs, 1987, Chapter 2], [Hackbusch, 1993, Section 10.1.1]).

For an arbitrary initial guess, the initial error $\mathbf{E}_0$ will in general contain components of all Fourier modes. It turns out, however, that after some iterations of the weighted Jacobi method, the error $\mathbf{E}_m$ becomes increasingly smoother. High-frequency error components are quickly eliminated by the weighted Jacobi method, whereas smooth error components are damped more slowly (the exact damping factor distribution depends on the weighting factor ω, see, e.g., [Briggs, 1987, Chapter 2], and cf. the last part of Section 3.2.1). Hence, one initially observes a fast decrease of the error when the oscillatory components are eliminated, but once the remaining error consists mainly of smooth components, the convergence slows down as smooth modes are only slowly reduced by the weighted Jacobi method.

3.2 The multigrid method

3.2.1 Basic elements of multigrid

Following [Briggs, 1987], we present in this section the basic ideas leading to the multigrid method. The key observation is that *a smooth Fourier mode on the grid $\Omega^{\mathbf{x}}_{R,h}$, when restricted to a coarser grid, appears more oscillatory on the coarse grid.* This phenomenon known as *aliasing* suggests the following strategy in order to accelerate the convergence of the weighted Jacobi method:

- Apply the weighted Jacobi method to the problem $A\mathbf{W} = \mathbf{RHS}$ on the grid $\Omega^{\mathbf{x}}_{R,h}$ until the error does not decrease significantly any more. This yields the approximation $\mathbf{W}_m$ to $\mathbf{W}$. Its error $\mathbf{E}_m$ cannot be calculated unless the exact solution $\mathbf{W}$ is known; we can, however, calculate its residual $\mathbf{R}_m$ from the known data.

- Restrict the residual $\mathbf{R}_m$ of $\mathbf{W}_m$ to a coarser mesh $\Omega^{\mathbf{x}}_{R,\tilde{h}}$ with $\tilde{h} > h$ and apply the weighted Jacobi method to the equation $A\,\mathbf{E}_m = \mathbf{R}_m$ (see (3.6)) *on the coarse grid* (with initial guess, e.g., zero) in order to obtain an estimate of the error $\mathbf{E}_m$. The error components which are smooth on $\Omega^{\mathbf{x}}_{R,h}$ and which could not be eliminated well any more on the fine grid now correspond to more oscillatory modes on $\Omega^{\mathbf{x}}_{R,\tilde{h}}$ and can thus be effectively removed by the weighted Jacobi method on the coarse grid.

- Interpolate the obtained approximation of $\mathbf{E}_m$ back to the fine grid $\Omega^{\mathbf{x}}_{R,h}$ and add it to $\mathbf{W}_m$ to obtain a better approximation $\tilde{\mathbf{W}}_m$ to $\mathbf{W}$ (note that $\mathbf{W} = \mathbf{W}_m + \mathbf{E}_m$ by (3.4)).

- One may further improve the approximation by applying some more iterations of the weighted Jacobi method to the problem $A\mathbf{W} = \mathbf{RHS}$ on the fine grid with initial guess $\tilde{\mathbf{W}}_m$.

Before formalising this procedure, we observe several details (see also [Hackbusch, 1985, Sections 3.4–3.7]). First, it is practical to use a subset of the fine-grid points as coarse grid. The most convenient and most popular choice is $\Omega^{\mathbf{x}}_{R,\tilde{h}} := \Omega^{\mathbf{x}}_{R,2h}$, i.e. one uses the grid of twice the mesh width of the original grid as coarse grid; we will study the relation between these two meshes in more detail in Section 3.2.3. Second, we need to find appropriate *intergrid transfer operations*, i.e. methods to restrict a vector given on the fine grid to the coarse grid, and to interpolate a vector given on the coarse grid to the fine grid. Again, this issue is addressed in Section 3.2.3. Finally, it has to be defined how the original linear system with system matrix A can be represented on the

coarse grid. For the discretisation matrix $\tilde{A}_h^{(\mathrm{ii})}$ of a partial differential operator in space, an appropriate possibility is to use the matrix $\tilde{A}_{2h}^{(\mathrm{ii})}$ on the coarse grid, i.e. the matrix that corresponds to the discretisation of the differential operator directly on the coarse grid. Consequently, the system matrix for the problem (3.2) on the grid $\Omega_{R,h}^{\mathbf{x}}$ will be defined by

$$B_{1,h} := \frac{1}{k}\, I_{(2M-1)^n} + \frac{1}{2} A_h^{(\mathrm{ii})} \quad \text{where } M = \frac{R}{h}. \tag{3.23}$$

It will be useful to introduce a similar mesh width subscript for vectors, indicating which grid they are defined on.

We are now ready to express the above idea as a formal algorithm, called *two-grid iteration* by [Hackbusch, 1985], in Algorithm 1. The key element is the *coarse-grid correction* in steps 2–5.

Algorithm 1 Two-grid iteration for solving $A_h \mathbf{W}_h = \mathbf{RHS}_h$

1: *Pre-smoothing*: Apply the weighted Jacobi method $\nu_1 \geq 0$ times to the equation
$A_h \mathbf{W}_h = \mathbf{RHS}_h$
on $\Omega_{R,h}^{\mathbf{x}}$ with a given initial guess $\mathbf{W}_{0,h}$ to obtain an approximation $\mathbf{W}_{\nu_1,h}$.
2: Compute the residual $\mathbf{R}_h = \mathbf{RHS}_h - A_h \mathbf{W}_{\nu_1,h}$, and restrict it to $\Omega_{R,2h}^{\mathbf{x}}$ to obtain $\mathbf{R}_{2h}$.
3: Apply the weighted Jacobi method $\kappa > 0$ times to the equation
$A_{2h} \mathbf{E}_{2h} = \mathbf{R}_{2h}$
on $\Omega_{R,2h}^{\mathbf{x}}$ with initial guess $\mathbf{0}$ to obtain an approximation $\mathbf{E}_{\kappa,2h}$.
4: Interpolate $\mathbf{E}_{\kappa,2h}$ to $\Omega_{R,h}^{\mathbf{x}}$.
5: Correct the approximation $\mathbf{W}_{\nu_1,h}$ on $\Omega_{R,h}^{\mathbf{x}}$ with the interpolated error estimate $\mathbf{E}_{\kappa,h}$.
6: *Post-smoothing*: Apply the weighted Jacobi method $\nu_2 \geq 0$ times to the equation
$A_h \mathbf{W}_h = \mathbf{RHS}_h$
on $\Omega_{R,h}^{\mathbf{x}}$ with $\mathbf{W}_{\nu_1,h} + \mathbf{E}_{\kappa,h}$ as initial guess.

This is combined with a smoothing operation performed before (pre-smoothing) and/or after (post-smoothing) the coarse-grid correction step; either ν_1 or ν_2 is required to be non-zero.

The role of the smoothing operation in the context of the multigrid method is to dampen primarily the *high-frequency* modes of the error, the smooth components being taken care of by the coarse-grid correction step. Thus, the weighting factor in the weighted Jacobi method is optimally chosen so as to maximise the damping factors in the high-frequency part of the spectrum. Following ideas presented, e.g., in [Hackbusch, 1985, Section 2.6.3], one can show that the optimal value of ω for the n-dimensional Laplace problem is given by

$$\omega_{\mathrm{opt}} = \frac{2n}{2n+1}. \tag{3.24}$$

Note that this is the optimal weighting factor for the weighted Jacobi method used as a smoothing operation within a multigrid method, not for the weighted Jacobi method used on its own.

In step 1 of Algorithm 1, the initial guess $\mathbf{W}_{0,h}$ is not specified. Again, the coarse grid can be used to find an informed initial guess: before starting the smoothing operation on the fine grid, we can apply the weighted Jacobi method to the equation $A_{2h}\mathbf{W}_{2h} = \mathbf{RHS}_{2h}$ *on the coarse grid*, and then interpolate the obtained approximation of $\mathbf{W}_{2h}$ in order to use it as initial guess on the fine grid. This strategy is called *nested iteration*. The trick of this idea is that applying the weighted

Jacobi method on the coarse grid is much less computationally expensive than on the fine grid (see also [Briggs, 1987, Chapter 4], [Hackbusch, 1985, Section 4.3]): as we have seen above, the effort of one weighted Jacobi iteration step is proportional to the number of degrees of freedom, i.e. $(2M-1)^n$ on $\Omega^{\mathbf{x}}_{R,h}$, and $(M-1)^n$ on $\Omega^{\mathbf{x}}_{R,2h}$. Hence, the effort of one weighted Jacobi step on the coarse grid is roughly $\frac{1}{2^n}$ times the effort of one weighted Jacobi step on the fine grid, and nested iteration can thus be added to the two-grid iteration algorithm at a minimum extra effort.

3.2.2 Multigrid algorithms

The strategies discussed so far involved only two meshes and served to illustrate the key elements of the multigrid method. We now arrive at the multigrid method by simply *recursively applying the two-grid method on a sequence of successively coarser meshes*, where the coarsest grid is advantageously chosen such that a direct solution of the linear system is possible there. In our case, we have assumed the mesh parameter M to be a power of two in Section 2.1, $M = 2^m$ for some $m \geq 0$, such that by successively doubling the mesh width, we obtain a sequence of $m+1$ meshes where the coarsest one is characterised by $M = 1$ and hence involves exactly one interior grid point. The linear system on the coarsest grid is thus a scalar equation which can easily be solved explicitly.

For the sake of clarity, let us define several subfunctions to be used in the multigrid algorithms:

$$\text{jacobi}\,(M, \mathbf{RHS}_h, \mathbf{W}_{0,h}, \omega, \nu) \tag{3.25}$$

returns the result of ν iterations of the weighted Jacobi method with weighting factor ω and initial guess $\mathbf{W}_{0,h}$ for the system $A_h\mathbf{W}_h = \mathbf{RHS}_h$ on the grid with parameter M;

$$\mathrm{R}\,(\mathbf{W}, M) \quad \text{and} \quad \mathrm{P}\,(\mathbf{W}, M) \tag{3.26}$$

denote, respectively, the restriction of the vector $\mathbf{W}$ from the grid with parameter $2M$ to the one with parameter M, and the prolongation or interpolation of the vector $\mathbf{W}$ from the grid with parameter $\frac{M}{2}$ to the one with parameter M.

We are now ready to state the first multigrid algorithm: recursive application of the two-grid iteration (Algorithm 1) yields the so-called *multigrid μ-cycle scheme* (see also [Briggs, 1987, Chapter 3], [Hackbusch, 1985, Algorithm (4.1.1)]) whose recursive function definition is given in Algorithm 2. While the coarsest grid is not reached yet (i.e. $M > 1$), pre-smoothing is performed (line 5), the residual is calculated (line 6), and the multigrid μ-cycle is called μ times on the next coarser grid (line 9) for the residual equation $A_{2h}\mathbf{E}_{2h} = \mathbf{R}_{2h}$ where the right-hand side is the residual $\mathbf{R}_h$, restricted to the coarser grid, and a zero initial guess is used (initialised in line 7). The obtained error estimate $\mathbf{E}_{2h}$ is interpolated back to the finer grid and used to correct the approximation W_h (line 11) before post-smoothing is performed (line 12). On the coarsest grid, in contrast, the linear system $A_h\mathbf{W}_h = \mathbf{RHS}_h$ is solved exactly (formally expressed in line 3).

The cycling parameter μ defines how many times the algorithm is called on the next coarser grid (see line 8 of Algorithm 2). $\mu = 1$ yields the so-called *multigrid V-cycle*, and $\mu = 2$ the *multigrid W-cycle*. These cycles are named after the patterns formed by the schedule of visited grids, which are illustrated in Figure 3.1. Cycling parameters higher than 2 are unproductive in practice according to [Hackbusch, 1985, Section 4.2].

The V-cycle scheme supplemented by the nested iteration strategy yields the so-called *full multigrid V-cycle scheme* which is particularly effective. The recursive definition of this method is given in Algorithm 3. This algorithm involves the cycling parameter ν indicating how many

Algorithm 2 Multigrid μ-cycle for solving $A_h \mathbf{W}_h = \mathbf{RHS}_h$

function $\mathbf{W}_h = \mathrm{MG}\,(M, \mathbf{RHS}_h, \mathbf{W}_{0,h}, \omega, \nu_1, \nu_2, \mu)$
Input: parameter of the finest grid $M = 2^m$, right-hand side vector $\mathbf{RHS}_h$, initial guess $\mathbf{W}_{0,h}$, weighting parameter ω, smoothing parameters ν_1, ν_2, cycling parameter μ
Output: approximation $\mathbf{W}_h$ on the finest grid
if $M = 1$ **then**
  $\mathbf{W}_h \leftarrow A_h^{-1}\,\mathbf{RHS}_h$
else
  $\mathbf{W}_h \leftarrow \mathrm{jacobi}\,(M, \mathbf{RHS}_h, \mathbf{W}_{0,h}, \omega, \nu_1)$
  $\mathbf{R}_h \leftarrow \mathbf{RHS}_h - A_h \mathbf{W}_h$
  $\mathbf{E}_{2h} \leftarrow \mathbf{0}$
  for $i = 1 : \mu$ **do**
    $\mathbf{E}_{2h} \leftarrow \mathrm{MG}\,(\frac{M}{2}, \mathrm{R}\,(\mathbf{R}_h, \frac{M}{2}), \mathbf{E}_{2h}, \omega, \nu_1, \nu_2, \mu)$
  end for
  $\mathbf{W}_h \leftarrow \mathbf{W}_h + \mathrm{P}\,(\mathbf{E}_{2h}, M)$
  $\mathbf{W}_h \leftarrow \mathrm{jacobi}\,(M, \mathbf{RHS}_h, \mathbf{W}_h, \omega, \nu_2)$
end if
return $\mathbf{W}_h$

V-cycles are to be performed on each grid once an initial guess has been established by nested iteration. Figure 3.2 illustrates the schedule of visited grids in the case $\nu = 1$.

3.2.3 Intergrid transfer

In our listings of the multigrid μ-cycle and full multigrid V-cycle schemes, we assumed that appropriate restriction and interpolation methods had already been defined (cf. (3.26)). These intergrid transfer operations are the subject of the current section.

Let us first take a closer look at the relation between the meshes $\Omega^{\mathbf{x}}_{R,h}$ and $\Omega^{\mathbf{x}}_{R,2h}$. For the two-dimensional case, this is illustrated in Figure 3.3. It is hence easily seen that the coarse grid is made up of those fine-grid points $\mathbf{x}_{\boldsymbol{\alpha}}$ whose multiindex $\boldsymbol{\alpha}$ consists of even components only;

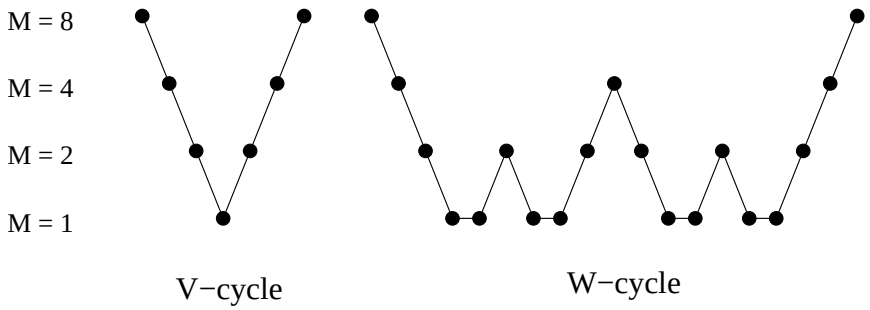

Figure 3.1: Schedule of grids where the weighted Jacobi method is performed in the case $M = 8$ for $\mu = 1$ (V-cycle) and $\mu = 2$ (W-cycle).

Algorithm 3 Full multigrid V-cycle for solving $A_h \mathbf{W}_h = \mathbf{RHS}_h$

function $\mathbf{W}_h = \mathrm{FMG}\,(M, \mathbf{RHS}_h, \omega, \nu_1, \nu_2, \nu)$

Input: parameter of the finest grid $M = 2^m$, right-hand side vector $\mathbf{RHS}_h$, weighting factor ω, smoothing parameters ν_1 and ν_2, cycling parameter ν

Output: approximation $\mathbf{W}_h$ on the finest grid

$\mathbf{W}_h \leftarrow \mathbf{0}$

if $M = 1$ **then**

  go to step 10

else

  $\mathbf{R}_h \leftarrow \mathbf{RHS}_h - A_h \mathbf{W}_h$ % in fact, $\mathbf{R}_h = \mathbf{RHS}_h$ here as $\mathbf{W}_h = \mathbf{0}$ at this stage

  $\mathbf{W}_{2h} \leftarrow \mathrm{FMG}\,(\frac{M}{2}, \mathrm{R}\,(\mathbf{R}_h, \frac{M}{2}), \omega, \nu_1, \nu_2, \nu)$

end if

$\mathbf{W}_h \leftarrow \mathbf{W}_h + \mathrm{P}\,(\mathbf{W}_{2h}, M)$

for $i = 1 : \nu$ **do**

  $\mathbf{W}_h \leftarrow \mathrm{MG}\,(M, \mathbf{RHS}_h, \mathbf{W}_h, \omega, \nu_1, \nu_2, 1)$

end for

return $\mathbf{W}_h$

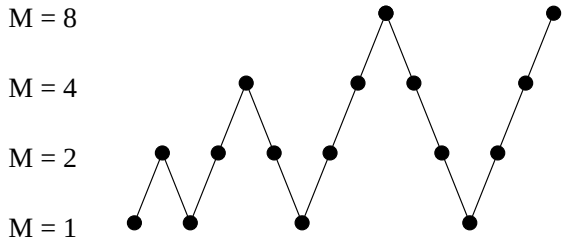

Figure 3.2: Schedule of grids where the weighted Jacobi method is performed in the full multigrid V-cycle in the case $M = 8$, $\nu = 1$.

formally,

$$(\mathbf{x}_{\boldsymbol{\alpha}})_{2h} = (\mathbf{x}_{2\boldsymbol{\alpha}})_h \quad \text{for all } \boldsymbol{\alpha} \in \left\{-\frac{M}{2}, \ldots, \frac{M}{2}\right\}^n. \tag{3.27}$$

A very simple restriction operation, called *injection*, consists in assigning each coarse-grid point the value of the corresponding fine-grid point. However, injection is often avoided in practice as it does not always satisfy the heuristic requirement stated in [Hackbusch, 1985, Note 3.5.1]. More sophisticated restriction operators are preferred which use the information contained in fine-grid points which are not part of the coarse grid as well (see also [Hackbusch, 1985, Section 3.8.5]); we will present a popular choice below.

For the prolongation operator, piecewise polynomial interpolation is an obvious choice. For our model problem, we will use *n-linear interpolation*, i.e. we assign each fine-grid point the evaluation at this point of the unique n-linear function interpolating all surrounding coarse-grid values (cf. [Briggs, 1987, Chapter 3], [Hackbusch, 1985, Section 3.4.2] for the one and two-dimensional cases). This is equivalent to assigning each fine-grid point the arithmetic mean of the values at the coarse-grid points *nearest* this fine-grid point. The number of nearest coarse-grid neighbours depends on the

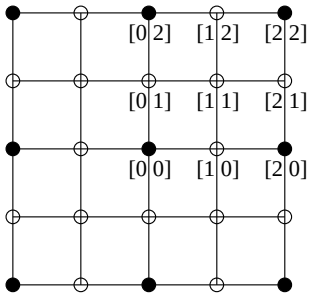

Figure 3.3: The grids $\Omega^{\mathbf{x}}_{R,R}$ (filled dots) and $\Omega^{\mathbf{x}}_{R,R/2}$ (all dots) in two dimensions. Selected grid points are labelled with their fine-grid multiindex.

position of the fine-grid point within the coarse grid: let $\boldsymbol{\alpha}$ be the multiindex of the fine-grid point. Then each odd component of $\boldsymbol{\alpha}$ corresponds to a direction in which the fine-grid point is "shifted" with respect to the main axes of the coarse grid. The nearest coarse-grid points are reached by altering each odd component of $\boldsymbol{\alpha}$ by $+1$ or -1. If p components of $\boldsymbol{\alpha}$ are odd, there are 2^p such combinations, hence $\mathbf{x}_{\boldsymbol{\alpha}}$ has 2^p nearest coarse-grid neighbours, with $0 \leq p \leq n$. In Figure 3.3, the different types of fine-grid points in two dimensions can be visualised: the fine-grid index $\boldsymbol{\alpha} = [0, 0]$, for instance, has only even components and the corresponding fine-grid point hence has one nearest coarse-grid neighbour, i.e. $\mathbf{x}_{\boldsymbol{\alpha}}$ itself. The fine-grid point with index $\boldsymbol{\alpha} = [0, 1]$, in contrast, lies on an "edge" between two coarse-grid points, whereas the fine-grid point with index $\boldsymbol{\alpha} = [1, 1]$ lies on a "face" of the coarse grid and has four nearest coarse-grid neighbours.

Conversely, we can determine the weight with which the value at each coarse-grid point enters into the values at the surrounding fine-grid points: given a coarse-grid point of coarse-grid index $\boldsymbol{\alpha}$, this point corresponds to the fine-grid point with index $2\boldsymbol{\alpha}$, and the value at this point is retained, i.e. it has weight 1 for the value at the fine-grid point of index $2\boldsymbol{\alpha}$. For all fine-grid points with index $2\boldsymbol{\alpha} \pm \mathbf{e}_i$ for some i, the weight of the value at the point with coarse-grid index $\boldsymbol{\alpha}$ is $\frac{1}{2}$; for $2\boldsymbol{\alpha} \pm \mathbf{e}_i \pm \mathbf{e}_j$ with $i \neq j$, the weight is $\frac{1}{4}$, and so on; finally, for $2\boldsymbol{\alpha} + \sum_{i=1}^{n} (\pm \mathbf{e}_i)$, the weight is $\frac{1}{2^n}$. This way, the matrix $P \in \mathbb{R}^{(2M+1)^n \times (M+1)^n}$ representing the prolongation mapping can be assembled; in the rows and columns of this matrix, the fine-grid and coarse-grid indices are listed, respectively.

By taking the adjoint of a polynomial prolongation operator, one obtains a restriction operator. [Briggs, 1987] and [Hackbusch, 1985] use the so-called *full weighting* restriction operator given by the adjoint of n-linear interpolation, multiplied by the normalising factor 2^{-n} (which results from the different scaling of the corresponding scalar products, cf. [Hackbusch, 1985, Section 3.5]), or in matrix notation, $R = 2^{-n} P^T$. Unlike injection, full weighting restriction also takes into account values at fine-grid points which are not part of the coarse grid.

A detailed discussion of different prolongation and restriction operators and criteria for an appropriate choice can be found in [Hackbusch, 1985, Sections 3.4 and 3.5]; for the discretisation of a second-order PDE as in our case, the combination of n-linear interpolation and full weighting restriction is approved, cf. [Hackbusch, 1985, Example 3.5.2].

Chapter 4

Implementation of a numerical Black-Scholes solver

We aim to solve the Crank-Nicolson and the implicit Euler schemes presented in Section 2.2.1 for the truncated transformed Black-Scholes problem (1.57)–(1.59) with initial function $\tilde{w}_0(\mathbf{x}) = V_0(\mathrm{e}^{x_1}, \ldots, \mathrm{e}^{x_n})$ for some continuous pay-off function V_0, and boundary function $\tilde{g}$ for which we found appropriate choices to be $\tilde{g}_1(\mathbf{x}, \tau) = \tilde{w}_0(\mathbf{x})$ or $\tilde{g}_2(\mathbf{x}, \tau) = \mathrm{e}^{-r\tau}\, V_0\big(\mathrm{e}^{x_1 + r\tau}, \ldots, \mathrm{e}^{x_n + r\tau}\big)$ in Section 1.4.4. Regarding (2.22)–(2.24), the θ-method defines the approximation vectors $\mathbf{W}^\ell$ for this problem as

$$\mathbf{W}^0 = \left(\tilde{w}_0(\mathbf{x}_{\boldsymbol{\alpha}})\right)_{\boldsymbol{\alpha}}, \tag{4.1}$$

$$\mathbf{W}^{\ell,\mathrm{bd}} = \tilde{\mathbf{g}}^\ell, \qquad \ell = 1, \ldots, K, \tag{4.2}$$

$$\begin{aligned}\Big(I + k\,\theta A_h^{(\mathrm{ii})}\Big)\mathbf{W}^{\ell+1}(\texttt{intind}) = {}& \Big(I - k\,(1-\theta)A_h^{(\mathrm{ii})}\Big)\mathbf{W}^{\ell}(\texttt{intind}) \\ & - k\,(1-\theta)A_h^{(\mathrm{ib})}\left(\tilde{\mathbf{g}}^{\ell}(\texttt{bdind}) + \tilde{\mathbf{g}}^{\ell+1}(\texttt{bdind})\right), \quad \ell = 0, \ldots, K-1,\end{aligned} \tag{4.3}$$

where the matrix A_h is given by

$$(A_h)_{\boldsymbol{\alpha},\boldsymbol{\beta}} = \begin{cases} r + \sum_{i=1}^n \frac{\sigma_i^2}{h^2} & \text{if } \boldsymbol{\beta} = \boldsymbol{\alpha} \text{ and } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{int}}, \\ -\frac{\sigma_i^2}{2h^2} + \frac{\sigma_i^2}{4h} - \frac{r}{2h} & \text{if } \boldsymbol{\beta} = \boldsymbol{\alpha} + \mathbf{e}_i,\ i = 1, \ldots, n, \text{ and } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{int}}, \\ -\frac{\sigma_i^2}{2h^2} - \frac{\sigma_i^2}{4h} + \frac{r}{2h} & \text{if } \boldsymbol{\beta} = \boldsymbol{\alpha} - \mathbf{e}_i,\ i = 1, \ldots, n, \text{ and } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{int}}, \\ -\frac{\sigma_i \sigma_j \rho_{ij}}{4h^2} & \text{if } \boldsymbol{\beta} = \boldsymbol{\alpha} + \mathbf{e}_i + \mathbf{e}_j \text{ or } \boldsymbol{\beta} = \boldsymbol{\alpha} - \mathbf{e}_i - \mathbf{e}_j,\ i \neq j, \text{ and } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{int}}, \\ \frac{\sigma_i \sigma_j \rho_{ij}}{4h^2} & \text{if } \boldsymbol{\beta} = \boldsymbol{\alpha} + \mathbf{e}_i - \mathbf{e}_j \text{ or } \boldsymbol{\beta} = \boldsymbol{\alpha} - \mathbf{e}_i + \mathbf{e}_j,\ i \neq j, \text{ and } \boldsymbol{\alpha} \in \boldsymbol{\alpha}_{\mathrm{int}}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

Our objective is to implement this scheme for $\theta = \frac{1}{2}$ (Crank-Nicolson) and $\theta = 1$ (implicit Euler) for an *arbitrary* number n of assets. In particular, for each θ, we aim to

- create a single implementation simultaneously applicable for any dimension n, and
- to take advantage of the fact that MATLAB is optimised for vector and matrix operations by *vectorising* the code, i.e. by replacing `for` and `while` loops by appropriate matrix operations whenever possible (cf. [MATLAB 7.7.0 Help, 2008]).

We will use the full multigrid V-cycle scheme (see Algorithm 3) to iteratively solve the sparse system (4.3).

4.1 Data structure

We have already specified the output arguments $\mathbf{W}^\ell$, $\ell = 0, \ldots, K$ to be $(2M+1)^n$-component vectors. Another possibility would be to create them as n-dimensional arrays where each entry $W^\ell_{\boldsymbol{\alpha}}$ is stored at the position which corresponds to the spatial location of the grid point $\mathbf{x}_{\boldsymbol{\alpha}}$ within the mesh. This is in principle feasible in MATLAB and may seem an intuitive approach regarding the association between grid points and output components. However, the code is in fact more straightforward if the array is "broken up" and stored as a single long vector, which is also the way MATLAB internally stores n-dimensional arrays (see [MATLAB 7.7.0 Help, 2008]). This is the data format used in the program presented below.

This storage method requires a unique mapping between the multiindex $\boldsymbol{\alpha}$ associated with the mesh and the array index defining the position of the corresponding component $W^\ell_{\boldsymbol{\alpha}}$ within the vector $\mathbf{W}^\ell$, or in other words, an appropriate ordering of the grid points. This ordering scheme has to be compatible with the program feature that the space dimension n is an a-priori arbitrary parameter. Furthermore, we have to be able to identify neighbouring grid points, interior and boundary grid points.

4.1.1 Ordering of the grid points

We have seen in the previous section that we need to define an appropriate ordering of the grid points of the spatial mesh. In the program presented below, we will always use *lexicographical ordering* of the grid points by their multiindex $\boldsymbol{\alpha}$, i.e. when we compare two indices, we consider their components in the order $\alpha_1, \ldots, \alpha_n$ and decide which index is "larger" by comparing the first component in which they differ. Table 4.1 illustrates this ordering in the case of three dimensions. It is MATLAB standard to start array indexing from 1.

array index	multi-index			array index	multi-index			array index	multi-index		
1	−1	−1	−1	10	0	−1	−1	19	1	−1	−1
2	−1	−1	0	11	0	−1	0	20	1	−1	0
3	−1	−1	1	12	0	−1	1	21	1	−1	1
4	−1	0	−1	13	0	0	−1	22	1	0	−1
5	−1	0	0	14	0	0	0	23	1	0	0
6	−1	0	1	15	0	0	1	24	1	0	1
7	−1	1	−1	16	0	1	−1	25	1	1	−1
8	−1	1	0	17	0	1	0	26	1	1	0
9	−1	1	1	18	0	1	1	27	1	1	1

Table 4.1: Ordering of grid points in a mesh with $n = 3$, $M = 1$.

Geometrically, this means that we move within the three-dimensional domain along "piles" of grid points in the positive z direction, proceeding to the next pile in the y direction until we reach

the "right" y direction boundary where we return to the "left" y direction boundary after a step in the positive x direction. Figure 4.1 illustrates this. An analogous "pile" interpretation holds in all dimensions.

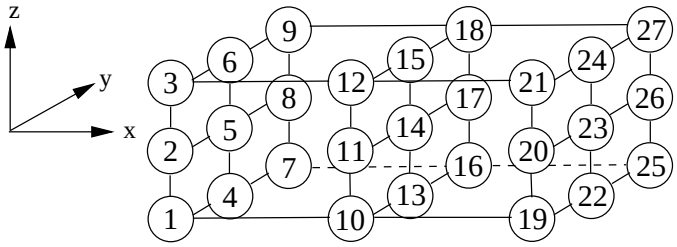

Figure 4.1: Illustration of the ordering of grid points in a mesh with $n = 3$, $M = 1$.

Note that if we add $(M, M, \ldots, M)$ to each multiindex, we do not alter their lexicographical order. The components of the shifted multiindices are all between 0 and $2M$ and can thus be interpreted as the digits of n-digit numbers in the base $(2M + 1)$ numeral system. Lexicographical ordering then corresponds to ordering these numbers by increasing size. With this interpretation, it is easy to see that arithmetically, the array index $\hat{\imath} \in \{1, \ldots, (2M + 1)^n\}$ corresponding to the multiindex $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ is given by

$$\hat{\imath}\,(\boldsymbol{\alpha}) = 1 + \sum_{i=1}^{n} (\alpha_i + M) \cdot (2M + 1)^{n-i}\,. \tag{4.5}$$

Furthermore, we find that the array indices of neighbouring grid points differ by a fixed increment depending on the dimension in which the grid points adjoin: namely, the relation between the array indices of a given grid point $\mathbf{x}_{\boldsymbol{\alpha}}$ and its right and left neighbours in the i-th direction $\mathbf{x}_{\boldsymbol{\alpha}+\mathbf{e}_i}$ and $\mathbf{x}_{\boldsymbol{\alpha}-\mathbf{e}_i}$ (provided that they exist) is

multiindex	array index
$\boldsymbol{\alpha}$	$\hat{\imath}\,(\boldsymbol{\alpha})$
$\boldsymbol{\alpha} + \mathbf{e}_i$	$\hat{\imath}\,(\boldsymbol{\alpha}) + (2M + 1)^{n-i}$
$\boldsymbol{\alpha} - \mathbf{e}_i$	$\hat{\imath}\,(\boldsymbol{\alpha}) - (2M + 1)^{n-i}$

(4.6)

4.1.2 Identifying interior and boundary grid points

As interior and boundary grid points have to be treated differently, we have to identify which positions within the vectors $\mathbf{W}^\ell$ correspond to which type of grid points. A geometrical consideration shows that the array indices of the boundary grid points can be systematically described as follows:

"left" boundary in the i-th dimension:

$$d \cdot (2M + 1)^{n-i+1} + 1 \qquad : \; d \cdot (2M + 1)^{n-i+1} + (2M + 1)^{n-i}, \qquad d = 0, \ldots, (2M + 1)^{i-1} - 1, \tag{4.7}$$

"right" boundary in the i-th dimension:

$$\begin{aligned} & d \cdot (2M+1)^{n-i+1} \\ & \quad + 2M\,(2M+1)^{n-i} + 1 \quad : \quad (d+1) \cdot (2M+1)^{n-i+1}, \qquad\qquad d = 0, \ldots, (2M+1)^{i-1} - 1, \end{aligned} \tag{4.8}$$

where the MATLAB colon notation is used to denote the sequence of unit-spaced values between and including the specified start and end values. The result of these formulae is displayed in Table 4.2 for the case $n = 3$, $M = 1$, and it is in accordance with Figure 4.1.

	$i = 1$	$i = 2$	$i = 3$		
"left" boundary	1 : 9	1 : 3	1 : 1,	4 : 4,	7 : 7,
		10 : 12	10 : 10,	13 : 13,	16 : 16,
		19 : 21	19 : 19,	22 : 22,	25 : 25
"right" boundary	19 : 27	7 : 9	3 : 3,	6 : 6,	9 : 9,
		16 : 18	12 : 12,	15 : 15,	18 : 18,
		25 : 27	21 : 21,	24 : 24,	27 : 27

Table 4.2: Array indices of the boundary grid points in a mesh with $n = 3$, $M = 1$.

Formulae (4.7) and (4.8) thus allow us to determine the set of array indices of all boundary grid points. We will now demonstrate how these formulae can be turned into a fully vectorised algorithm. First, note that for each dimension i, we can write the set of "left" boundary indices (4.7) in matrix form as

$$(2M+1)^{n-i+1} \cdot \begin{pmatrix} 0 & 0 & \ldots & 0 \\ 1 & 1 & \ldots & 1 \\ 2 & 2 & \ldots & 2 \\ \vdots & \vdots & \vdots & \vdots \\ d_{\max} & d_{\max} & \ldots & d_{\max} \end{pmatrix} + \begin{pmatrix} 1 & 2 & \ldots & (2M+1)^{n-i} \\ 1 & 2 & \ldots & (2M+1)^{n-i} \\ 1 & 2 & \ldots & (2M+1)^{n-i} \\ \ldots & \ldots & \ldots & \ldots \\ 1 & 2 & \ldots & (2M+1)^{n-i} \end{pmatrix} \tag{4.9}$$

where $d_{\max} := (2M+1)^{i-1} - 1$, or writing the matrix components row-wise into $(2M+1)^{n-1}$-com-

ponent column vectors, as

$$(2M+1)^{n-i+1} \cdot \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 2 \\ 2 \\ \vdots \\ 2 \\ \vdots \\ d_{\max} \\ d_{\max} \\ \vdots \\ d_{\max} \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \\ \vdots \\ (2M+1)^{n-i} \\ 1 \\ 2 \\ \vdots \\ (2M+1)^{n-i} \\ 1 \\ 2 \\ \vdots \\ (2M+1)^{n-i} \\ \vdots \\ 1 \\ 2 \\ \vdots \\ (2M+1)^{n-i} \end{pmatrix}. \tag{4.10}$$

Analogously, the set of "right" boundary indices is given by (4.10) with $2M\,(2M+1)^{n-i}$ added to each component. We now horizontally concatenate the vectors (4.10) obtained for each dimension i. For our illustration case $n = 3$, $M = 1$, this becomes

$$\text{"left" boundary:} \quad \underbrace{\begin{pmatrix} 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \end{pmatrix}}_{=:M_1} .* \underbrace{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \\ 0 & 1 & 3 \\ 0 & 1 & 4 \\ 0 & 1 & 5 \\ 0 & 2 & 6 \\ 0 & 2 & 7 \\ 0 & 2 & 8 \end{pmatrix}}_{=:M_2} + \underbrace{\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ 3 & 3 & 1 \\ 4 & 1 & 1 \\ 5 & 2 & 1 \\ 6 & 3 & 1 \\ 7 & 1 & 1 \\ 8 & 2 & 1 \\ 9 & 3 & 1 \end{pmatrix}}_{=:M_3}, \tag{4.11}$$

$$\text{"right" boundary:} \quad \overbrace{\begin{pmatrix} 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \\ 3^3 & 3^2 & 3^1 \end{pmatrix}}^{=:M_1} .* \overbrace{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \\ 0 & 1 & 3 \\ 0 & 1 & 4 \\ 0 & 1 & 5 \\ 0 & 2 & 6 \\ 0 & 2 & 7 \\ 0 & 2 & 8 \end{pmatrix}}^{=:M_2} + \overbrace{\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ 3 & 3 & 1 \\ 4 & 1 & 1 \\ 5 & 2 & 1 \\ 6 & 3 & 1 \\ 7 & 1 & 1 \\ 8 & 2 & 1 \\ 9 & 3 & 1 \end{pmatrix}}^{=:M_3} + 2 \cdot \underbrace{\begin{pmatrix} 3^2 & 3^1 & 3^0 \\ 3^2 & 3^1 & 3^0 \\ 3^2 & 3^1 & 3^0 \\ 3^2 & 3^1 & 3^0 \\ 3^2 & 3^1 & 3^0 \\ 3^2 & 3^1 & 3^0 \\ 3^2 & 3^1 & 3^0 \\ 3^2 & 3^1 & 3^0 \\ 3^2 & 3^1 & 3^0 \end{pmatrix}}_{=:M_4} \tag{4.12}$$

(compare with Table 4.2) where we used the MATLAB operator $.*$ denoting component-wise matrix multiplication. We immediately see that $M_1 = (2M+1) \cdot M_4$. Defining in addition the auxiliary matrix

$$M_0 := \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \\ 5 & 5 & 5 \\ 6 & 6 & 6 \\ 7 & 7 & 7 \\ 8 & 8 & 8 \end{pmatrix}, \tag{4.13}$$

we find that M_2 is given by the component-wise integer division of M_0 by M_4. Furthermore, M_3 is obtained as the result of $M_0 - M_4 .* M_2$, component-wise increased by 1. With these relations, we can write the complete "left" boundary matrix as

$$\begin{aligned} M_1 .* M_2 + M_3 &= (2M+1) \cdot M_4 .* M_2 + M_0 - M_4 .* M_2 + \mathbf{1} \\ &= 2M \cdot M_4 .* M_2 + M_0 + \mathbf{1} \end{aligned} \tag{4.14}$$

where $\mathbf{1}$ denotes the $(2M+1)^{n-1} \times n$ matrix containing all ones. The "right" boundary matrix is obtained by adding $2M \cdot M_4$ to (4.14).

For illustrative purposes, we derived Formula (4.14) for the special case $n = 3$, $M = 1$. However, with appropriately defined auxiliary matrices M_0, M_2 and M_4, we ultimately arrive at the same formula for *any* n and M. Namely, using the MATLAB colon notation already explained, we define

$$M_0 := \begin{pmatrix} 0 \ : \ (2M+1)^{n-1} - 1 \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ 0 \ : \ (2M+1)^{n-1} - 1 \end{pmatrix}^{\mathrm{T}} \in \mathbb{R}^{(2M+1)^{n-1} \times n}, \tag{4.15}$$

$$M_4 := \begin{pmatrix} (2M+1)^{n-1} & (2M+1)^{n-2} & \ldots & (2M+1) & 1 \\ \cdots\cdots & \cdots\cdots & \cdots & \cdots\cdots & \cdots \\ (2M+1)^{n-1} & (2M+1)^{n-2} & \ldots & (2M+1) & 1 \end{pmatrix} \in \mathbb{R}^{(2M+1)^{n-1} \times n}, \tag{4.16}$$

and M_2 as the result of component-wise integer division of M_0 by M_4 as before. Matrices like M_0 and M_4 can easily be constructed in MATLAB with the aid of the `repmat` function (see [MATLAB 7.7.0 Help, 2008]): `repmat(A,m,n)` generates a matrix consisting of copies of the array A "tiled" in an $m \times n$ arrangement.

In Algorithm 4, we provide a MATLAB code fragment generating the vectors `intind` and `bdind` of boundary and interior grid point array indices, respectively, for an n-dimensional mesh comprising $2M+1$ grid points in each dimension. For further reference on the MATLAB functions and operators used, see [MATLAB 7.7.0 Help, 2008]. M_0 and M_4 are generated from vectors with the `repmat` function as described above. The ' operator in line 1 denotes transposition. In line 2, we construct the vector `pow` $= \big((2M+1)^{n-1}, (2M+1)^{n-2}, \ldots, (2M+1), 1\big)$ making up M_4 by generating the equally spaced vector $(n-1, n-2, \ldots, 1, 0)$ with the aid of the colon operator with increment -1, and then generating the vector of $(2M+1)$ to the power of all these values with the $.\hat{}$ operator for component-wise powers. In line 4, the integer division of M_0 by M_4 is implemented as the `floor` function (component-wise rounding to the nearest integer towards minus infinity)

Algorithm 4 Finding boundary and interior grid point indices

```
1 M0 = repmat( (0:((2*M+1)^(n-1)-1))', 1,n);
2 pow = (2*M+1).^(n-1:-1:0);
3 M4 = repmat( pow, (2*M+1)^(n-1),1);
4 M2 = floor( M0./M4 );
5 leftbd = 2*M* M4.*M2 + M0 + 1;
6 bdind = reshape( [leftbd; leftbd+2*M*M4], 2*n*(2*M+1)^(n-1),1);
7 intind = setdiff((1:(2*M+1)^n)', bdind);
```

of the component-wise matrix quotient $M_0./M_4$. Line 5 implements Formula (4.14). Note that MATLAB interprets adding the scalar 1 to a matrix as adding 1 to each matrix component. In line 6, we first vertically concatenate the "left" and "right" boundary index matrices `leftbd` and `leftbd`$+2M \cdot M_4$ and then reshape the result as a single column vector. The final output argument `bdind` is a $(2n\,(2M+1)^{n-1})$-component column vector containing all boundary grid point indices in the order

$$\begin{pmatrix} \text{“left” boundary in the first dimension} \\ \text{“right” boundary in the first dimension} \\ \vdots \\ \text{“left” boundary in the } n\text{-th dimension} \\ \text{“right” boundary in the } n\text{-th dimension} \end{pmatrix}. \tag{4.17}$$

Note that grid points which are on the boundary with respect to more than one dimension feature more than once in `bdind`, i.e. `bdind` contains repetitions. The output vector `intind` simply obtained as the set difference of $\{1, \ldots, (2M+1)^n\}$ and `bdind` using the MATLAB `setdiff` function, in contrast, is automatically uniquely valued and sorted; it is a $(2M-1)^n$-component column vector containing the array indices of the interior grid points.

4.2 Evaluating initial and boundary data

We shall need to evaluate the given functions $\tilde{w}_0$ and $\tilde{g}$ at certain sets of grid points. As we aim for a fully vectorised algorithm, these evaluations are to be done simultaneously. To this end, we require the functions $\tilde{w}_0$ and $\tilde{g}$ to be provided in the form of MATLAB function handles (see [MATLAB 7.7.0 Help, 2008]) which can deal with matrix arguments: if we pass a matrix as space argument, we expect the output to be a column vector of evaluations of the respective function at each row of the matrix as space arguments, i.e. if $X \in \mathbb{R}^{N \times n}$ is a "list" of N space arguments $\mathbf{x}_k \in \mathbb{R}^{1 \times n}$ with $\mathbf{x}_k = X(k, 1:n)$, we demand that the function f yields the output $Y \in \mathbb{R}^N$ with $Y(k) = f(\mathbf{x}_k)$. This is a technical issue of entering the initial and boundary data, and MATLAB functionalities facilitate the creation of such function handles; we will present a concrete example in Section 5.4.2.

Suppose the column vector $\mathbf{y}$ contains the array indices of the subset of grid points where we wish to evaluate one of these functions (e.g., $\mathbf{y}$ = `bdind` for $\tilde{g}$, or $\mathbf{y} = (1:(2M+1)^n)^{\mathrm{T}}$ for the entire grid). Provided that a function handle meeting the above requirements is available, all we need to do to obtain the vector of evaluations is to create a matrix that contains in its rows the coordinates of the respective grid points and pass this matrix as space argument to the function handle. The x_i coordinate of the grid point $\mathbf{x}_{\boldsymbol{\alpha}}$ is $h\alpha_i$ (see the definition (2.2) of $\Omega^{\mathbf{x}}_{R,h}$). Hence,

we need an operation to convert array indices into the corresponding multiindices, i.e. the inverse of (4.5). Dividing (4.5) by $(2M+1)^{n-j}$ for any j between 1 and n, we obtain

$$\begin{aligned}\frac{\hat{\imath}-1}{(2M+1)^{n-j}} &= \sum_{i=1}^{n}(\alpha_i+M)\cdot(2M+1)^{j-i} \\ &= \sum_{i=1}^{j}(\alpha_i+M)\cdot(2M+1)^{j-i} + \sum_{i=j+1}^{n}(\alpha_i+M)\cdot(2M+1)^{j-i}. \end{aligned} \tag{4.18}$$

The first partial sum is integer as $j-i\geq 0$, and non-negative. The second partial sum can be estimated like

$$0 \leq \sum_{i=j+1}^{n}(\alpha_i+M)\cdot(2M+1)^{j-i} \leq 2M\sum_{i=j+1}^{n}(2M+1)^{j-i} = 1-\Big(\frac{1}{2M+1}\Big)^{n-j} \leq 1. \tag{4.19}$$

Taking the integral part of (4.18) hence yields

$$\left\lfloor \frac{\hat{\imath}-1}{(2M+1)^{n-j}} \right\rfloor = \sum_{i=1}^{j}(\alpha_i+M)\cdot(2M+1)^{j-i}, \tag{4.20}$$

and we finally obtain the conversion formula,

$$\alpha_j(\,\hat{\imath}\,) = \left\lfloor \frac{\hat{\imath}-1}{(2M+1)^{n-j}} \right\rfloor - (2M+1)\left\lfloor \frac{\hat{\imath}-1}{(2M+1)^{n-j+1}} \right\rfloor - M, \quad j=1,\ldots,n. \tag{4.21}$$

Again, this relation is straightforward if $\boldsymbol{\alpha}+(M,M,\ldots,M)$ is interpreted as the base $(2M+1)$ numeral system representation of the number $\hat{\imath}-1$.

Algorithm 5 presents a MATLAB procedure to simultaneously convert all array indices listed in the column vector $\mathbf{y}$ into the corresponding multiindices: In the first line, each component of $\mathbf{y}$

Algorithm 5 Converting array indices to multiindices

```
1 y = repmat(y-1,1,n);
2 aux = y./repmat( (2*M+1).^(n:-1:1),length(y),1 );
3 alpha = floor((2*M+1)*aux) - (2*M+1)*floor(aux) - M;
```

is decreased by 1, and n copies of this vector are horizontally concatenated. The first column of the resulting matrix is then divided by $(2M+1)^n$, the second by $(2M+1)^{n-1}$ and so on, and finally, the n-column matrix `alpha` containing in its rows the multiindex components corresponding to each array index is calculated according to (4.21) using the floor of the respective quotients.

In order to obtain the vector of function evaluations at the grid points listed in $\mathbf{y}$, the function handle has to be called with the matrix $h\cdot$ `alpha` as space argument.

4.3 The matrices A_h and $B_{1,h}$

The linear system (4.3) to be solved involves submatrices of $A_h \in \mathbb{R}^{(2M+1)^n\times(2M+1)^n}$. This matrix is in general very large, and storing it component-wise would require a vast amount of memory. As

A_h is sparse, however, the memory requirement reduces considerably if only the non-zero elements together with their array indices are stored. MATLAB provides an appropriate sparse matrix data structure along with a number of functions optimised for sparse matrices and thus allows to store and manipulate sparse matrices highly efficiently (see [MATLAB 7.7.0 Help, 2008]). If we take a closer look at the problem, however, we find that we do not even need to generate the matrix A_h explicitly: the right-hand side of the system (4.3) involves $A_h^{(\mathrm{ii})}$ and $A_h^{(\mathrm{ib})}$ merely in the context of matrix-vector products. The same holds true for the system matrix $I + k\,\theta A_h^{(\mathrm{ii})} = kB_{1,h}$ within the multigrid method: in the multigrid algorithms 2 and 3, the system matrix is needed in the context of the weighted Jacobi method for smoothing steps and for the calculation of residuals, and in both cases exclusively in the form of matrix-vector products again (see (3.5) and (3.7)). Hence, it suffices to implement the multiplication of vectors by A_h and $kB_{1,h}$, and this can be done without assembling the full matrices as we will show below.

Using array indexing as explained in Section 4.1.1, the matrix A_h is given by

$$(A_h)_{\hat{\imath}\hat{\jmath}} = \begin{cases} r + \sum_{i=1}^{n} \frac{\sigma_i^2}{h^2} & \text{if } \hat{\imath} \in \texttt{intind} \text{ and } \hat{\jmath} = \hat{\imath} \\ -\frac{\sigma_i^2}{2h^2} + \frac{\sigma_i^2}{4h} - \frac{r}{2h} & \text{if } \hat{\imath} \in \texttt{intind} \text{ and } \hat{\jmath} = \hat{\imath} + (2M+1)^{n-i}, i = 1, \ldots, n, \\ -\frac{\sigma_i^2}{2h^2} - \frac{\sigma_i^2}{4h} + \frac{r}{2h} & \text{if } \hat{\imath} \in \texttt{intind} \text{ and } \hat{\jmath} = \hat{\imath} - (2M+1)^{n-i}, i = 1, \ldots, n, \\ -\frac{\sigma_i \sigma_j \rho_{ij}}{4h^2} & \text{if } \hat{\imath} \in \texttt{intind} \text{ and } \hat{\jmath} = \hat{\imath} + (2M+1)^{n-i} + (2M+1)^{n-j} \\ & \quad \text{or } \hat{\jmath} = \hat{\imath} - (2M+1)^{n-i} - (2M+1)^{n-j},\ i \neq j, \\ \frac{\sigma_i \sigma_j \rho_{ij}}{4h^2} & \text{if } \hat{\imath} \in \texttt{intind} \text{ and } \hat{\jmath} = \hat{\imath} + (2M+1)^{n-i} - (2M+1)^{n-j} \\ & \quad \text{or } \hat{\jmath} = \hat{\imath} - (2M+1)^{n-i} + (2M+1)^{n-j},\ i \neq j, \\ 0 & \text{otherwise,} \end{cases} \tag{4.22}$$

and the entries of $I_{(2M+1)^n} + k\,\theta A_h$ (i.e. $kB_{1,h}$ supplemented by rows and columns corresponding to boundary indices) derive from those of A_h like

$$\Big(I_{(2M+1)^n} + k\,\theta A_h\Big)_{\hat{\imath}\hat{\jmath}} = \delta_{\hat{\imath}\hat{\jmath}} + k\,\theta (A_h)_{\hat{\imath}\hat{\jmath}} \tag{4.23}$$

with the Kronecker delta. We now see that with lexicographical grid point ordering, A_h is *banded* with (maximal) upper and lower band width $(2M+1)^{n-1} + (2M+1)^{n-2}$ (if $n \geq 2$; in the one-dimensional case, the matrix is tridiagonal). Figure 4.2 illustrates the typical sparsity pattern of A_h in several space dimensions. The plots were generated with the MATLAB command `spy` (cf. [MATLAB 7.7.0 Help, 2008]). Empty rows correspond to boundary grid points.

Furthermore, (4.22) reveals that each row of A_h that corresponds to an *interior* index contains the *same* set of (potential) non-zero components, in the sense of actual values and their position with respect to the diagonal. The $2n^2+1$ elements of this set are systematically listed in Table 4.3.

Based on these observations, the multiplication of a vector $\mathbf{W} \in \mathbb{R}^{(2M+1)^n}$ by the matrix A_h can be implemented as follows: write the increments listed in the first column of Table 4.3 into the $(2n^2+1)$-component row vector `increments`, and the corresponding matrix entries into the equally sized vector `vals`. The $\hat{\imath}$-th component of $A_h\mathbf{W}$ is then given by

$$(A_h\mathbf{W})_{\hat{\imath}} = \begin{cases} 0, & \hat{\imath} \in \texttt{bdind}, \\ \sum_{\hat{\jmath}=1}^{(2M+1)^n} (A_h)_{\hat{\imath}\hat{\jmath}}\, W_{\hat{\jmath}} = \sum_{j=1}^{2n^2+1} \texttt{vals}(j)\, W_{\hat{\imath}+\texttt{increments}(j)} & \\ = \mathbf{W}\big((\hat{\imath}, \hat{\imath}, \ldots, \hat{\imath}) + \texttt{increments}\big) \cdot \texttt{vals}^{\mathrm{T}}, & \hat{\imath} \in \texttt{intind}. \end{cases} \tag{4.24}$$

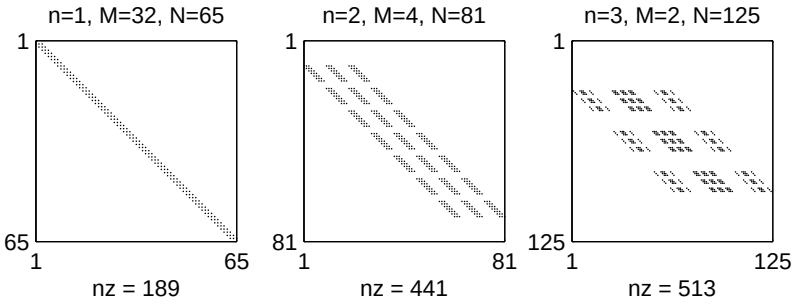

Figure 4.2: Visualisation of the sparsity pattern of the matrix A_h for different space dimensions. nz denotes the number of non-zero entries.

distance from the diagonal		value
0		$r + \sum_{i=1}^{n} \frac{\sigma_i^2}{h^2}$
$(2M+1)^{n-i}$	for all i	$-\frac{\sigma_i^2}{2h^2} + \frac{\sigma_i^2}{4h} - \frac{r}{2h}$
$-(2M+1)^{n-i}$	for all i	$-\frac{\sigma_i^2}{2h^2} - \frac{\sigma_i^2}{4h} + \frac{r}{2h}$
$(2M+1)^{n-i} + (2M+1)^{n-j}$	for all (i,j) with $i<j$	$-\frac{\sigma_i\sigma_j\rho_{ij}}{4h^2}$
$-(2M+1)^{n-i} - (2M+1)^{n-j}$	for all (i,j) with $i<j$	$-\frac{\sigma_i\sigma_j\rho_{ij}}{4h^2}$
$(2M+1)^{n-i} - (2M+1)^{n-j}$	for all (i,j) with $i<j$	$\frac{\sigma_i\sigma_j\rho_{ij}}{4h^2}$
$-(2M+1)^{n-i} + (2M+1)^{n-j}$	for all (i,j) with $i<j$	$\frac{\sigma_i\sigma_j\rho_{ij}}{4h^2}$

Table 4.3: The entries in each row of A_h corresponding to an interior index. The "distance from the diagonal" is to be understood as follows: for $\hat{\imath} \in \texttt{intind}$, the column index of the corresponding entry is obtained by adding the listed "distance" to $\hat{\imath}$.

Using the relations (2.34) and (2.35), we further find that

$$A_h^{(\mathrm{ii})}\mathbf{W}^{\mathrm{int}}(\texttt{intind}) = \big(A_h\mathbf{W}^{\mathrm{int}}\big)(\texttt{intind}) = \mathbf{W}^{\mathrm{int}}(\texttt{incrind}) \cdot \texttt{vals}^{\mathrm{T}}, \tag{4.25}$$

$$A_h^{(\mathrm{ib})}\mathbf{W}^{\mathrm{bd}}(\texttt{bdind}) = \big(A_h\mathbf{W}^{\mathrm{bd}}\big)(\texttt{intind}) = \mathbf{W}^{\mathrm{bd}}(\texttt{incrind}) \cdot \texttt{vals}^{\mathrm{T}}, \tag{4.26}$$

where

$$\texttt{incrind} := \texttt{repmat}(\,\texttt{intind}, 1, 2n^2+1\,) + \texttt{repmat}(\,\texttt{increments}, (2M-1)^n, 1\,), \tag{4.27}$$

i.e. the $(2M-1)^n \times (2n^2+1)$ matrix `incrind` contains in each row the set of a fixed interior index $\hat{\imath}$ increased or decreased by all increments listed in Table 4.3; $\mathbf{W}(\texttt{incrind})$ denotes the matrix made up of the corresponding entries of $\mathbf{W}$ and is valid MATLAB syntax. Thus, the product $A_h\mathbf{W}$ can be implemented as a multiplication by the $(2n^2+1)$-component vector `vals`, i.e. in linear complexity, and without generating the matrix itself.

As for $kB_{1,h} = I_{(2M-1)^n} + k\,\theta A_h^{(\mathrm{ii})}$, the product $kB_{1,h}\tilde{\mathbf{W}}$ with a vector $\tilde{\mathbf{W}} \in \mathbb{R}^{(2M-1)^n}$ can be implemented in a similar way as (4.25): we first supplement $\tilde{\mathbf{W}}$ by zero boundary values to a

$(2M+1)^n$-component vector $\tilde{\mathbf{W}}_{\mathrm{BV}} = \tilde{\mathbf{W}}^{\mathrm{int}}_{\mathrm{BV}}$, and extend $kB_{1,h}$ to $I_{(2M+1)^n} + k\,\theta A_h$. The entries of this matrix are given in (4.23); hence, the corresponding `kB1vals` vector can be generated from the `vals` vector corresponding to A_h by multiplying each component by $k\,\theta$ and then adding 1 to the first component (representing the diagonal element). The corresponding `increments` vector is the same as for the matrix A_h. Hence, we obtain

$$\begin{aligned} kB_{1,h}\tilde{\mathbf{W}} &= \Big(I_{(2M-1)^n} + k\,\theta A_h^{(\mathrm{ii})}\Big)\tilde{\mathbf{W}}^{\mathrm{int}}_{\mathrm{BV}}(\texttt{intind}) = \Big(\big(I_{(2M+1)^n} + k\,\theta A_h\big)\tilde{\mathbf{W}}^{\mathrm{int}}_{\mathrm{BV}}\Big)(\texttt{intind}) \\ &= \tilde{\mathbf{W}}^{\mathrm{int}}_{\mathrm{BV}}(\texttt{incrind}) \cdot \texttt{kB1vals}^{\mathrm{T}} = \tilde{\mathbf{W}}_{\mathrm{BV}}(\texttt{incrind}) \cdot \texttt{kB1vals}^{\mathrm{T}}. \end{aligned} \tag{4.28}$$

It remains to provide a vectorised algorithm to create the vectors `increments` and `vals`. As for the increments corresponding to the one-step neighbours, we can use again the `pow` vector as defined in line 2 of Algorithm 4, and hence define the first $2n+1$ entries of `increments` by `[0,pow,-pow]`. The corresponding `vals` entries can be computed from the input parameters r, R, M, and Σ (passed as an $n \times n$ matrix `sigma`): for a matrix `sigma`, the MATLAB command `diag(sigma)` (see [MATLAB 7.7.0 Help, 2008]) returns a column vector formed from the diagonal entries of `sigma`, i.e. in our case, $(\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2)^{\mathrm{T}}$. The transpose of this vector can be directly used to generate the entire `vals` fragments $\left(-\frac{\sigma_i^2}{2h^2} \pm \left(\frac{\sigma_i^2}{4h} - \frac{r}{2h}\right)\right)_{i=1}^{n}$, and $\sum_{i=1}^n \sigma_i^2$ needed for the first `vals` entry is obtained by the command `sum(diag(sigma),1)` where the first argument is an array and the second argument denotes the dimension along which the array elements are to be added up, cf. [MATLAB 7.7.0 Help, 2008].

For the `increments` entries corresponding to the two-step neighbours, we need increment combinations of the form

$$\begin{pmatrix} (2M+1)^{n-1} & \pm & (2M+1)^{n-2} \\ (2M+1)^{n-1} & \pm & (2M+1)^{n-3} \\ \vdots & & \vdots \\ (2M+1)^{n-1} & \pm & 1 \\ (2M+1)^{n-2} & \pm & (2M+1)^{n-3} \\ \vdots & & \vdots \\ (2M+1)^{n-2} & \pm & 1 \\ \vdots & & \vdots \\ (2M+1) & \pm & 1 \end{pmatrix}^{\mathrm{T}} \tag{4.29}$$

with all combinations $(n-i, n-j)$ with $i < j$ as exponents. In order to generate a matrix of the form (4.29), we first create a $2 \times \frac{n(n-1)}{2}$ matrix containing in its columns all combinations (i, j) with $i < j$, and thereafter take $(2M+1)$ to the required powers with the aid of the `.^` operator. The rows of the resulting matrix can then be added or subtracted as desired. A matrix of all combinations (i, j) with $i < j$ can be constructed according to the following idea: we first generate an $n \times n$ matrix containing the sequence $(1 : n)$ in each row. The lower and upper triangular part of this matrix (excluding the diagonal), extracted column-wise and row-wise, respectively, and written row-wise into a two-row matrix, then yield the required pairs (i, j). This is illustrated in Figure 4.3 for the case $n = 4$. In MATLAB, the functions `triu` and `tril` are available to extract the upper and lower triangular parts of a matrix, respectively (see [MATLAB 7.7.0 Help, 2008]): for a square matrix A, `tril(A,-1)` yields a matrix of the same size as A where the elements below the

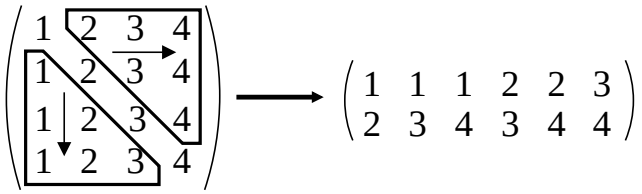

Figure 4.3: The construction of all combinations (i, j) with $i < j$.

diagonal of A are retained and all other elements are set to zero. If we know that all subdiagonal elements of A are positive, we can column-wise extract them with the command `A(tril(A,-1)>0)` (cf. [MATLAB 7.7.0 Help, 2008]).

The corresponding `vals` fragments $\pm\big(\frac{\sigma_i\sigma_j\rho_{ij}}{4h^2}\big)$with the (i, j) combinations from the columns of the two-row matrix shown in Figure 4.3 can be obtained by converting the array indices (i, j) to linear indices (see [MATLAB 7.7.0 Help, 2008]) and thus extracting the desired entries of the matrix Σ.

4.3.1 Implementation of the weighted Jacobi method

By (3.7), the $(m + 1)$-th Jacobi iterate for the system $A\mathbf{W} = \mathbf{RHS}$ is defined by

$$\mathbf{W}_{m+1} = \mathbf{W}_m + \omega D^{-1}\mathbf{R}_m = \mathbf{W}_m + \omega D^{-1}(\mathbf{RHS} - A\mathbf{W}_m), \tag{4.30}$$

where D is made up of the diagonal of A. As already indicated, the matrix-vector multiplication algorithm developed in the previous section (see (4.25)–(4.28)) can be used both for assembling the right-hand side vector of the system (4.3) at each time step (where the approximation vector $\mathbf{W}^\ell$ is known at time step $\ell + 1$), and in the context of the weighted Jacobi method. The system matrix of (4.3) is $kB_{1,h} = I_{(2M-1)^n} + k\,\theta A_h^{(\mathrm{ii})}$ whose diagonal elements are all equal to

$$\texttt{kB1vals}(1) = 1 + k\,\theta\Big(r + \sum_{i=1}^{n}\frac{\sigma_i^2}{h^2}\Big) \tag{4.31}$$

independently of the time step. Hence, the subfunction jacobi $(M, \mathbf{RHS}_h, \mathbf{W}_{0,h}, \omega, \nu)$ (see (3.25)) used in the multigrid algorithm listings can be implemented as indicated in Algorithm 6. Line 7 implements the calculation of the current residual using (4.28) for the product $kB_{1,h}\mathbf{W}$; to this end, the vector $\mathbf{W}_{\mathrm{BV}}$ supplementing $\mathbf{W}$ by zero boundary values has to be generated in line 5. Line 8 then implements (4.30). As all diagonal elements of $kB_{1,h}$ are equal, the multiplication by D^{-1} in (4.30) reduces to a division by the scalar `kB1vals`(1). Finally, $\mathbf{W}_{\mathrm{BV}}$ is accordingly updated in line 9. Note that the algorithm can be called with $\nu = 0$ which may occur in the context of the multigrid method if no pre- or post-smoothing is required; in this case, $\mathbf{W}_0$ is returned due to the initialisation in line 4, i.e. the current approximation remains unchanged.

4.4 Intergrid transfer operations

An important feature needed in the multigrid method is intergrid transfer between the meshes $\Omega^{\mathbf{x}}_{R,h}$ and $\Omega^{\mathbf{x}}_{R,2h}$; in our case, we will use n-linear interpolation and full weighting restriction. In

Algorithm 6 Weighted Jacobi method applied to $kB_{1,h}\mathbf{W} = \mathbf{RHS}$ on the interior grid points of $\Omega^{\mathbf{x}}_{R,h}$

1: **function** $\mathbf{W} = \text{jacobi}\,(M, \mathbf{RHS}, \mathbf{W}_0, \omega, \nu, k, R, r, \Sigma)$
Input: grid parameter M, right-hand side vector $\mathbf{RHS} \in \mathbb{R}^{(2M-1)^n}$, initial guess $\mathbf{W}_0 \in \mathbb{R}^{(2M-1)^n}$, weighting factor ω, number of iterations $\nu \geq 0$, time step k, problem parameters R, r, Σ
Output: ν-th iterate of the weighted Jacobi method with initial guess $\mathbf{W}_0$
2: Generate the `intind` vector for the mesh $\Omega^{\mathbf{x}}_{R,h}$.
3: Generate the `kB1vals` and `increments` vectors and the matrix `incrind` described in Section 4.3.
4: $\mathbf{W} \leftarrow \mathbf{W}_0$
5: Initialise $\mathbf{W}_{\text{BV}}$ as a $(2M+1)^n$-component vector of zeros, and set $\mathbf{W}_{\text{BV}}(\texttt{intind}) \leftarrow \mathbf{W}$
6: **for** $i = 1 : \nu$ **do**
7: $\quad \mathbf{R} \leftarrow \mathbf{RHS} - \mathbf{W}_{\text{BV}}(\texttt{incrind}) \cdot \texttt{kB1vals}^{\text{T}}$
8: $\quad \mathbf{W} \leftarrow \mathbf{W} + \frac{\omega}{\texttt{kB1vals(1)}}\,\mathbf{R}$
9: $\quad \mathbf{W}_{\text{BV}}(\texttt{intind}) \leftarrow \mathbf{W}$
10: **end for**
11: **return** $\mathbf{W}$

Section 3.2.3, we have already indicated how to construct the (sparse) matrix P representing the n-linear interpolation mapping from $\mathbb{R}^{(M+1)^n}$ to $\mathbb{R}^{(2M+1)^n}$: each column corresponds to a coarse-grid point, and in all rows corresponding to surrounding fine-grid points, the weight with which the coarse-grid value enters into the value at the respective fine-grid point is entered. Thus, row and column index vectors and a value vector can be defined which the matrix P can be generated from: an $m \times n$ sparse matrix P can be directly generated in MATLAB with the function call `P = sparse(I,J,Pvals,m,n)`, where `Pvals` is a vector containing the non-zero elements of the matrix, and I and J are vectors of the same length containing the corresponding row and column indices, respectively (see [MATLAB 7.7.0 Help, 2008]).

In the practical implementation, we may proceed by creating a list of all coarse-grid multiindices, doubling each component in order to obtain the corresponding fine-grid indices (cf. (3.27)), and then creating all combinations of increasing or decreasing the components by 1 in order to obtain the fine-grid multiindices of all surrounding fine-grid points which are not part of the coarse grid. Using Formula (4.5), we can convert the multiindices back to fine-grid array indices, corresponding to row numbers in the matrix P. However, we have to be careful about the treatment of boundary grid points, as those do not have the full set of surrounding fine-grid points. Instead of individually determining the neighbours of each boundary coarse-grid point, however, we observe that this problem does not occur when interpolating vectors with zero boundary values, and subsequently show how we can use the prolongation routine for vectors with zero boundary values to treat vectors with arbitrary boundary values as well.

4.4.1 n-linear interpolation for vectors with zero boundary values

If both the coarse-grid and the fine-grid vector have zero boundary values, Figure 4.4 illustrates that it suffices to consider only the *interior* coarse-grid points in the above procedure: All *interior* fine-grid points can be reached and assigned all relevant value contributions; the contribution of the boundary coarse-grid points can be neglected as it is zero. As for the fine-grid boundary values,

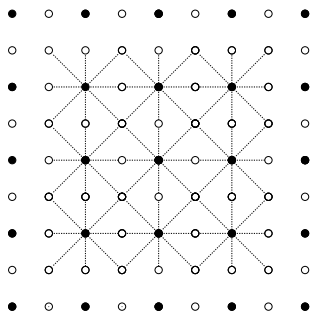

Figure 4.4: Illustration of which fine-grid points can be reached as neighbours of interior coarse-grid points (filled) in a two-dimensional mesh.

no coarse-grid contributions are assigned in this case, yet the resulting zero rows in the matrix P correctly define the boundary values to be zero. The advantage now is that every coarse-grid point taken into consideration has fine-grid neighbours in any direction and hence that no distinction has to be made.

Let $\frac{M}{2}$ be the grid parameter of the original coarse mesh. As suggested above, we start by creating a matrix listing the coarse-grid multiindices of all *interior* coarse-grid points, i.e. all multiindices with components between $-(\frac{M}{2}-1)$ and $\frac{M}{2}-1$. To this end, we devise an algorithm to create all n-tupels with components between and including arbitrary numbers a and b in lexicographical order; we can use a similar strategy as for the matrix M_2 in (4.11). For illustration, let us consider the case $n = 2$, $a = 0$ and $b = 2$, then we are aiming to create the matrix

$$\texttt{ind} := \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 2 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 2 & 0 \\ 2 & 1 \\ 2 & 2 \end{pmatrix}. \tag{4.32}$$

Define the auxiliary matrices,

$$\tilde{M}_0 := \begin{pmatrix} 0 \ : \ (b-a+1)^n - 1 \\ \cdots\cdots\cdots\cdots\cdots \\ 0 \ : \ (b-a+1)^n - 1 \end{pmatrix}^{\mathrm{T}} \qquad \in \mathbb{R}^{(b-a+1)^n \times n}, \tag{4.33}$$

$$\tilde{M}_4 := \begin{pmatrix} (b-a+1)^{n-1} & (b-a+1)^{n-2} & \ldots & (b-a+1) & 1 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ (b-a+1)^{n-1} & (b-a+1)^{n-2} & \ldots & (b-a+1) & 1 \end{pmatrix} \qquad \in \mathbb{R}^{(b-a+1)^n \times n}, \tag{4.34}$$

$$\tilde{M}_1 := (b-a+1) \cdot \tilde{M}_4. \tag{4.35}$$

In the illustration case, these matrices are given by

$$\tilde{M}_0 = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 2 & 2 \\ 3 & 3 \\ 4 & 4 \\ 5 & 5 \\ 6 & 6 \\ 7 & 7 \\ 8 & 8 \end{pmatrix}, \quad \tilde{M}_1 = \begin{pmatrix} 9 & 3 \\ 9 & 3 \\ 9 & 3 \\ 9 & 3 \\ 9 & 3 \\ 9 & 3 \\ 9 & 3 \\ 9 & 3 \\ 9 & 3 \end{pmatrix}, \quad \tilde{M}_4 = \begin{pmatrix} 3 & 1 \\ 3 & 1 \\ 3 & 1 \\ 3 & 1 \\ 3 & 1 \\ 3 & 1 \\ 3 & 1 \\ 3 & 1 \\ 3 & 1 \end{pmatrix}. \tag{4.36}$$

The matrix of remainders after the component-wise division of $\tilde{M}_0$ by $\tilde{M}_1$, $\texttt{rem}\,(\tilde{M}_0, \tilde{M}_1)$ in MATLAB notation (see [MATLAB 7.7.0 Help, 2008]), is

$$\begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 2 & 2 \\ 3 & 0 \\ 4 & 1 \\ 5 & 2 \\ 6 & 0 \\ 7 & 1 \\ 8 & 2 \end{pmatrix}. \tag{4.37}$$

After component-wise integer division of this matrix by $\tilde{M}_4$, we arrive at

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 2 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 2 & 0 \\ 2 & 1 \\ 2 & 2 \end{pmatrix}, \tag{4.38}$$

i.e. the desired matrix `ind`. In the general case, the result at this stage is a matrix containing all n-tupels with components between 0 and $b-a$, and we have to add a to each component to obtain the n-tupels with components between a and b. A MATLAB code fragment implementing this procedure for arbitrary n is shown in Algorithm 7.

We can thus generate a list `ind` of coarse-grid multiindices of all interior coarse-grid points with the aid of Algorithm 7 with $a = -\frac{M}{2}+1$, $b = \frac{M}{2}-1$. The rows of this matrix can then be converted to the column vector `intcoarse` of corresponding coarse-grid array indices by using Formula (4.5) with $\frac{M}{2}$ instead of M. The vector `intfine` of fine-grid array indices of the same grid points is obtained by using (4.5) with M and the rows of $2 \cdot$ `ind` (cf. (3.27)). With the aid of these vectors and following the ideas laid out in Section 3.2.3, we can assemble the row index, column index and value vectors `I`, `J` and `Pvals` needed to generate the matrix P as shown in Table 4.4.

Algorithm 7 Creating the set of all n-tupels with components between and including a and b, written in the rows of the matrix `ind` in lexicographical order.

```
1 pow = (b-a+1).^(n-1:-1:0);
2 M0 = repmat( (0:((b-a+1)^n-1))', 1,n);
3 M4 = repmat( pow, (b-a+1)^n,1);
4 ind = floor( rem( M0,(b-a+1)*M4 )./ M4 ) + a;
```

I		J	Pvals
`intfine`		`intcoarse`	$\mathbf{1}$
`intfine` $\pm(2M+1)^{n-i}\cdot\mathbf{1}$	for all i	`intcoarse`	$\frac{1}{2}\cdot\mathbf{1}$
`intfine` $\pm(2M+1)^{n-i}\cdot\mathbf{1}$ $\pm(2M+1)^{n-j}\cdot\mathbf{1}$	for all $i<j$	`intcoarse`	$\frac{1}{4}\cdot\mathbf{1}$
$\vdots$		$\vdots$	$\vdots$
`intfine` $+\sum_{i=1}^{n}\left[\pm(2M+1)^{n-i}\right]\cdot\mathbf{1}$		`intcoarse`	$\frac{1}{2^n}\cdot\mathbf{1}$

Table 4.4: Fragments making up the row index, column index, and value vectors for P. $\mathbf{1}$ denotes an $(M-1)^n$-component vector of all ones.

For illustration, let us consider the two-dimensional case again. For the row index fragments, we need increments of the form

$$\underbrace{\begin{pmatrix} 0 & 0\\ 0 & -1\\ 0 & 1\\ -1 & 0\\ 1 & 0\\ -1 & -1\\ -1 & 1\\ 1 & -1\\ 1 & 1 \end{pmatrix}}_{=:\texttt{comb}} \cdot \binom{2M+1}{1}. \tag{4.39}$$

Note that the $3^n \times n$ matrix `comb` corresponds to the set of all n-tupels with components between -1 and 1, hence we can use Algorithm 7 to create it. This yields the n-tupels in lexicographical order unlike shown in (4.39), yet the order is irrelevant in this case. Thus, the entire `I` vector can be generated by adding `comb*((2*M+1).^(n-1:-1:0)')` to each component of `intfine`. Finally, the corresponding exponent of 2 in the `Pvals` entry is given by the negative sum of the moduli of the entries in the corresponding row of `comb`, corresponding to the number of directions in which the fine-grid point is shifted with respect to the coarse grid axes.

To summarise, we present a full MATLAB code to generate the matrix P in Algorithm 8. Lines 1–4 implement Algorithm 7 with $a = -\frac{M}{2}+1$, $b = \frac{M}{2}-1$ in a more compact way, and finally

Algorithm 8 Creating the matrix P representing n-linear interpolation from $\Omega^{\mathbf{x}}_{R,2h}$ to $\Omega^{\mathbf{x}}_{R,h}$ for vectors with zero boundary values.

```
1  pow = (M-1).^(n-1:-1:0);
2  ind = floor( rem( repmat( (0:((M-1)^n-1))', 1,n), ...
3                    repmat( (M-1)*pow, (M-1)^n,1) )...
4               ./ repmat( pow, (M-1)^n,1)           ) + 1;
5  ind = ind';
6  intcoarse = ((  M+1).^(n-1:-1:0))*  ind + 1;
7  intfine   = ((2*M+1).^(n-1:-1:0))*2*ind + 1;
8  comb = floor( rem( repmat( (0:(3^n -1))', 1,n),...
9                     repmat( 3.^(n:-1:1), 3^n,1) )...
10                ./ repmat( 3.^(n-1:-1:0), 3^n,1)   ) - 1;
11 incr = comb*((2*M+1).^(n-1:-1:0)');
12 I = repmat( intfine, 3^n,1) + repmat( incr, 1,(M-1)^n);
13 J = repmat( intcoarse, 3^n,1);
14 Pvals = 2.^(-sum(abs(comb),2));
15 P = sparse( I(:), J(:), repmat( Pvals, (M-1)^n,1), (2*M+1)^n, (M+1)^n );
```

increase each component of `ind` by $\frac{M}{2}$ as a first step in the conversion to coarse-grid array indices. In lines 6 and 7, the array index vectors `intcoarse` and `intfine` are generated as indicated above using a vectorised version of Formula (4.5), expressed as an inner product. In lines 8–10, the `comb` vector is generated according to Algorithm 7 using $a = -1$, $b = 1$. In line 12, the row index vector `I` is assembled by vertically concatenating 3^n copies of the row vector `intfine`, and adding $(M-1)^n$ horizontally concatenated copies of the column vector `incr` containing the neighbour increments as in (4.39). By this combination of vertical and horizontal copies, we achieve that each `intfine` component is combined with each increment. The resulting matrix is converted back to a column vector in line 15: for an arbitrarily shaped array A, the MATLAB command `A(:)` creates a vector containing its components as a single column (see [MATLAB 7.7.0 Help, 2008]). Line 14 generates the `Pvals` vector by taking 2 to the negative power of the row-wise summed matrix of absolute values of `comb`.

4.4.2 n-linear interpolation for vectors with arbitrary boundary values

Algorithm 8 is designed for vectors whose boundary entries are all zero, as only the interior coarse-grid point values are taken into account. Figure 4.5 highlights the fine-grid points in a two-dimensional mesh on which the coarse-grid boundary values have no influence during n-linear interpolation. Hence, the following idea how to extend Algorithm 8 to a prolongation routine for vectors with arbitrary boundary values suggests itself:

- Create the matrix representing the prolongation mapping for vectors with zero boundary values from a coarse grid with parameter $\frac{M}{2} + 1$ to the fine grid with parameter $M + 2$, i.e. grids which are slightly larger than the ones in consideration.
- Delete the entries corresponding to the two "outermost fine-grid boundary layers", i.e. the grid points of $\Omega^{\mathbf{x}}_{R+2h,h}$ which are not part of $\Omega^{\mathbf{x}}_{R,h}$, and of $\Omega^{\mathbf{x}}_{R+2h,2h}$ which are not part of

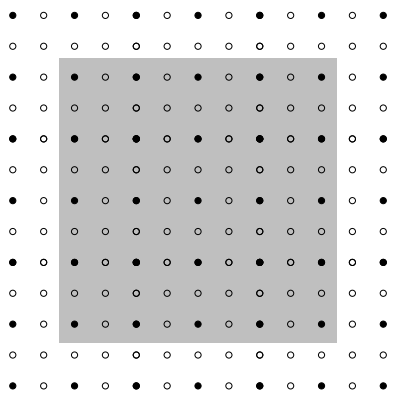

Figure 4.5: Illustration of the properties of the prolongation procedure in Algorithm 8: for all fine-grid points within the grey box, the values at all nearest coarse-grid neighbours are taken into account and there is no influence of the boundary values.

$\Omega^{\mathbf{x}}_{R,2h}$ (those outside the grey box in Figure 4.5).

This leaves the submatrix corresponding to the prolongation mapping from $\Omega^{\mathbf{x}}_{R,2h}$ to $\Omega^{\mathbf{x}}_{R,h}$ where all boundary values are taken into account.

Before we proceed to the practical implementation of this idea, two remarks are in place: first, note that Algorithm 8 does *not* require the mesh parameter M to be a power of two, it rather works for any integer $M \geq 2$. Therefore, we can always apply it to the mesh with parameter $M+2$ in the first step of the above algorithm. Second, although we mention the actual diameters R of the meshes in the second step, they play no role in the creation of the prolongation matrix in fact, as only the number and relative positions of grid points are relevant.

Algorithm 9 shows how the above idea can be implemented using known routines. From the

Algorithm 9 Creating the matrix P representing n-linear interpolation from $\Omega^{\mathbf{x}}_{R,2h}$ to $\Omega^{\mathbf{x}}_{R,h}$ for vectors with arbitrary boundary values.

1: Use Algorithm 8 with fine-grid parameter $M+2$ to obtain the matrix $P \in \mathbb{R}^{(2M+5)^n \times (M+3)^n}$.
2: Find the set of boundary grid point indices for the mesh with parameter $M+2$ (Algorithm 4), and delete the rows of P corresponding to these indices.
3: Find the set of boundary grid point indices for the mesh with parameter $M+1$ (Algorithm 4), and delete the rows of P corresponding to these indices.
4: Find the set of boundary grid point indices for the mesh with parameter $\frac{M}{2}+1$ (Algorithm 4), and delete the columns of P corresponding to these indices.
5: Return the resulting submatrix $P \in \mathbb{R}^{(2M+1)^n \times (M+1)^n}$.

rows of the original "large" matrix P, we have to eliminate all components corresponding to grid points of the mesh with parameter $M+2$ which are not part of $\Omega^{\mathbf{x}}_{R,h}$. This is done by first deleting the rows corresponding to boundary grid points in line 2. Note that the remaining grid points correspond to the mesh with parameter $M+1$, in lexicographical order. In order to eliminate the "second boundary layer" (cf. Figure 4.5), we thus delete the entries corresponding to boundary grid points of this mesh. Note that the order of lines 2 and 3 is substantial, whereas line 4 taking care

of the coarse-grid points may be executed at any stage.

Regarding Section 3.2.3, once the prolongation matrix P has been created, the matrix representing full weighting restriction is simply obtained by transposing P and weighting it by 2^{-n}. If a fine-grid vector with zero boundary values is to be restricted, the matrix P may be generated according to Algorithm 8 rather than Algorithm 9.

4.5 The full Black-Scholes solver

Combining the routines developed in the previous sections, we can assemble a full numerical scheme to approximately compute the solution vectors $\mathbf{W}^\ell$, $\ell = 0, \ldots, K$ of the Crank-Nicolson and the implicit Euler discretisations (4.1)–(4.3) of the truncated transformed Black-Scholes problem (1.57)–(1.59), based on the full multigrid V-cycle method with the weighted Jacobi method as smoothing operation, n-linear interpolation and full weighting restriction.

Algorithm 10 presents a pseudocode implementation of the Crank-Nicolson method. Note that

Algorithm 10 Crank-Nicolson method for the truncated transformed Black-Scholes problem

function $\mathbf{W} = \mathrm{CN}\,(n, M, R, k, T, r, \Sigma, \tilde{w}_0, \tilde{g}, \nu_1, \nu_2, \nu, \omega)$

Input: number of underlying assets n, grid parameters $M = 2^m$ and R, time step k, maturity date T, riskless interest rate r, covariance matrix Σ, initial function $\tilde{w}_0$, boundary function $\tilde{g}$, smoothing parameters ν_1 and ν_2 for the multigrid method, cycling parameter ν for the full multigrid method, weighting factor ω for the weighted Jacobi method

Output: matrix $\mathbf{W} \in \mathbb{R}^{(2M+1)^n \times (K+1)}$ containing in its columns the vectors $\mathbf{W}^\ell$, $\ell = 0, \ldots, K$ (approximately) solving (4.1)–(4.3)

Initialise $\mathbf{W}$ as a zero matrix.

Generate a matrix `coord` of all grid point coordinates $\big(\frac{R}{M} \cdot$ `alpha` with `alpha` according to Algorithm 5 with $\mathbf{y} = (1 : (2M+1)^n)^{\mathrm{T}}\big)$.

Evaluate $\tilde{w}_0$ at `coord` and hence assign the first column of $\mathbf{W}$ with the initial data.

Generate the boundary and interior array index vectors `bdind` and `intind` for the mesh $\Omega^{\mathbf{x}}_{R,h}$ (Algorithm 4).

Generate the `vals` vector for the matrix A_h and the `incrind` matrix according to Section 4.3. Analogously to `kB1vals` described there, generate the `kB0vals` vector of non-zero values of the matrix $I - \frac{k}{2}A_h$.

for $i = 1 : K$ **do**

  Generate the vectors $\tilde{\mathbf{g}}^\ell$ and $\tilde{\mathbf{g}}^{\ell+1}$ by evaluating $\tilde{g}$ at `coord(bdind,:)` and at time $(i-1)k$ and ik, respectively, and supplementing the results by zero interior values.

  Assemble the right-hand side vector of (4.3) with $\theta = \frac{1}{2}$ using the matrix-vector product algorithm from (4.25)–(4.28):
  $\mathbf{W}^{\ell,\mathrm{int}} \leftarrow \mathbf{W}($`intind`$, i)$ supplemented by zero boundary values
  $\mathbf{RHS} \leftarrow \mathbf{W}^{\ell,\mathrm{int}}($`incrind`$) \cdot$ `kB0vals`$^{\mathrm{T}} - \frac{k}{2}\big(\tilde{\mathbf{g}}^\ell($`incrind`$) + \tilde{\mathbf{g}}^{\ell+1}($`incrind`$)\big) \cdot$ `vals`$^{\mathrm{T}}$

  $\mathbf{W}(:, i+1) \leftarrow \mathrm{FMG}^*\,(n, M, R, k, r, \Sigma, \mathbf{RHS}, \tilde{\mathbf{g}}^{\ell+1}, \nu_1, \nu_2, \nu, \omega)$

end for

return $\mathbf{W}$

although only the interior components of the $\mathbf{W}^\ell$ vectors are involved in the linear system (4.3), we

assume the full multigrid V-cycle function FMG* called in line 10 to return the *full* $W^{\ell+1}$ vector including the correct boundary values (using a colon instead of a specified vector as row or column reference is interpreted as *all* existing rows or columns in MATLAB syntax, see [MATLAB 7.7.0 Help, 2008]). The reason for this is that the boundary data are required in the full multigrid V-cycle algorithm (see the subsequent paragraph). As the vector $\tilde{\mathbf{g}}^{\ell+1}$ of boundary values of $\mathbf{W}^{\ell+1}$ is calculated in line 8 of Algorithm 10 in any case, it makes sense to pass this knowledge to the FMG* function as an input parameter in order to avoid redundant evaluations of the function $\tilde{g}$ which are in general computationally expensive. On the other hand, the FMG* function can thus conveniently be defined in such a way that the output already includes the correct boundary values.

In the nested iteration scheme, approximations of the vector $\mathbf{W}$ on different grids are involved. The correct boundary values of $\mathbf{W}$ on any grid are prescribed by the boundary function $\tilde{g}$ which is not zero in general. Hence, in interpolating approximations of $\mathbf{W}$ from a coarse to a fine grid, the boundary values have to be taken into account in order to obtain a correct interpolation at fine-grid points near the boundary. For the fine-grid points directly on the boundary, however, the obtained interpolation will in general not be equal to the correct boundary values prescribed by $\tilde{g}$. Hence, coarse-grid boundary values are necessary for the interpolation, yet the fine-grid boundary values have to be additionally corrected after the interpolation step.

Algorithm 11 presents an appropriate implementation of the FMG* function. This is a slightly modified version of Algorithm 3 involving the modified multigrid μ-cycle function MG* that is shown in Algorithm 12 below. First, unless we are on the coarsest grid already, the right-hand side vector is restricted to the coarser mesh $\Omega^{\mathbf{x}}_{R,2h}$ (line 6). Note that we have so far always defined the prolongation matrix P to interpolate coarse-grid vectors *including boundary values* to fine-grid vectors *including boundary values*, hence the corresponding weighted restriction matrix $R = \frac{1}{2^n}P^{\mathrm{T}}$ acts on fine-grid vectors in $\mathbb{R}^{(2M+1)^n}$. However, the right-hand side vector **RHS** only includes components corresponding to *interior* grid points, hence it cannot directly be multiplied by R. A possible way to bypass this mismatch would be to extend **RHS** by arbitrary boundary values before applying the restriction matrix. The boundary values have no influence on the *interior* components of the result (note that this holds for restriction, but not for prolongation!), and only those are needed in the subsequent process as right-hand side vector on the coarser grid. We may for instance extend **RHS** by zero boundary values. The product $R\,\mathbf{RHS}_{\mathrm{BV}}$ is then the same as if we multiplied the original vector **RHS** by the submatrix of R obtained by omitting the columns corresponding to fine-grid boundary indices. Hence, instead of extending **RHS**, we may as well restrict R, and this is the strategy we use in Algorithm 11: in line 6, we extract only the rows of P corresponding to interior fine-grid indices, i.e. we obtain only the columns of R corresponding to interior fine-grid indices. As we are only interested in the interior components of the restricted vector, we additionally restrict to the columns of P corresponding to interior coarse-grid indices (i.e. rows of R corresponding to interior coarse-grid indices). Note that as the boundary entries of P are omitted in this step, we actually do not need the prolongation matrix for vectors with arbitrary boundary values (according to Algorithm 9), but we might as well use the prolongation matrix for vectors with zero boundary values (according to the less complex Algorithm 8). However, as we have argued above, we need the prolongation matrix for vectors with arbitrary boundary values in line 9 where $\mathbf{W}_{2h}$ is interpolated to the finer grid $\Omega^{\mathbf{x}}_{R,h}$. Therefore, it makes sense to generate the matrix P for vectors with arbitrary boundary values once and for all in line 5. After restricting the right-hand side vector, the FMG* function is called on the mesh $\Omega^{\mathbf{x}}_{R,2h}$ in line 8. However, FMG* also requires the boundary values of the result (i.e. of $\mathbf{W}_{2h}$) as input. These may be calculated with

Algorithm 11 Full multigrid V-cycle method for use in a Crank-Nicolson or implicit Euler discretisation

1: **function** $\mathbf{W} = \mathrm{FMG}^*\,(n, M, R, k, r, \Sigma, \mathbf{RHS}, \mathrm{BVs}, \nu_1, \nu_2, \nu, \omega)$

Input: number of underlying assets n, grid parameters $M = 2^m$ and R, time step k, riskless interest rate r, covariance matrix Σ, right-hand side vector $\mathbf{RHS} \in \mathbb{R}^{(2M-1)^n}$, boundary value vector $\mathrm{BVs} \in \mathbb{R}^{(2M+1)^n}$, smoothing parameters ν_1 and ν_2, cycling parameter ν, weighting factor ω for the weighted Jacobi method

Output: $\mathbf{W} \in \mathbb{R}^{(2M+1)^n}$: result of the full multigrid V-cycle method (with the specified parameters) applied to the problem (4.1)–(4.3)

2: Initialise $\mathbf{W}$ as a zero vector.

3: Generate the boundary and interior array index vectors `bdind` and `intind` for the mesh $\Omega^{\mathbf{x}}_{R,h}$, and $\mathtt{bdind}_{2h}$ and $\mathtt{intind}_{2h}$ for the mesh $\Omega^{\mathbf{x}}_{R,2h}$ (Algorithm 4).

4: **if** $M > 1$ **then**

5: Generate the matrix P representing n-linear interpolation from $\Omega^{\mathbf{x}}_{R,2h}$ to $\Omega^{\mathbf{x}}_{R,h}$ for vectors with arbitrary boundary values (Algorithm 9).

6: Use the submatrix of P corresponding to interior indices to restrict the right-hand side vector to $\Omega^{\mathbf{x}}_{R,2h}$:
$\mathbf{RHS}_{2h} \leftarrow \frac{1}{2^n} P(\mathtt{intind}, \mathtt{intind}_{2h})^{\mathrm{T}}\,\mathbf{RHS}$

7: Define the vector BVs_{2h} of coarse-grid boundary values by extracting the appropriate entries of BVs (to find those, convert the coarse-grid array index vector $\mathtt{bdind}_{2h}$ to coarse-grid multiindices using Algorithm 5 and then convert these to fine-grid array indices as in creating `intfine` in Algorithm 8).

8: $\mathbf{W}_{2h} \leftarrow \mathrm{FMG}^*\left(n, \frac{M}{2}, R, k, r, \Sigma, \mathbf{RHS}_{2h}, \mathrm{BVs}_{2h}, \nu_1, \nu_2, \nu, \omega\right)$

9: $\mathbf{W} \leftarrow \mathbf{W} + P\,\mathbf{W}_{2h}$

10: **end if**

11: $\mathbf{W}(\mathtt{bdind}) \leftarrow \mathrm{BVs}(\mathtt{bdind})$

12: **for** $i = 1 : \nu$ **do**

13: $\mathbf{W}(\mathtt{intind}) \leftarrow \mathrm{MG}^*\,(n, M, R, k, r, \Sigma, \mathbf{W}(\mathtt{intind}), \mathbf{RHS}, \nu_1, \nu_2, 1, \omega)$

14: **end for**

15: **return** $\mathbf{W}$

the aid of the boundary function $\tilde{g}$. However, we can again avoid redundant function evaluations at this step: all coarse-grid boundary grid points are part of the fine-grid boundary as well, and the corresponding values are included in the known BVs vector. We can extract them as described in line 7. The result of the FMG* function call then has to be interpolated to $\Omega^{\mathbf{x}}_{R,h}$ again in line 9, and as mentioned above, the boundary values have to be corrected after that in line 11. This line is also executed on the coarsest grid in order to define the boundary values of $\mathbf{W}$.

The procedure up to this point implements the nested iteration method to find a good initial guess for $\mathbf{W}$ to be used in the multigrid μ-cycle method. The multigrid μ-cycle merely concerns the linear system in the *interior* components of the $\mathbf{W}$ vector; boundary values are not needed any more as we will see below. Hence, the MG* function is called only for $\mathbf{W}(\mathtt{intind})$ and with initial guess $\mathbf{W}(\mathtt{intind})$ in line 13 of Algorithm 11. In any case, the output of FMG* contains the correct boundary values due to the previous assignment in line 11.

The corresponding multigrid μ-cycle implementation is shown in Algorithm 12. As indicated in Algorithm 2, if the coarsest grid with $M = 1$ is reached, the linear system $kB_{1,h}\mathbf{W} = \mathbf{RHS}$ is

Algorithm 12 Multigrid μ-cycle

1: **function** $\mathbf{W} = \mathrm{MG}^*\,(n, M, R, k, r, \Sigma, \mathbf{W}_0, \mathbf{RHS}, \nu_1, \nu_2, \mu, \omega)$

Input: number of underlying assets n, grid parameters $M = 2^m$ and R, time step k, risk-less interest rate r, covariance matrix Σ, initial guess $\mathbf{W}_0 \in \mathbb{R}^{(2M-1)^n}$, right-hand side vector $\mathbf{RHS} \in \mathbb{R}^{(2M-1)^n}$, smoothing parameters ν_1 and ν_2, cycling parameter μ, weighting factor ω for the weighted Jacobi method

Output: $\mathbf{W} \in \mathbb{R}^{(2M-1)^n}$: result of the multigrid μ-cycle method (with the specified parameters) applied to the problem $kB_{1,h}\mathbf{W} = \mathbf{RHS}$ on the interior grid points of the mesh $\Omega^{\mathbf{x}}_{R,h}$ with initial guess $\mathbf{W}_0$

2: **if** $M = 1$ **then**

3: $\quad \mathbf{W} \leftarrow \frac{1}{1+k\,\theta\left(r+\sum_{i=1}^{n}\frac{\sigma_i^2}{h^2}\right)}\,\mathbf{RHS}$

4: **else**

5: $\quad \mathbf{W} \leftarrow \mathrm{jacobi}\,(M, \mathbf{RHS}, \mathbf{W}_0, \omega, \nu_1, k, R, r, \Sigma)$ (Algorithm 6)

6: $\quad$ Calculate the residual $\mathbf{R}$ of $\mathbf{W}$ as in line 7 of Algorithm 6.

7: $\quad$ Generate the matrix P representing n-linear interpolation from $\Omega^{\mathbf{x}}_{R,2h}$ to $\Omega^{\mathbf{x}}_{R,h}$ for vectors with zero boundary values (Algorithm 8).

8: $\quad \mathbf{R}_{2h} \leftarrow \frac{1}{2^n} P(\mathtt{intind}, \mathtt{intind}_{2h})^{\mathrm{T}}\,\mathbf{R}$

9: $\quad \mathbf{E}_{2h} \leftarrow \mathbf{0}$

10: $\quad$ **for** $i = 1 : \mu$ **do**

11: $\quad\quad \mathbf{E}_{2h} \leftarrow \mathrm{MG}^*\left(n, \frac{M}{2}, R, k, r, \Sigma, \mathbf{E}_{2h}, \mathbf{R}_{2h}, \nu_1, \nu_2, \mu, \omega\right)$

12: $\quad$ **end for**

13: $\quad \mathbf{W} \leftarrow \mathbf{W} + P(\mathtt{intind}, \mathtt{intind}_{2h})\,\mathbf{E}_{2h}$

14: $\quad \mathbf{W} \leftarrow \mathrm{jacobi}\,(M, \mathbf{RHS}, \mathbf{W}, \omega, \nu_2, k, R, r, \Sigma)$ (Algorithm 6)

15: **end if**

16: **return** $\mathbf{W}$

to be solved exactly. As we have already seen, there is only one interior grid point in this mesh, hence $kB_{1,h}$ is a 1×1 matrix made up of the "diagonal element" $1 + k\,\theta\left(r + \sum_{i=1}^{n}\frac{\sigma_i^2}{h^2}\right)$ (see (4.31)), and the solution of the corresponding scalar equation is obtained by dividing **RHS** by this value (see line 3). Otherwise, pre-smoothing is performed (line 5), the residual is calculated (line 6) and restricted to the grid $\Omega^{\mathbf{x}}_{R,2h}$ (line 8). Note that like in the case of restricting the right-hand side vector in Algorithm 11, we want to restrict a vector without boundary values here. As we have seen above, we can implement this by omitting the rows and columns of P corresponding to boundary grid points. In contrast to the FMG* function, the less complex prolongation matrix for vectors with zero boundary values can be used within the MG* function (generated in line 7): for the restriction, this is insignificant in both cases; as for the prolongation, however, the vector to be interpolated is $\mathbf{W}_{2h}$ in FMG* and $\mathbf{E}_{2h}$ in MG*. The former has potentially non-zero boundary values prescribed by $\tilde{g}$, whereas the latter is the error of $\mathbf{W}_{2h}$ with respect to the exact solution of the system (4.3) on $\Omega^{\mathbf{x}}_{R,2h}$. As the boundary values are defined by (4.2) for the approximation as well as the exact solution, we can say that if $\mathbf{E}_{2h}$ were supplemented by boundary values, it would have zero boundary values. Hence, the prolongation matrix for vectors with zero boundary values, or in effect, the submatrix corresponding to interior grid points, can be used in line 13. Finally, post-smoothing is performed in line 14.

In order to implement the implicit Euler method instead of the Crank-Nicolson method, only

small changes are necessary in Algorithm 10. A pseudocode implementation of the corresponding IE function is given in Algorithm 13. Furthermore, the MG* function (in particular, the weighted Jacobi method) has to be modified to solve the system $\left(I_{(2M-1)^n} + k\, A_h^{(\mathrm{ii})}\right)\mathbf{W} = \mathbf{RHS}$ rather than $\left(I_{(2M-1)^n} + \frac{k}{2} A_h^{(\mathrm{ii})}\right)\mathbf{W} = \mathbf{RHS}$; the FMG* function need not be altered.

Algorithm 13 Implicit Euler method for the truncated transformed Black-Scholes problem

1: **function** $\mathbf{W} = \mathrm{IE}\,(n, M, R, k, T, r, \Sigma, \tilde{w}_0, \tilde{g}, \nu_1, \nu_2, \nu, \omega)$

Input: number of underlying assets n, grid parameters $M = 2^m$ and R, time step k, maturity date T, riskless interest rate r, covariance matrix Σ, initial function $\tilde{w}_0$, boundary function $\tilde{g}$, smoothing parameters ν_1 and ν_2 for the multigrid method, cycling parameter ν for the full multigrid method, weighting factor ω for the weighted Jacobi method

Output: matrix $\mathbf{W} \in \mathbb{R}^{(2M+1)^n \times (K+1)}$ containing in its columns the vectors $\mathbf{W}^\ell$, $\ell = 0, \ldots, K$ (approximately) solving (4.1)–(4.3)

2: Initialise $\mathbf{W}$ as a zero matrix.

3: Generate a matrix `coord` of all grid point coordinates $\left(\frac{R}{M} \cdot \texttt{alpha}\right.$ with `alpha` according to Algorithm 5 with $\mathbf{y} = (1 : (2M+1)^n)^{\mathrm{T}}\big)$.

4: Evaluate $\tilde{w}_0$ at `coord` and hence assign the first column of $\mathbf{W}$ with the initial data.

5: Generate the boundary and interior array index vectors `bdind` and `intind` for the mesh $\Omega^{\mathbf{x}}_{R,h}$ (Algorithm 4).

6: Generate the `vals` vector for the matrix A_h and the `incrind` matrix according to Section 4.3.

7: **for** $i = 1 : K$ **do**

8: Generate the vector $\tilde{\mathbf{g}}^{\ell+1}$ by evaluating $\tilde{g}$ at `coord(bdind,:)` and at time ik, and supplementing the result by zero interior values.

9: Assemble the right-hand side vector of (4.3) with $\theta = 1$:
$\mathbf{RHS} \leftarrow \mathbf{W}(\texttt{intind}, i) - k\, \tilde{\mathbf{g}}^{\ell+1}(\texttt{incrind}) \cdot \texttt{vals}^{\mathrm{T}}$

10: $\mathbf{W}(:, i+1) \leftarrow \mathrm{FMG}^*\,(n, M, R, k, r, \Sigma, \mathbf{RHS}, \tilde{\mathbf{g}}^{\ell+1}, \nu_1, \nu_2, \nu, \omega)$

11: **end for**

12: **return** $\mathbf{W}$

Chapter 5

Numerical results

In this chapter, we will present some numerical results of the Black-Scholes solver derived in the previous chapters when applied to realistic option pricing problems, and thus assess the performance of the method and the implementation. Most computations (especially those where computation times are given) were done in MATLAB 7.7 (R2008b) running on a UNIX PC with an AMD Opteron Processor 248 (2 cores, 2.2 GHz, 1 MB CPU Cache, 8 GB RAM); the most costly computations were executed in MATLAB 7.8 (R2009a) on a UNIX PC with an Intel Xeon processor (8 cores, 2.5 GHz, 6144 KB CPU Cache, 32 GB RAM).

We will also compare the obtained results to results published in literature. As the most interesting test cases are option types, where a closed form solution for the price is known, we will give a summary of some explicit pricing formulae first.

5.1 Closed form option pricing formulae

5.1.1 Calls and puts on a single asset

For European call and put options on a single asset, an explicit valuation formula was already derived in [Black and Scholes, 1973]: let $C(S, E, \tau, r, \sigma)$ denote the price of a European call with exercise price E at time to maturity τ and current asset price S, when the riskless interest rate is r and the volatility of the underlying asset is σ, and $P(S, E, \tau, r, \sigma)$ the price of the corresponding put. Then

$$C(S, E, \tau, r, \sigma) = \quad SN(\ \ d_1) - E\mathrm{e}^{-r\tau}N(\ \ d_2), \tag{5.1}$$

$$P(S, E, \tau, r, \sigma) = -SN(-d_1) + E\mathrm{e}^{-r\tau}N(-d_2), \tag{5.2}$$

where

$$d_1 := d_1(S, E, \tau, r, \sigma) := \frac{\ln \frac{S}{E} + \left(r + \frac{\sigma^2}{2}\right)\tau}{\sigma\sqrt{\tau}}, \tag{5.3}$$

$$d_2 := d_2(S, E, \tau, r, \sigma) := \frac{\ln \frac{S}{E} + \left(r - \frac{\sigma^2}{2}\right)\tau}{\sigma\sqrt{\tau}} \ = d_1(S, E, \tau, r, \sigma) - \sigma\sqrt{\tau}, \tag{5.4}$$

and $N(\,.\,)$ denotes the cumulative distribution function of the standard normal distribution. There holds the *put-call-parity* relation

$$C(S, E, \tau, r, \sigma) + E\mathrm{e}^{-r\tau} = P(S, E, \tau, r, \sigma) + S. \tag{5.5}$$

5.1.2 Exchange options

Options that give the holder the right, at expiry, to exchange one asset for another were first examined by [Margrabe, 1978]. This type of option is called *exchange option* by [Wilmott, 1998] and *Margrabe option*, *exchangeable option*, or *outperformance option* in [BusinessDictionary.com, 2009]. Following [Wilmott, 1998], one can reduce the two-dimensional Black-Scholes problem for an option to exchange q_2 units of asset S_2 for q_1 units of asset S_1 to the one-dimensional Black-Scholes problem of an ordinary call option on the underlying $\frac{S_1}{S_2}$. Hence, with $\Sigma = \begin{pmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 \\ \rho_{12}\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$, one obtains,

$$\begin{aligned} V(S_1, S_2, q_1, q_2, \tau, r, \Sigma) &= q_1 S_2 \; C\Big(\frac{S_1}{S_2}, \frac{q_2}{q_1}, \tau, 0, \bar{\sigma}\Big) \\ &= q_1 S_1 \, N\Big(d_1(q_1 S_1, q_2 S_2, \tau, 0, \bar{\sigma})\Big) - q_2 S_2 \, N\Big(d_2(q_1 S_1, q_2 S_2, \tau, 0, \bar{\sigma})\Big) \end{aligned} \tag{5.6}$$

with

$$\bar{\sigma}^2 := \sigma_1^2 + \sigma_2^2 - 2\rho_{12}\sigma_1\sigma_2. \tag{5.7}$$

5.1.3 Options on the maximum or the minimum of n assets

[Stulz, 1982] derived a valuation formula for European call and put options on the minimum and the maximum of two assets. The resulting pricing formula involves evaluations of the bivariate normal cumulative distribution function. Subsequently, [Johnson, 1987] generalised this formula to options on the minimum and the maximum of an arbitrary number of assets. Let

$$N_n(x_1, \ldots, x_n, R) \tag{5.8}$$

denote the evaluation at $(x_1, \ldots, x_n)$ of the cumulative distribution function of the n-variate normal distribution with mean $\mathbf{0}$ and covariance matrix R. According to the formula by [Johnson, 1987], where we correct an obvious typographical error in the second term, the price of a call option on the maximum of n assets with uniform exercise price E is given by

$$\begin{aligned} C_{\max}(\mathbf{S}, E, \tau, r, \Sigma) = \\ & S_1 \, N_n\Big(d_1(S_1, E, \tau, r, \sigma_1),\, d_1(S_1, S_2, \tau, 0, \sigma_{12}), \ldots,\, d_1(S_1, S_n, \tau, 0, \sigma_{1n}), R_1\Big) \\ & + S_2 \, N_n\Big(d_1(S_2, E, \tau, r, \sigma_2),\, d_1(S_2, S_1, \tau, 0, \sigma_{12}), \ldots,\, d_1(S_2, S_n, \tau, 0, \sigma_{2n}), R_2\Big) \\ & + \quad \ldots \\ & + S_n \, N_n\Big(d_1(S_n, E, \tau, r, \sigma_n),\, d_1(S_n, S_1, \tau, 0, \sigma_{1n}), \ldots,\, d_1(S_n, S_{n-1}, \tau, 0, \sigma_{n-1\,n}), R_n\Big) \\ & - E\mathrm{e}^{-r\tau}\Big(1 - N_n\Big(-d_2(S_1, E, \tau, r, \sigma_1), -d_2(S_2, E, \tau, r, \sigma_2), \ldots, -d_2(S_n, E, \tau, r, \sigma_n), R\Big)\Big) \end{aligned} \tag{5.9}$$

with d_1 and d_2 as defined in (5.3)–(5.4),

$$\sigma_{ij}^2 := \sigma_i^2 - 2\rho_{ij}\sigma_i\sigma_j + \sigma_j^2, \tag{5.10}$$

and covariance matrices

$$R := \begin{pmatrix} 1 & \rho_{12} & \rho_{13} & \cdots & \rho_{1\,n-1} & \rho_{1n} \\ \rho_{12} & 1 & \rho_{23} & \cdots & \rho_{2\,n-1} & \rho_{2n} \\ \rho_{13} & \rho_{23} & 1 & \cdots & \rho_{3\,n-1} & \rho_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \rho_{1\,n-1} & \rho_{2\,n-1} & \rho_{3\,n-1} & \cdots & 1 & \rho_{n-1\,n} \\ \rho_{1n} & \rho_{2n} & \rho_{3n} & \cdots & \rho_{n-1\,n} & 1 \end{pmatrix} \tag{5.11}$$

and

$$R_i := \begin{pmatrix} 1 & \rho_{ij_1(i)k_1(i)} & \rho_{ij_2(i)k_2(i)} & \cdots & \rho_{ij_{n-2}(i)k_{n-2}(i)} & \rho_{ij_{n-1}(i)k_{n-1}(i)} \\ \rho_{ij_1(i)k_1(i)} & 1 & \rho_{ij_n(i)k_n(i)} & \cdots & \rho_{ij_{2n-4}(i)k_{2n-4}(i)} & \rho_{ij_{2n-3}(i)k_{2n-3}(i)} \\ \rho_{ij_2(i)k_2(i)} & \rho_{ij_n(i)k_n(i)} & 1 & \cdots & \rho_{ij_{3n-7}(i)k_{3n-7}(i)} & \rho_{ij_{3n-6}(i)k_{3n-6}(i)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & 1 & \rho_{ij_{n(n-1)/2}(i)k_{n(n-1)/2}(i)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & 1 \end{pmatrix} \tag{5.12}$$

for $i = 1, \ldots, n$, where the triple indexed correlation coefficients are defined by

$$\rho_{iij} := \rho_{iji} := \frac{\sigma_i - \rho_{ij}\sigma_j}{\sigma_{ij}} \qquad \text{for } i \neq j, \tag{5.13}$$

$$\rho_{ijk} := \rho_{ikj} := \frac{\sigma_i^2 - \rho_{ij}\sigma_i\sigma_j - \rho_{ik}\sigma_i\sigma_k + \rho_{jk}\sigma_j\sigma_k}{\sigma_{ij}\sigma_{ik}} \qquad \text{for } i,j,k \text{ mutually distinct} \tag{5.14}$$

and where the vectors $\mathbf{j}(i)$ and $\mathbf{k}(i)$ containing the $(j(i), k(i))$ combinations appearing in (5.12) can be found as follows: create an $n \times n$ matrix by vertically concatenating n copies of the row vector `[i,1:(i-1),(i+1):n]`. The lower triangular part of this matrix (excluding the diagonal), extracted column-wise, yields $\mathbf{j}(i)$, and the upper triangular part, extracted row-wise, yields $\mathbf{k}(i)$. Note that this procedure is similar to the one described in Section 4.3 to find all (i, j) combinations with $i < j$, only now the i-th column of the construction matrix is put in front. For illustration, the $\mathbf{j}(i)$ and $\mathbf{k}(i)$ vectors for $n = 4$ are shown in Table 5.1.

	construction matrix	$\begin{pmatrix} \mathbf{j}(i) \\ \mathbf{k}(i) \end{pmatrix}$
$i = 1$	$\begin{pmatrix} 1&2&3&4 \\ 1&2&3&4 \\ 1&2&3&4 \\ 1&2&3&4 \end{pmatrix}$	$\begin{pmatrix} 1&1&1&2&2&3 \\ 2&3&4&3&4&4 \end{pmatrix}$
$i = 2$	$\begin{pmatrix} 2&1&3&4 \\ 2&1&3&4 \\ 2&1&3&4 \\ 2&1&3&4 \end{pmatrix}$	$\begin{pmatrix} 2&2&2&1&1&3 \\ 1&3&4&3&4&4 \end{pmatrix}$
$i = 3$	$\begin{pmatrix} 3&1&2&4 \\ 3&1&2&4 \\ 3&1&2&4 \\ 3&1&2&4 \end{pmatrix}$	$\begin{pmatrix} 3&3&3&1&1&2 \\ 1&2&4&2&4&4 \end{pmatrix}$
$i = 4$	$\begin{pmatrix} 4&1&2&3 \\ 4&1&2&3 \\ 4&1&2&3 \\ 4&1&2&3 \end{pmatrix}$	$\begin{pmatrix} 4&4&4&1&1&2 \\ 1&2&3&2&3&3 \end{pmatrix}$

Table 5.1: The $(j(i), k(i))$ combinations appearing in (5.12) in the case $n = 4$.

The price of a call option with exercise price E on the minimum of n assets is given by

$$\begin{aligned}
C_{\min}(\mathbf{S}, E, \tau, r, \Sigma) = \\
&S_1\, N_n\Big(d_1(S_1, E, \tau, r, \sigma_1), -d_1(S_1, S_2, \tau, 0, \sigma_{12}), \ldots, -d_1(S_1, S_n, \tau, 0, \sigma_{1n}), \tilde{R}_1\Big) \\
&+ S_2\, N_n\Big(d_1(S_2, E, \tau, r, \sigma_2), -d_1(S_2, S_1, \tau, 0, \sigma_{12}), \ldots, -d_1(S_2, S_n, \tau, 0, \sigma_{2n}), \tilde{R}_2\Big) \\
&+ \quad \ldots \\
&+ S_n\, N_n\Big(d_1(S_n, E, \tau, r, \sigma_n), -d_1(S_n, S_1, \tau, 0, \sigma_{1n}), \ldots, -d_1(S_n, S_{n-1}, \tau, 0, \sigma_{n-1\,n}), \tilde{R}_n\Big) \\
&- E\mathrm{e}^{-r\tau} N_n\Big(d_2(S_1, E, \tau, r, \sigma_1),\, d_2(S_2, E, \tau, r, \sigma_2), \ldots,\, d_2(S_n, E, \tau, r, \sigma_n), R\Big), \qquad (5.15)
\end{aligned}$$

where the covariance matrices $\tilde{R}_i$ differ from the R_i defined by (5.12) in that the first $(n-1)$ triple indexed correlation coefficients (i.e. the ones in the first row and column of $\tilde{R}_i$) are taken negative. For $n = 2$, (5.15) becomes

$$\begin{aligned}
C_{\min}(S_1, S_2, E, \tau, r, \Sigma) = S_1\, N_2\Big(d_1(S_1, E, \tau, r, \sigma_1), -d_1(S_1, S_2, \tau, 0, \sigma_{12}), \begin{pmatrix} 1 & -\rho_{112} \\ -\rho_{112} & 1 \end{pmatrix}\Big) \\
+ S_2\, N_2\Big(d_1(S_2, E, \tau, r, \sigma_2), -d_1(S_2, S_1, \tau, 0, \sigma_{12}), \begin{pmatrix} 1 & -\rho_{221} \\ -\rho_{221} & 1 \end{pmatrix}\Big) \\
- E\mathrm{e}^{-r\tau} N_2\Big(d_2(S_1, E, \tau, r, \sigma_1),\, d_2(S_2, E, \tau, r, \sigma_2), \begin{pmatrix} 1 & \rho_{12} \\ \rho_{12} & 1 \end{pmatrix}\Big) \qquad (5.16)
\end{aligned}$$

with $\sigma_{12}^2 = \sigma_1^2 - 2\rho_{12}\sigma_1\sigma_2 + \sigma_2^2$, $\rho_{112} = \frac{\sigma_1 - \rho_{12}\sigma_2}{\sigma_{12}}$ and $\rho_{221} = \frac{\sigma_2 - \rho_{12}\sigma_1}{\sigma_{12}}$, which is consistent with equation (11) in [Stulz, 1982] corrected by the typographical error pointed out by [Johnson, 1987].

For the two-asset case, [Stulz, 1982] in addition proves the relations

$$\begin{aligned}
C_{\min}(S_1, S_2, 0, \tau, r, \Sigma) &= S_1 - V(S_1, S_2, 1, 1, \tau, r, \Sigma) \\
&= S_1 - S_1\, N\Big(d_1(S_1, S_2, \tau, 0, \sigma_{12})\Big) + S_2\, N\Big(d_2(S_1, S_2, \tau, 0, \sigma_{12})\Big), \qquad (5.17)
\end{aligned}$$

where V is the price of the exchange option to exchange one unit of asset S_2 for one unit of asset S_1 (see (5.6)),

$$C_{\max}(S_1, S_2, E, \tau, r, \Sigma) = C(S_1, E, \tau, r, \sigma) + C(S_2, E, \tau, r, \sigma) - C_{\min}(S_1, S_2, E, \tau, r, \Sigma), \qquad (5.18)$$

where C denotes the price of a call option on a single asset as given by (5.1), and the parity relations yielding the price of put options on the minimum and maximum of two assets, respectively,

$$P_{\min}(S_1, S_2, E, \tau, r, \Sigma) = E\mathrm{e}^{-r\tau} - C_{\min}(S_1, S_2, 0, \tau, r, \Sigma) + C_{\min}(S_1, S_2, E, \tau, r, \Sigma), \qquad (5.19)$$

$$P_{\max}(S_1, S_2, E, \tau, r, \Sigma) = E\mathrm{e}^{-r\tau} - C_{\max}(S_1, S_2, 0, \tau, r, \Sigma) + C_{\max}(S_1, S_2, E, \tau, r, \Sigma). \qquad (5.20)$$

5.1.4 Other cases

[Boyle et al., 1989] mention that a closed form solution can be found for the price of options on the geometric average of n assets, yet they do not state this solution.

Referring to the closed form solution published by [Johnson, 1987] for options on the maximum and the minimum of n assets (see Section 5.1.3), [Barraquand, 1995] says,

> *To the best of our knowledge, the above problem of option pricing on the maximum or minimum of several assets is the only case reported in the literature where solutions to high-dimensional (say, more than 5) problems have been developed.*

5.2 Choice of parameters

Apart from the problem parameters (number of underlying assets, volatilities and correlation coefficients, option type, exercise price, maturity date, interest rate), a number of computational parameters have to be specified.

First, the parameter θ in the θ-method determines the type of discretisation of the PDE. In Section 2.3.1, we argued why using the explicit Euler method or an implicit method with $\theta < \frac{1}{2}$ is not recommendable. We will use both $\theta = \frac{1}{2}$ (Crank-Nicolson method) and $\theta = 1$ (implicit Euler method) and compare the performance of these two methods.

Depending on the range of asset prices where the option price is of interest, the size of the computational domain has to be chosen. Suppose, for example, that we are interested in the option price for asset prices between \$0 and \$40. $S_i = 40$ corresponds to $x_i = \ln 40$, hence we need $R \geq \ln 40$. According to the results of Section 1.4.3, the localisation error decreases with R. The estimate (1.93) might be used to determine R such that the localisation error is less than some predefined tolerance within the domain $\overline{\Omega_{\tilde{R}}}$ with $\tilde{R} = \ln 40$. It turns out, however, that (1.93) is extremely conservative; we will experimentally show that taking

$$R = 2\tilde{R} \qquad \text{or} \qquad R = 2 \ln E \tag{5.21}$$

yields good results in practice. As $\tilde{R} \geq \ln E$ in general and $E > 4$ in our experiments below, this is in turn more conservative than the rule of thumb (1.92) to define the far field boundary at $S_{\max}$ equal to $3E$ or $4E$, i.e. $R = \ln S_{\max} = \ln E + \ln 3$ or $\ln E + \ln 4$, and we can thus expect our choice of R to be large enough to keep the localisation error acceptably small. We will in addition numerically examine how fast the the localisation error decays with R. If we choose $R = 2 \ln E$, $S_i = E$ is a grid point for any admissible mesh width. As the initial function is not smooth at these points for certain option types, it may be advantageous for the numerical solution to have them as grid points; this is recommended, e.g., by [Seydel, 2004]. We will come back to this question in the experiments below.

The localisation error also depends on the choice of the boundary function; in Section 1.4.4, we showed that

$$\tilde{g}_1(\mathbf{x}, \tau) = \tilde{w}_0(\mathbf{x}) = V_0(\mathrm{e}^{x_1}, \ldots, \mathrm{e}^{x_n}) \qquad \text{or} \qquad \tilde{g}_2(\mathbf{x}, \tau) = \mathrm{e}^{-r\tau}\, V_0\big(\mathrm{e}^{x_1 + r\tau}, \ldots, \mathrm{e}^{x_n + r\tau}\big) \tag{5.22}$$

are appropriate. We will investigate below which is the better choice. A priori, economic reasoning makes $\tilde{g}_2$ more plausible, whereas $\tilde{g}_1$ has the advantage of being independent of time. In the latter case, the evaluation vector $\tilde{\mathbf{g}}$ and its contribution $-kA_h^{(\mathrm{ib})}\tilde{\mathbf{g}}(\texttt{bdind})$ to the right-hand side only have to be calculated once, and the computational effort can thus be reduced if Algorithm 10 or 13 is accordingly modified.

Once the computational domain has been set, appropriate space and time mesh widths have to be chosen. In our analysis of the θ-method, we found in Proposition 13 that the Crank-Nicolson method is uniformly solvable if

$$h \leq \min_{i=1,\ldots,n} \frac{\sigma_i^2}{\left|\frac{\sigma_i^2}{2} - r\right|} \qquad \text{and} \qquad \frac{k}{h^2} < \frac{1}{\theta\left(\sum_{i<j} \sigma_i \sigma_j |\rho_{ij}|\right)}. \tag{5.23}$$

The same conditions also imply that the weighted Jacobi method with any $\omega \in (0, 1]$ converges, as we have seen in Proposition 15. The first of these conditions is not very restrictive in general.

The second one, however, imposes a bound on $\frac{k}{h^2}$ unless all assets are uncorrelated, and thus requires the time step to be very small compared to the spatial mesh width. However, we do not know whether the sufficient conditions (5.23) are necessary as well. As for the further convergence criteria of consistency and stability, we mentioned in Section 2.3.1 that the implicit Euler method for the transformed Black-Scholes PDE is consistent of order $O(h^2)+O(k)$, and the Crank-Nicolson method even of order $O(h^2) + O(k^2)$. Furthermore, both methods are stable with respect to the ℓ^2-norm under no further conditions on the mesh widths. However, we also found that the truncated Black-Scholes problem in the domain $\Omega_R \times (0, T]$ is not well-posed in the sense required for the Lax Equivalence Theorem, so that we were not able to prove convergence of our numerical methods. We will experimentally examine the convergence properties below.

Furthermore, the multigrid functions require the multigrid parameters ν_1 (number of pre-smoothing operations on each grid), ν_2 (number of post-smoothing operations on each grid), ν (number of V-cycles performed on each grid once an initial guess has been established by nested iteration), and the weighting factor ω in the weighted Jacobi method. In the context of the multigrid method, ω is optimally chosen such that the high-frequency modes are damped most efficiently (cf. Sections 3.1 and 3.2.1). We will use the theoretically optimal weighting factor given by (3.24), $\omega_{\text{opt}} = \frac{2n}{2n+1}$. The parameters ν_1, ν_2 and ν determine the accuracy as well as the computational complexity of the full multigrid V-cycle method; the higher they are chosen, the more accurate is the result and the higher is the computation time in general. Thus, a trade-off has to be found. In our practical experiments, we will find suitable parameters such that *one* full multigrid V-cycle yields an acceptable result as follows: We will always use the weighting factor ω_{opt} given by (3.24) and at the same time increase the parameters ν_1, ν_2 and ν until no significant decrease in the error is achieved any more. According to [Hackbusch, 1985, e.g., Sections 2.6.2 and 4.3], the performance of the multigrid method depends primarily on the overall number of smoothing steps $\nu_1 + \nu_2$ rather than on the distribution of pre- and post-smoothing steps; we will use $\nu_1 = \nu_2$.

5.2.1 Overview of numerical experiments

We will now give a brief overview of the test problems that we will present below, and motivate the choices of parameters. We will numerically price options on 1, 2, and 3 underlying assets for which in most cases, the exact solution is known and the error in the numerical solution can thus be calculated. As the solution of the Black-Scholes problem represents option prices, the maximum norm is a more appropriate error measure than the ℓ^2-norm (cf. also [Kangro and Nicolaides, 2000]). Note further that an accuracy corresponding to 10^{-3} of the respective currency unit is in general sufficient.

Our choice of problem parameters is inspired by reference results available in literature. Typically, volatilities between 20% and 35% are assumed for each asset. [Boyle and Tse, 1990] present numerical results for values of European calls on the maximum and the minimum of three assets, using a range of symmetric and unsymmetric volatility and correlation assumptions. The case where the highest errors are observed is the most unsymmetric case with $\sigma_1 = 0.25$, $\sigma_2 = 0.3$, $\sigma_3 = 0.35$, $\rho_{12} = \rho_{23} = 0.6$, and $\rho_{13} = 0.4$ ([Boyle and Tse, 1990, Table 5]). As for the other problem parameters, a riskless interest rate of 10%, a maturity date in one year's time (i.e. $T = 1$), and a uniform exercise price of \$30 (i.e. $E_i = 30 =: E$ for all i) are assumed, and the option price is computed for asset prices of \$40 each. The same problem is considered in [Engelmann and Schwendner, 1998, Tables 1 and 2]. We will use the same parameters for our three-dimensional test case, and consider the option price for asset prices in the domain $[\frac{1}{40}, 40]^3$ (i.e. $\tilde{R} = \ln 40$ in (5.21))

such that we can compare the results. [Engelmann and Schwendner, 1998] consider in addition a case with individual exercise prices for each asset where no exact solution is known, albeit evaluated at asset price combinations which cannot be generated as grid points of uniform meshes as used in our algorithms. Hence, we will not present numerical results for this case although our solver can deal with this case when the pay-off function is accordingly defined.

Further numerical results for three-asset options are given in [Boyle et al., 1989, Table 2] for European call and put options on the maximum, the minimum, the geometric average, and the arithmetic average, respectively. The problem parameters used there are $\sigma_i = 0.2$ for all i, $\rho_{ij} = 0.2$ for all $i \neq j$, $E = 100$, $r = 0.1$, and $T = 1$. We will present results of our solver for the two types of basket options as well.

In [Barraquand and Martineau, 1995, Tables 1–5], results for European and American single-asset calls and puts, and options on the maximum of 3 and 10 assets with equal correlation coefficients are presented. Here, longer times to expiry of 4 and 7 years are considered as well. However, varying T in a time-stepping algorithm does not seem to provide more insight than varying the step size k at fixed T, hence we will restrict to the case $T = 1$. Furthermore, options often have a life-span of less than a year. As for the ten-dimensional test case presented in [Barraquand and Martineau, 1995], we cannot repeat this experiment as the computational effort and memory requirement are too high. The same holds for the valuation of quality options in up to 50 dimensions presented in [Boyle and Tse, 1990, Table 1]. Note, however, that all numerical results cited so far refer to algorithms where the option price can only be evaluated at one specific point $(\mathbf{S}, \tau)$, whereas our algorithm simultaneously yields results on a whole grid of $(\mathbf{S}, \tau)$ combinations. This difference has to be considered when comparing the complexity. We will come back to the pros and cons of the two algorithm types in Section 6.

In one and two dimensions, we will consider single-asset calls and puts, exchange options, and calls on the maximum and minimum of two assets. We choose similar problem parameters as in the three-dimensional test case from [Boyle and Tse, 1990, Table 5] and [Engelmann and Schwendner, 1998, Tables 1 and 2] mentioned above, i.e. volatilities of 25% and 30%, $r = 0.1$, $T = 1$, $E = 30$ where applicable, and $\tilde{R} = \ln 40$. We assume a higher correlation coefficient of $\rho = 0.9$ in the two-dimensional experiments, however; economically, such a high positive correlation may occur for shares of companies in the same branch of trade, for instance.

The test cases we consider are summarised below, together with full multigrid V-cycle parameters that were empirically found suitable in the way mentioned in the previous section. In general, for the accuracy required here, using $\nu = 1$ is sufficient, and the computation time is less than for higher values of ν even when the latter are used with lower numbers ν_1 and ν_2 of smoothing steps.

$n = 1$: **European call and put options**
Section 5.3

exercise price:	\$30
time to maturity:	one year
asset volatility:	$\sigma = 0.3$
interest rate:	10%
domain of interest:	$S \leq \$40$
types of θ-methods used:	Crank-Nicolson and implicit Euler
size of computational domain:	$R = 2 \ln 40$ and $R = 2 \ln 30$

grid sizes:	M varied between 32 and 4096,
	k varied between 0.1 and 0.0001
boundary conditions:	$\tilde{g}_1$ and $\tilde{g}_2$
Jacobi weighting factor:	$\omega_{\text{opt}} = \frac{2}{3}$
multigrid parameters:	$\nu_1 = \nu_2 = 7$, $\nu = 1$

for the examination of the localisation error:

exercise price:	\$1
domain of interest:	$\tilde{R} = 0.005$
types of θ-methods used:	implicit Euler
size of computational domain:	R varied between 0.04 and 0.35
grid sizes:	sufficiently fine to see no dependence on M and k
boundary conditions:	$\tilde{g}_2$

$n = 2$: **Exchange option**
Section 5.4.1

exchange ratio:	$q_1 = q_2 = 1$
time to maturity:	one year
asset volatilities:	$\sigma_1 = 0.25$, $\sigma_2 = 0.3$
correlation coefficient:	$\rho = 0.9$
interest rate:	10%
domain of interest:	$\tilde{R} = 0.005$
types of θ-methods used:	implicit Euler
size of computational domain:	R varied between 0.005 and 0.1
grid sizes:	sufficiently fine to see no dependence on M and k
boundary conditions:	$\tilde{g}_1 = \tilde{g}_2$
Jacobi weighting factor:	$\omega_{\text{opt}} = 0.8$
multigrid parameters:	$\nu_1 = \nu_2 = 10$, $\nu = 1$

European call options on the maximum and the minimum of two assets
Section 5.4.2

exercise price:	\$30 for both assets
time to maturity:	one year
asset volatilities:	$\sigma_1 = 0.25, \sigma_2 = 0.3$
correlation coefficient:	$\rho = 0.9$
interest rate:	10%
domain of interest:	$S_i \leq$ \$40
types of θ-methods used:	Crank-Nicolson and implicit Euler
size of computational domain:	$R = 2\ln 40$ and $R = 2\ln 30$
grid sizes:	M varied between 16 and 1024,
	k varied between 0.1 and 0.001
boundary conditions:	$\tilde{g}_1$ and $\tilde{g}_2$
Jacobi weighting factor:	$\omega_{\text{opt}} = 0.8$
multigrid parameters:	$\nu_1 = \nu_2 = 10$, $\nu = 1$

$n = 3$: **European call options on the maximum and the minimum of three assets**

Section 5.5.1

exercise price:	\$30 for each asset
time to maturity:	one year
asset volatilities:	$\sigma_1 = 0.25$, $\sigma_2 = 0.3$, $\sigma_3 = 0.35$
correlation coefficients:	$\rho_{12} = \rho_{23} = 0.6$, $\rho_{13} = 0.4$
interest rate:	10%
domain of interest:	$S_i \leq \$40$
types of θ-methods used:	implicit Euler
size of computational domain:	$R = 2 \ln 40$
grid sizes:	M varied between 8 and 64, k varied between 0.1 and 0.0001
boundary conditions:	$\tilde{g}_2$
Jacobi weighting factor:	$\omega_{\text{opt}} = \frac{6}{7}$
multigrid parameters:	$\nu_1 = \nu_2 = 12$, $\nu = 1$
reference results:	[Boyle and Tse, 1990, Table 5], [Engelmann and Schwendner, 1998, Tables 1 and 2]

European call options on the arithmetic and geometric average

Section 5.5.2

exercise price:	\$100
time to maturity:	one year
asset volatilities:	$\sigma_1 = \sigma_2 = \sigma_3 = 0.2$
correlation coefficients:	$\rho_{12} = \rho_{23} = \rho_{13} = 0.2$
interest rate:	10%
domain of interest:	$S_i \leq \$100$
types of θ-methods used:	implicit Euler
size of computational domain:	$R = 2 \ln 100$
grid sizes:	M varied between 8 and 128, k varied between 0.1 and 0.0001
boundary conditions:	$\tilde{g}_1$
Jacobi weighting factor:	$\omega_{\text{opt}} = \frac{6}{7}$
multigrid parameters:	$\nu_1 = \nu_2 = 12$, $\nu = 1$
reference results:	[Boyle et al., 1989, Table 2]

5.3 Numerical results for single-asset options

Although our solver is primarily designed for multi-asset options, it works for the one-dimensional case as well, and as this case requires the least computational effort, it is well suited for investigating the convergence properties of our method. We will present results for European call and put options with the parameters listed in the previous section. The exact option prices can be

calculated from (5.1) and (5.2), respectively; the results can be directly obtained in MATLAB by a call of the function `blsprice` from the MATLAB Financial Toolbox (see [MATLAB 7.7.0 Help, 2008]). Thus, we can determine the error of the result obtained by our solver. We assume that we are interested in the results for asset prices up to \$40, hence we mainly consider the error in the domain $[\frac{1}{40}, 40] \times [0, T]$.

In Tables 5.3 and 5.4, results obtained with the *Crank-Nicolson method* are shown for domain sizes $R = 2 \ln 40$ and $R = 2 \ln 30$, respectively, for different mesh widths and time steps. We list the errors $\max_{\mathbf{x}_{\boldsymbol{\alpha}} \in \Omega_R,\, 0 \le \ell \le K} |W_{\boldsymbol{\alpha}}^{\ell} - w_{\boldsymbol{\alpha}}^{\ell}|$ and $\max_{\mathbf{x}_{\boldsymbol{\alpha}} \in [\frac{1}{40}, 40],\, 0 \le \ell \le K} |W_{\boldsymbol{\alpha}}^{\ell} - w_{\boldsymbol{\alpha}}^{\ell}|$, where $\mathbf{W}^{\ell}$, $\ell = 0, \ldots, K$ denotes the set of numerical solution vectors and $\mathbf{w}^{\ell}$, $\ell = 0, \ldots, K$ the set of evaluations of the exact solution of the untruncated Black-Scholes problem. In addition, Tables 5.3 and 5.4 allow to compare the two choices of boundary functions (5.22).

The first thing we observe from Tables 5.3 and 5.4 is the fact that we do obtain results that agree up to less than one cent with the exact solution for asset prices in $[\frac{1}{40}, 40]$ and uniformly at all time steps for certain (h, k) combinations. However, for k larger than 0.0001, we observe that the error first decreases with h, attains a minimum and then increases again. This may indicate that the relation between k and h is critical for the convergence of the Crank-Nicolson method after all. The conditions (5.23) for uniform solvability and convergence of the weighted Jacobi method do not impose any bound on $\frac{k}{h^2}$ in the one-dimensional case as the denominator is zero then. We stated in Section 2.3.1 that such a constraint is usually necessary for the stability of θ-schemes with $\theta < \frac{1}{2}$ but not for the Crank-Nicolson scheme; however, this result refers to stability in the ℓ^2-norm, whereas we are considering the maximum norm of the error in Tables 5.3 and 5.4 as this is the more appropriate error measure for the Black-Scholes problem. Convergence in the maximum norm may require more restrictive conditions on k and h as both tend to zero. If we take a closer look at the (k, h) combinations that yield the best results in Tables 5.3 and 5.4, i.e. $(0.1, 128)$, $(0.01, 256)$, $(0.001, 1024)$, and $(0.0001, 4096)$, we find that $\frac{k}{h^2}$ is of the same order of magnitude for all of them and bounded by 31 for $R = 2 \ln 40$ and 37 for $R = 2 \ln 30$, respectively. This suggests a convergence condition of the form $\frac{k}{h^2} \le C$ after all.

When comparing the two choices of boundary functions, we observe that $\tilde{g}_1$ entails a higher error within the full computational domain than $\tilde{g}_2$ in all cases. Furthermore, the error associated with $\tilde{g}_1$ is independent of the mesh; a closer examination reveals that this error is always attained at the grid point $\tau = 1$, $S = S_{\max}$ for calls and $\tau = 1$, $S = \frac{1}{S_{\max}}$ for puts, i.e. on the extreme boundary, and it therefore reflects an error in the prescribed boundary conditions rather than an error in the actual result of the linear system. In contrast, if we use the boundary function $\tilde{g}_2$, the maximum error on the whole computational domain is much lower, and for sufficiently small h, it coincides with the error in the interior domain. This proves that $\tilde{g}_2$ better reflects the true behaviour of the solution at the boundaries. We predicted that $\tilde{g}_2$ might be better in Section 1.2.1. However, we have to bear in mind that we are actually only interested in the numerical results for $S \in [\frac{1}{40}, 40]$, and it turns out that the maximum error in this interior domain is indistinguishable for both choices of boundary functions in all cases, confirming the assertion in Section 1.2.1 that the boundary conditions have no significant influence on the solution in the interior of the domain for this type of problem. Thus, using $\tilde{g}_1$ with an algorithm exploiting its time independence is empirically advisable.

The maximum error within $[\frac{1}{40}, 40] \times [0, T]$ is in most cases attained at the first non-zero time step and at an asset price near $S = 30$, i.e. at the point where the pay-off function is not smooth; as

we predicted, this is the most critical region for the numerical solution. The quality of the numerical solution at different time steps is illustrated in Figure 5.1 as well. When we compare the results obtained with $R = 2\ln 40$ and $R = 2\ln 30$, we find that $R = 2\ln 40$ yields slightly more accurate results in almost all cases. Apparently, there is no gain in placing grid points at the location of the discontinuity in the derivative of the initial function. Further increasing R does not seem to have an influence on the result any more; with $R = 4\ln 40$, we obtain identical results in the domain of interest as with $R = 2\ln 40$ at corresponding mesh widths (results not shown here).

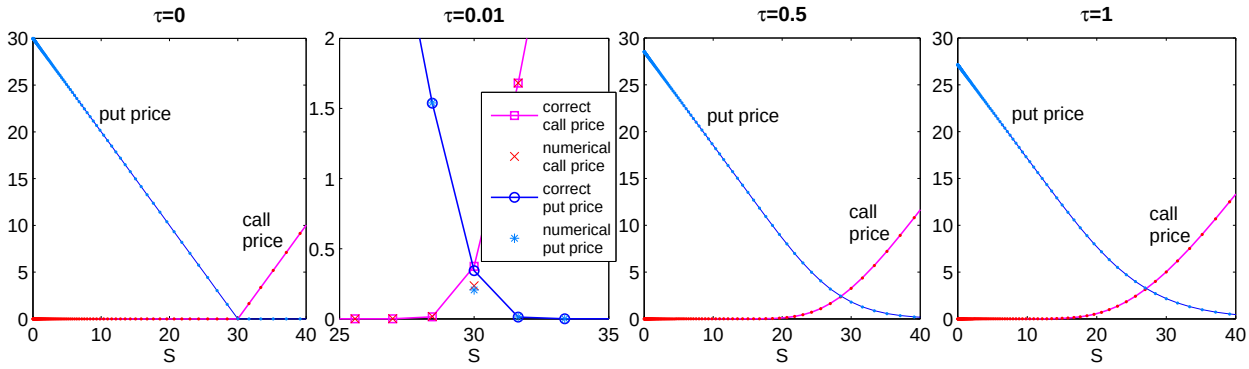

Figure 5.1: Plot of the solution for single-asset call and put options at different time steps. Dots correspond to the numerical solution obtained with the Crank-Nicolson method, whereas lines interpolate the exact solution obtained by `blsprice` unless otherwise specified. The parameters used are $E = 30$, $T = 1$, $\sigma = 0.3$, $r = 0.1$, $M = 128$, $k = 0.01$, $R = 2\ln 30$, $\nu_1 = \nu_2 = 7$, $\nu = 1$, and $\omega = \frac{2}{3}$.

Next, we repeat the same experiments but using the *implicit Euler method* instead of the Crank-Nicolson scheme. The results are shown in Tables 5.5 and 5.6.

As for the dependence of the error on the boundary function and the size of the computational domain, we observe the same behaviour as for the Crank-Nicolson method above. However, the behaviour of the maximum norm error with respect to the grid sizes is significantly different now: apart from minor irregularities, the error decreases whenever the spatial mesh width h or the time step k is reduced, until a level is reached where it remains constant. This is the behaviour we expect from a convergent method. Whereas the Crank-Nicolson method seems to converge only if $\frac{k}{h^2}$ remains bounded, the implicit Euler method appears insensitive to the relation between h and k, the so-called *refinement path*. This constitutes a significant advantage of the implicit Euler method over the Crank-Nicolson method. For a graphical comparison of the two methods, see also Figure 5.2.

The observed error in the numerical solution with respect to the exact solution of the Black-Scholes model (1.10)–(1.11) derives from several origins. Known error sources are the localisation error associated with the truncation of the problem to a bounded domain and imposing artificial boundary conditions, the discretisation error of the finite-difference discretisation of the PDE (i.e. the error in the θ-scheme), the error in the iterative solution of the corresponding linear system, and rounding errors. Some of these can be approximately quantified. We examined the localisation error in detail in Section 1.4.3. Experimentally, we found that the choice $R = 2\ln \tilde{R}$ where $\overline{\Omega_{\tilde{R}}}$ is the

domain of interest produces acceptable results, in accordance with the empirical rule (1.92); furthermore, we observed that further increasing R had no influence on the result. This suggests that the localisation error is kept well under control; we will analyse the localisation error in detail at the end of this section. As for the error associated with the boundary conditions, we observed that the influence of different boundary functions is indiscernible in the interior of the domain, hence this error can be neglected. The error in the solution of the linear system by the multigrid method can be quantified by calculating a reference solution using a direct solution method that produces the accurate result (subject to rounding errors); in the one-dimensional case, the computational effort of directly solving the linear system is not excessive. We present some results in Table 5.7. These demonstrate that when using the parameters $\nu = 1$, $\nu_1 = \nu_2 = 7$ as in all numerical experiments above, the multigrid error is negligible with respect to the overall error in our test problems. With a higher number of V-cycles and smoothing operations on each grid, e.g. $\nu = 3$, $\nu_1 = \nu_2 = 15$ as in the lower part of Table 5.7, one can even achieve an accuracy of the multigrid method that is comparable to the direct solution of the linear system. For the first choice of multigrid parameters in Table 5.7, we observe that for fixed k, the error initially decreases and eventually increases with M again. This can be attributed to the trade-off between increasing theoretical accuracy and growing impact of rounding errors when h is reduced. For the higher multigrid parameters, the accuracy is very good even for the coarsest grids, and we observe only the effect of rounding errors as h becomes smaller.

The dominant error seems to be the discretisation error, i.e. the error by which the exact solution of the θ-scheme differs from the solution of the truncated differential problem. For problems that are well-posed in the sense required for the Lax Equivalence Theorem (see the Remark to Definition 4), this error is of the same order as the truncation error (see the Remark to Theorem 14), i.e. $O(h^2) + O(k)$ for the implicit Euler method and $O(h^2) + O(k^2)$ for the Crank-Nicolson method. However, as the usual convergence theory does not apply to our special problem (see the argument in Section 2.3.1), we cannot predict the order of convergence in this case. In Figure 5.2, we examine the experimentally observed errors within $[\frac{1}{40}, 40] \times [0, T]$ from Tables 5.3 and 5.5. Note that this is not the error with respect to the *truncated* but the original Black-Scholes problem; we do not know the exact solution of the truncated problem, however, the difference should be small within the interior domain $[\frac{1}{40}, 40] \times [0, T]$ according to our observations so far.

In the upper four plots of Figure 5.2, we plot the maximum norm error at fixed k against M and at fixed M against k, respectively, for the Crank-Nicolson method and the implicit Euler method, in a double logarithmic scale. The slope of the obtained curves should indicate the order of convergence with respect to the respective mesh parameter. For the Crank-Nicolson method, however, the curves are not monotonically decreasing but reaching a minimum and then increasing again. We already observed this phenomenon in Table 5.3 and deduced the empirical convergence condition $\frac{k}{h^2} \leq C$: When h becomes so small compared to the fixed time step k that the condition $\frac{k}{h^2} \leq C$ is violated, the error increases again. However, this interpretation does not explain the form of the curves at fixed M. In any case, this phenomenon does not occur in the implicit Euler method. Here, the error curves are decreasing and eventually levelling off. As for the slopes of the curves, the result in the first Crank-Nicolson plot is ambiguous. The observed order lies between $O(1/M)$ and $O(1/M^2)$, i.e. $O(h)$ and $O(h^2)$. Some curve sections seem to display quadratic order, but the slope of the full decreasing curve sections is hard to judge. It is possible that the lack of regularity of the option pricing problem causes the actual convergence order to be less than the theoretical order of h^2. As for the order with respect to time, this is clearly the case: for

sufficiently regular problems, the Crank-Nicolson method is of order k^2, yet the second plot in Figure 5.2 does definitely not display second order convergence but instead a good agreement with the reference curve indicating an order of k. For the implicit Euler method, we observe an order of $O(h)$ with respect to space, and an order eventually reaching $O(k^{1/2})$ with respect to time, hence the implicit Euler method does not reach its theoretical convergence order of $O(h^2) + O(k)$ either. The higher consistency order of the Crank-Nicolson method is visible despite the lack of regularity of the problem. Nevertheless, the implicit Euler method is preferable due to its insensitivity to the refinement path.

The last two plots of Figure 5.2 contain the maximum norm as well as the discrete ℓ^2-norm (2.49) of the errors within the interior domain, plotted against M at fixed k. In practice, the maximum norm is more significant as mentioned above, however, it is of mathematical interest to compare the behaviour of the error in the two norms. By (2.54), the ℓ^2-norm is the weaker of the two, hence the convergence conditions for this norm might be less restrictive. Figure 5.2 illustrates, however, that the error curves for the ℓ^2-norm are similar in shape as the ones for the maximum norm; in fact, the corresponding pairs of curves are almost parallel. This indicates that the quotient of the two norms is approximately constant. In particular, the order of convergence with respect to both norms is the same, and the error of the Crank-Nicolson method has the same strange behaviour when measured in the ℓ^2-norm as in the maximum norm.

It remains to investigate the behaviour of the localisation error. By Estimate (1.112) from Section 1.4.3, the localisation error within the domain $\Omega_{R/2} \times [0, T]$ is controlled by a bound of the form $C\mathrm{e}^{-\frac{\bar{\mu}}{2}R}$ in the one-dimensional case, albeit not with respect to the maximum norm that we are interested in. In order to examine this, we try to extract the localisation error by choosing a very small computational domain where the localisation error should be notable, and fine meshes to keep the discretisation error small. The choice $M = 1024$, $k = 0.001$ turned out appropriate for all domain sizes R we used here. For the exact problem parameters, refer to the overview in Section 5.2.1. The results are plotted in Figure 5.3 together with reference curves representing decay laws of $O(1/R)$ as predicted by (1.91), $\mathrm{e}^{O(-R)}$ as predicted by (1.112), and $\mathrm{e}^{O(-R^2)}$. In the first plot, the logarithms of the maximum norm errors are plotted versus R such that the curve should be linear if the errors are proportional to $\mathrm{e}^{O(-R)}$; in the second plot, we use R^2 on the x-axis such that the curve is linear if the errors are proportional to $\mathrm{e}^{O(-R^2)}$. The result is rather clear: the error obviously decays faster than $1/R$ and even faster than exponentially with respect to R; the good accordance with the reference curve of $\mathrm{e}^{-30\,R^2}$ indicates a decay that is almost exponential in R^2. This result is better than the estimates given in Section 1.4.3 and confirms that the localisation error in practical numerical computations is very small.

5.4 Numerical results for options on two underlying assets

5.4.1 Exchange options

We start our two-dimensional experiments by investigating the behaviour of the localisation error measured in the maximum norm. Exchange options are convenient for this experiment because the exact solution (5.6) can be computed fast. The localisation error estimate that we derived for the two-dimensional case is (1.91) which predicts an algebraic decay if the maximum error on the boundary of the domain is bounded independently of the domain size R. However, this is not the

case for exchange options, so we cannot even apply (1.91). Note further that the two choices of boundary functions (5.22) are the same for exchange options.

In Figure 5.4, we present semilogarithmic plots of the numerically observed localisation errors (for the parameters specified in the overview in Section 5.2.1) versus R and R^2, respectively. We immediately see that the decay is not like $1/R$ but in fact at least exponentially in R; the error curve is in good accordance with the reference curve of e^{-4R}. Even though the localisation error decays more slowly than in the one-dimensional case where we observed an order of approximately $\mathrm{e}^{O(-R^2)}$, this result is highly satisfactory as it indicates that the localisation error decreases so fast that it should not interfere in practice.

Figure 5.5 depicts the numerical solution for $R = 0.01$, i.e. $\overline{\Omega_R} \approx [0.99005, 1.01005]$, and its point-wise error within $\overline{\Omega_{0.005}} \approx [0.99501, 1.00501]$. In the last plot, both the exact and the numerical solution are shown on the whole domain. The discrepancy is significant, the numerical solution is more curved than the exact one and shifted by a considerable distance, which has to be attributed primarily to the boundary conditions.

5.4.2 Options on the maximum and the minimum of two assets

The second two-dimensional test problem will be European call options on the maximum and the minimum of two assets with parameters as specified in the overview in Section 5.2.1. With the chosen parameters, the conditions (5.23) for uniform solvability and convergence of the weighted Jacobi method become

$$h \leq 0.909 \qquad \text{and} \qquad \frac{k}{h^2} < 29.6; \tag{5.24}$$

we will examine below whether a bound on $\frac{k}{h^2}$ is in fact necessary for convergence of the Crank-Nicolson method and irrelevant for convergence of the implicit Euler method as in the one-dimensional case.

The exact prices of call options on the minimum and the maximum of two assets are given by (5.16) and (5.18), respectively, which we implement in MATLAB to generate the exact solutions. Note, however, that the valuation formulae involve the cumulative distribution function of the bivariate normal distribution which cannot be analytically evaluated. In MATLAB, evaluations of the multivariate normal cumulative distribution function are computed by the function `normcdf` using adaptive quadrature with a default absolute error tolerance of 10^{-8} for dimensions $n = 2$ and $n = 3$, and a quasi-Monte Carlo integration algorithm with a default absolute error tolerance of 10^{-4} for $n \geq 4$ (see [MATLAB 7.7.0 Help, 2008]). We thus have to bear in mind that the computed "exact" solution is obtained as a numerical approximation as well; however, the result will still be sufficiently accurate to be regarded as the exact solution.

Let us briefly comment on the implementation of the initial and boundary function for this test problem. In Section 4.2, we specified certain requirements regarding the simultaneous evaluation of matrix arguments. The initial function for a call on the maximum of two assets with uniform exercise price,

$$\tilde{w}_0(x_1, x_2) = V_0(\mathrm{e}^{x_1}, \mathrm{e}^{x_2}) = \max\left\{ \max\{\mathrm{e}^{x_1}, \mathrm{e}^{x_2}\} - E,\, 0 \right\} \tag{5.25}$$

according to Table 1.1, can be implemented as

$$\texttt{w0} \;=\; @(\texttt{x})\ \texttt{max}(\texttt{max}(\texttt{exp}(\texttt{x}), [], 2) - \texttt{E}, 0), \tag{5.26}$$

where the `@` construction defines `w0` to be a function handle representing a function in the variable `x` (see [MATLAB 7.7.0 Help, 2008]). If `x` is a matrix, `exp(x)` is the component-wise exponential

function, and `max(exp(x),[],2)` is the column vector containing the largest entry in each row of `exp(x)`. Finally, the outer `max` function yields the column vector containing in each row the component of the previous vector diminished by E or zero, whichever is larger. Hence, (5.26) acts on matrix arguments precisely as desired. Once `w0` is available, the boundary functions $\tilde{g}_1$ and $\tilde{g}_2$ can simply be defined as

$$\texttt{g1} = \texttt{@(x,t) w0(x)} \qquad \text{and} \qquad \texttt{g2} = \texttt{@(x,t) exp(-r*t)*w0(x+r*t)}. \tag{5.27}$$

Numerical results obtained with the implicit Euler method with $R = 2\ln 40$ and $R = 2\ln 30$ and a range of meshes are presented in Tables 5.8 and 5.9, and plots of the numerical solution and the point-wise error are shown in Figures 5.6 and 5.7. Note that due to the higher computational complexity in two dimensions, we cannot refine the meshes to such an extent as in the one-dimensional test problem. Furthermore, as the calculation of the exact solution turns out to be of significantly higher computational effort than our solver, we only compute it for the time steps $\tau = 0, 0.1, 0.2, \ldots, 1$ and compare it to the numerical solution only at these points in time in Tables 5.8 and 5.9. Results for the Crank-Nicolson method are not shown in tables, but the observed errors are plotted in Figure 5.8.

The quality of the results is slightly different to the one-dimensional case. In Tables 5.8 and 5.9, we observe maximum norm errors that largely decrease monotonically with h within $[\frac{1}{40}, 40]^2 \times (0 : 0.1 : 1)$ at fixed k. For fixed h, we observe constant errors for all k for large h; however, the error seems to decrease with k if only h is sufficiently small. The results for options on the maximum and on the minimum are similar. Again, the error within the interior domain is indistinguishable for both choices of boundary functions. As for the error on the whole computational domain, we observe a large maximum error that is independent of the mesh width and that is attained at $\tau = 1$, $\mathbf{S} = (S_{\max}, S_{\max})$ for both boundary functions now. In addition, the pay-off function is not smooth at $S_i = 30$, $S_j \leq 30$ and at $S_1 = S_2 \geq E$ (cf. the plot for $\tau = 0$ in Figure 5.6). This results in the maximum error within $[\frac{1}{40}, 40]^2 \times (0 : 0.1 : 1)$ occurring at the grid point nearest $(40, 40)$ in most cases; in Figure 5.6, the numerical solution appears to be "pulled up" at this point. In Figure 5.7, we show a plot of the corresponding point-wise error. Again, we see that the largest error by far is attained at the boundary. At the locations where the initial condition is non-smooth, we observe initial disturbances which become smoother with time thanks to the diffusive property of the problem. The disturbance along the $S_i = S_j$ line is the most significant and causes the maximum error in the domain of interest. The fact that the choice $R = 2\ln 30$ now yields better results than $R = 2\ln 40$, in particular on the coarser meshes, indicates that it is advantageous to place grid points at this location in this case. Using $R = 4\ln 30$ does not yield better results than $R = 2\ln 30$, however (results not shown here).

When comparing the performance of the Crank-Nicolson and the implicit Euler methods in Figure 5.8, we first note that the maximum norm error curves of the Crank-Nicolson method are decreasing and levelling off, but never increasing now. This indicates that the Crank-Nicolson method is not as restrictive with respect to the refinement path as in the one-dimensional test case. The convergence order with respect to h is similar for both methods. Interestingly, the error curves are concave rather than linear, reflecting an accelerating convergence order that is lower than $O(h)$ in the beginning but approaches the optimal order of $O(h^2)$ for small h. The error curves of the Crank-Nicolson method level off earlier than the ones of the implicit Euler method, i.e. the final error obtained with the implicit Euler method is smaller. With respect to the time step k, the convergence order of the Crank-Nicolson method is higher again. On the finest mesh,

the slope of the Crank-Nicolson curve is higher than $O(k)$, whereas the implicit Euler curve is less steep and corresponds to an order between $O(k^{1/2})$ and $O(k)$. However, more data would be necessary for a reliable judgement. In particular, the observed slopes are becoming steeper when k is reduced, i.e. the true convergence order with respect to k is probably not reached yet. However, the computational complexity prevents us from further reducing k.

Table 5.10 shows that the error in the multigrid method is negligible for fine meshes again; in addition, we compare the observed computation times for the generation of the solution at all spatial and time grid points in Figure 5.9. For a fixed number K of time steps, we expect our solver to be of linear complexity with respect to the number $N := (2M-1)^2$ of unknowns in the linear system, and the direct solution of the K linear systems to be of order $O(N^3)$. In the plot, we observe that our solver is not quite of order $O(N)$ but rather of $O(N \log N)$. A closer examination reveals that this is due to the `sparse` function assembling sparse matrices; apparently, this involves an ordering algorithm of order $O(N \log N)$. Nevertheless, our solver is of considerably lower computational effort than the direct solver even if this is of order $O(N^2)$ here rather than $O(N^3)$; apparently, MATLAB can make use of the structure of the system matrix to apply a more efficient direct solution method than usual Gaussian elimination with pivoting strategy (for full matrices). According to [MATLAB 7.7.0 Help, 2008], banded solvers are used for banded matrices. Direct solvers for banded matrices with upper and lower band width p can be constructed with complexity $O(Np^2)$. The system matrix of our two-dimensional model problem has a bandwidth of $p = 2M + 2 = O(N^{1/2})$ (cf. (4.22)), hence $Np^2 = O(N^2)$.

5.5 Numerical results for options on three underlying assets

5.5.1 Options on the maximum and the minimum of three assets

As mentioned in Section 5.2.1, we use the same problem parameters as in [Boyle and Tse, 1990, Table 5] and [Engelmann and Schwendner, 1998, Tables 1 and 2] and will compare the results below. We use the implicit Euler method on a computational domain of size $R = 2 \ln 40$; this has the disadvantage that there are no grid points at the locations where the pay-off function is non-smooth, but as the results in literature are evaluated at $\mathbf{S} = (40, 40, 40)$, we choose R such that $(40, 40, 40)$ is a grid point. Furthermore, we use the boundary function $\tilde{g}_2$. As explained in Section 5.4.2, the computation of the exact solution according to Formulae (5.9) and (5.15) is expensive, and we only generate the exact solution at the time steps $\tau = 0, 0.1, \ldots, 1$ again.

The obtained maximum norm errors are shown in Table 5.11. Their quality is similar to the one observed for options on the maximum and the minimum of two assets: the boundary conditions entail a considerable error at $(S_{\max}, S_{\max}, S_{\max})$, but the maximum error within the domain of interest decays monotonically with h and k until levelling off. The maximum error within $[\frac{1}{40}, 40]^3 \times (0 : 0.1 : 1)$ is attained at $(40, 40, 40)$ at an early time step in most cases, i.e. along the line $S_1 = S_2 = S_3 \geq E$ where the pay-off function is not smooth and during the initial phase before the disturbance is smoothed out. The convergence order observed for calls on the maximum (plot not shown) is comparable to the implicit Euler method in two dimensions in Figure 5.8, i.e. between $O(h)$ and $O(h^2)$ with respect to the spatial mesh width and less than $O(k)$ with respect to time; however, we have not enough data available for a reliable evaluation of the convergence behaviour.

Apart from the maximum norm errors within $\overline{\Omega_R}$ and $[\frac{1}{40}, 40]^3$, Table 5.11 shows the obtained option value at the grid point $\mathbf{S} = (40, 40, 40)$, $\tau = 1$, and the point-wise error there. This error is

considerably smaller than the maximum error within $[\frac{1}{40}, 40]^3$, however, even if the former decreases with k, the point-wise error at $\mathbf{S} = (40, 40, 40)$, $\tau = 1$ increases for calls on the maximum. For calls on the minimum, in contrast, we achieve a good accuracy.

In the table below, we compare the results where we achieved the best overall accuracy, i.e. the results for $M = 64$, $k = 0.01$, to those by [Boyle and Tse, 1990] and [Engelmann and Schwendner, 1998]. Both papers present algorithms to numerically compute the value of multi-asset options at *one* specific point $(\mathbf{S}, \tau)$. [Boyle and Tse, 1990] approximate the price of options on the maximum or the minimum of n assets in terms of order statistics of n jointly normal random variables. [Engelmann and Schwendner, 1998] devise an efficient Fourier grid method which can be used for any type of European multi-asset option.

evaluation at $\mathbf{S} = (40, 40, 40)$, $\tau = 1$		[Boyle and Tse, 1990, Table 5]		[Engelmann and Schwendner, 1998, Tables 1 and 2]		our solver with $M = 64$, $k = 0.01$	
	exact value	value	relative error	value	relative error	value	relative error
call on the maximum	20.153	20.185	+0.159%	20.150	−0.015%	20.232	+0.392%
call on the minimum	7.172	7.172	0.000%	7.175	+0.042%	7.169	−0.042%

We observe that the performance of our PDE-based solver is comparable to the method by [Engelmann and Schwendner, 1998] for the call on the minimum. For the call on the maximum, however, the error of our solver is higher than for the other methods. We achieved better point-wise results on coarser meshes, yet the objective of our solver is not to deliver optimal point-wise results but a high accuracy within the whole domain of interest.

Note in addition that we only take into account our results on the grids that we considered in Table 5.11. The limiting factor for the mesh widths there was the computation time of generating the exact solution, not the numerical Black-Scholes solver. Our solver can be used on even finer meshes at acceptable computational effort, hence we might obtain better results than those listed in the table above. We do not demonstrate this for options on the maximum and minimum of three assets, but for basket options in the following section.

5.5.2 Basket options

We will now repeat experiments by [Boyle et al., 1989, Table 2] for calls on the arithmetic and the geometric average of three assets. We do not have any analytical solution available for these cases, hence we cannot judge the maximum error on the grid, but we will merely consider the obtained option value at $\mathbf{S} = (100, 100, 100)$, $\tau = 1$ as [Boyle et al., 1989]. We use the implicit Euler method on a computational domain of size $R = 2 \ln 100$ and with the boundary function $\tilde{g}_1$. Results for a range of grid parameters entailing an acceptable computation time are shown in Table 5.12. The memory capacity of the available computers allows for a spatial grid parameter of $M = 128$ at most. The time step $k = 0.0001$ can be used on grids with up to $M = 32$ (computation time 15 hours).

For both option types, the obtained values at $\mathbf{S} = (100, 100, 100)$, $\tau = 1$ increase monotonically when h or k is reduced; the spatial mesh width h has a more noticeable influence than the time step k. For the option on the geometric average, [Boyle et al., 1989] provide the accurate value

of 11.581 obtained by some not further specified analytic formula. We thus see that the values obtained by our solver are always too low but rise against the exact solution as h and k are reduced.

In the table below, we compare the point-wise results on the finest mesh we used to those by [Boyle et al., 1989]. The results there are obtained using an n-dimensional extension of the lattice binomial method developed by [Cox et al., 1979] for single-asset options.

evaluation at $\mathbf{S} = (100, 100, 100)$, $\tau = 1$	[Boyle et al., 1989, Table 2]			our solver with $M = 128$, $k = 0.01$	
	exact value	value	relative error	value	relative error
call on the arithmetic average	–	12.078	–	12.072	–
call on the geometric average	11.581	11.573	−0.069%	11.551	−0.259%

We note that the numerical results for a call on the arithmetic average obtained by [Boyle et al., 1989] and by our solver are in very good accordance. Our result for the call on the geometric average is worse, yet the difference to the accurate result is only 3 cents.

5.6 Summary of numerical results

We have presented results of our numerical solver applied to realistic test cases in one, two, and three dimensions. Although the convergence of the θ-method could not theoretically be established in Section 2.3.1, we experimentally observed satisfactory convergence of the implicit Euler method with respect to the maximum norm (and the discrete ℓ^2-norm in a one-dimensional experiment), even if the theoretically optimal convergence order of $O(h^2) + O(k)$ was not attained. The Crank-Nicolson method, in contrast, did not perform well in the one-dimensional test case; we conjectured a convergence condition of the form $\frac{k}{h^2} \leq C$ (for both choices of norms), which is undesirable in practice. This problem did not occur in the two- and three-dimensional test cases. The higher consistency order of the Crank-Nicolson method with respect to time was apparent in all our experiments. Nevertheless, we consider the implicit Euler method preferable for its reliability.

A comparison showed that the full multigrid V-cycle method (with the weighted Jacobi method with $\omega = \frac{2n}{2n+1}$ as smoothing operation, n-linear interpolation and full weighting restriction) with appropriately chosen smoothing and cycling parameters converges and achieves an accuracy comparable to a direct solution method, at an execution time that is essentially linear with respect to the number of unknowns in the linear system and thus almost optimal.

We experimentally examined the quality of the choices (5.22) of artificial boundary conditions and found that the boundary function had no perceptible effect on the solution in the interior of the computational domain. Thus, a simple choice (e.g., the time-independent function $\tilde{g}_1$) can be used in practice. The relative insignificance of the boundary conditions can be explained by the dominant diffusion part in the transformed Black-Scholes PDE (cf. (1.60) and [Persson and von Sydow, 2007]). This also accounts for the smoothing of initial disturbances in the numerical solution that we observed at non-smooth points of the initial function. For the choice of parameters used here, the convection part of the PDE did not cause problems in the numerical solution.

The estimate of the localisation error that we derived in Section 1.4.3 yields a convergence rate of $O(\frac{1}{R})$ at best. If this estimate was sharp, the size of the computational domain would have to be chosen much larger than the domain of interest in order to keep the localisation error small. We experimentally analysed the size of the localisation error measured in the maximum norm in one and two dimensions. There, we observed a much faster decay: in fact, the localisation error decays approximately exponentially with respect to the domain size R if $n = 2$, and exponentially with respect to R^2 if $n = 1$. This is even better than the exponential bound we found by a variational approach for the one-dimensional case in Section 1.4.3. To conclude, the experimental results demonstrated that our analytical localisation error estimates are very pessimistic, and that the localisation error is in practice negligible with respect to the discretisation error. In particular, it suffices to choose the computational domain of twice the diameter of the domain of interest (with respect to log-price variables). For options on the maximum and the minimum of several assets, it turned out advantageous to choose the computational domain such that there are grid points at the locations where the pay-off function is not smooth.

In all, the performance of our solver, particularly the implicit Euler version, was satisfactory in all test cases and achieved a maximum norm error of fractions of one currency unit within the whole domain of interesting asset prices. However, fine grids have to be used to achieve this accuracy, and the associated computational effort is already very high for three-dimensional problems. The number of spatial grid points and hence the number of unknowns in the linear system which controls the computation time is exponential in the space dimension n. Hence, our solver is in practice only applicable to multi-asset options with few underlying assets, depending on the computational capacity available.

Crank-Nicolson method $R = 2\ln 40$			**call option** error in $\overline{\Omega_R} \times [0,T]$		error in $[\frac{1}{40}, 40] \times [0,T]$		**put option** error in $\overline{\Omega_R} \times [0,T]$		error in $[\frac{1}{40}, 40] \times [0,T]$	
M	h	k	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$
32	0.23	0.1	2.855	0.615	0.161	0.161	2.855	0.158	0.158	0.158
64	0.12	0.1	2.855	0.158	0.047	0.047	2.855	0.047	0.047	0.047
128	0.058	0.1	2.855	0.094	0.094	0.094	2.855	0.094	0.094	0.094
256	0.029	0.1	2.855	0.223	0.223	0.223	2.855	0.223	0.223	0.223
512	0.014	0.1	2.855	0.266	0.266	0.266	2.855	0.266	0.266	0.266
1024	0.0072	0.1	2.855	0.277	0.277	0.277	2.855	0.277	0.277	0.277
2048	0.0036	0.1	2.855	0.279	0.279	0.279	2.855	0.279	0.279	0.279
4096	0.0018	0.1	2.855	0.280	0.280	0.280	2.855	0.280	0.280	0.280
32	0.23	0.01	2.855	0.615	0.163	0.163	2.855	0.160	0.160	0.160
64	0.12	0.01	2.855	0.158	0.074	0.074	2.855	0.074	0.074	0.074
128	0.058	0.01	2.855	0.145	0.145	0.145	2.855	0.145	0.145	0.145
256	0.029	0.01	2.855	0.032	0.032	0.032	2.855	0.032	0.032	0.032
512	0.014	0.01	2.855	0.046	0.046	0.046	2.855	0.046	0.046	0.046
1024	0.0072	0.01	2.855	0.074	0.074	0.074	2.855	0.074	0.074	0.074
2048	0.0036	0.01	2.855	0.082	0.082	0.082	2.855	0.082	0.082	0.082
4096	0.0018	0.01	2.855	0.083	0.083	0.083	2.855	0.083	0.083	0.083
32	0.23	0.001	2.855	0.615	0.163	0.163	2.855	0.160	0.160	0.160
64	0.12	0.001	2.855	0.158	0.074	0.074	2.855	0.074	0.074	0.074
128	0.058	0.001	2.855	0.146	0.146	0.146	2.855	0.146	0.146	0.146
256	0.029	0.001	2.855	0.070	0.070	0.070	2.855	0.070	0.070	0.070
512	0.014	0.001	2.855	0.030	0.030	0.030	2.855	0.030	0.030	0.030
1024	0.0072	0.001	2.855	0.007	0.007	0.007	2.855	0.007	0.007	0.007
2048	0.0036	0.001	2.855	0.017	0.017	0.017	2.855	0.017	0.017	0.017
4096	0.0018	0.001	2.855	0.021	0.021	0.021	2.855	0.021	0.021	0.021
32	0.23	0.0001	2.855	0.615	0.163	0.163	2.855	0.160	0.160	0.160
64	0.12	0.0001	2.855	0.160	0.074	0.074	2.855	0.074	0.074	0.074
128	0.058	0.0001	2.855	0.146	0.146	0.146	2.855	0.146	0.146	0.146
256	0.029	0.0001	2.855	0.070	0.070	0.070	2.855	0.070	0.070	0.070
512	0.014	0.0001	2.855	0.032	0.032	0.032	2.855	0.032	0.032	0.032
1024	0.0072	0.0001	2.855	0.014	0.014	0.014	2.855	0.014	0.014	0.014
2048	0.0036	0.0001	2.855	0.004	0.004	0.004	2.855	0.004	0.004	0.004
4096	0.0018	0.0001	2.855	0.002	0.002	0.002	2.855	0.002	0.002	0.002

Table 5.3: Results obtained with the Crank-Nicolson method for single-asset options with exercise price $E = 30$, $T = 1$, $\sigma = 0.3$, $r = 0.1$, $R = 2\ln 40$, i.e. $S_{\max} = \$1600$, $\nu_1 = \nu_2 = 7$, $\nu = 1$, and $\omega = \frac{2}{3}$. The error is measured in the maximum norm and with respect to the exact solution obtained by `blsprice`.

Crank-Nicolson method $R = 2\ln 30$			**call option** error in $\overline{\Omega_R} \times [0,T]$		**call option** error in $[\frac{1}{40}, 40] \times [0,T]$		**put option** error in $\overline{\Omega_R} \times [0,T]$		**put option** error in $[\frac{1}{40}, 40] \times [0,T]$	
M	h	k	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$
32	0.21	0.1	2.855	0.543	0.543	0.543	2.855	0.545	0.545	0.545
64	0.11	0.1	2.855	0.184	0.184	0.184	2.855	0.184	0.184	0.184
128	0.053	0.1	2.855	0.107	0.107	0.107	2.855	0.107	0.107	0.107
256	0.027	0.1	2.855	0.235	0.235	0.235	2.855	0.235	0.235	0.235
512	0.013	0.1	2.855	0.274	0.274	0.274	2.855	0.274	0.274	0.274
1024	0.0066	0.1	2.855	0.284	0.284	0.284	2.855	0.284	0.284	0.284
2048	0.0033	0.1	2.855	0.287	0.287	0.287	2.855	0.287	0.287	0.287
4096	0.0017	0.1	2.855	0.287	0.287	0.287	2.855	0.287	0.287	0.287
32	0.21	0.01	2.855	0.549	0.549	0.549	2.855	0.551	0.551	0.551
64	0.11	0.01	2.855	0.280	0.280	0.280	2.855	0.280	0.280	0.280
128	0.053	0.01	2.855	0.138	0.138	0.138	2.855	0.138	0.138	0.138
256	0.027	0.01	2.855	0.025	0.025	0.025	2.855	0.025	0.025	0.025
512	0.013	0.01	2.855	0.052	0.052	0.052	2.855	0.052	0.052	0.052
1024	0.0066	0.01	2.855	0.080	0.080	0.080	2.855	0.080	0.080	0.080
2048	0.0033	0.01	2.855	0.088	0.088	0.088	2.855	0.088	0.088	0.088
4096	0.0017	0.01	2.855	0.090	0.090	0.090	2.855	0.090	0.090	0.090
32	0.21	0.001	2.855	0.549	0.549	0.549	2.855	0.551	0.551	0.551
64	0.11	0.001	2.855	0.280	0.280	0.280	2.855	0.280	0.280	0.280
128	0.053	0.001	2.855	0.141	0.141	0.141	2.855	0.141	0.141	0.141
256	0.027	0.001	2.855	0.070	0.070	0.070	2.855	0.070	0.070	0.070
512	0.013	0.001	2.855	0.031	0.031	0.031	2.855	0.031	0.031	0.031
1024	0.0066	0.001	2.855	0.009	0.009	0.009	2.855	0.009	0.009	0.009
2048	0.0033	0.001	2.855	0.021	0.021	0.021	2.855	0.021	0.021	0.021
4096	0.0017	0.001	2.855	0.027	0.027	0.027	2.855	0.027	0.027	0.027
32	0.213	0.0001	2.855	0.549	0.549	0.549	2.855	0.551	0.551	0.551
64	0.11	0.0001	2.855	0.280	0.280	0.280	2.855	0.280	0.280	0.280
128	0.053	0.0001	2.855	0.141	0.141	0.141	2.855	0.141	0.141	0.141
256	0.027	0.0001	2.855	0.070	0.070	0.070	2.855	0.070	0.070	0.070
512	0.013	0.0001	2.855	0.035	0.035	0.035	2.855	0.035	0.035	0.035
1024	0.0066	0.0001	2.855	0.017	0.017	0.017	2.855	0.017	0.017	0.017
2048	0.0033	0.0001	2.855	0.006	0.006	0.006	2.855	0.006	0.006	0.006
4096	0.0017	0.0001	2.855	0.003	0.003	0.003	2.855	0.003	0.003	0.003

Table 5.4: Results obtained with the Crank-Nicolson method for single-asset options with exercise price $E = 30$, $T = 1$, $\sigma = 0.3$, $r = 0.1$, $R = 2\ln 30$, i.e. $S_{\max} = \$900$, $\nu_1 = \nu_2 = 7$, $\nu = 1$, and $\omega = \frac{2}{3}$. The error is measured in the maximum norm and with respect to the exact solution obtained by `blsprice`.

Implicit Euler method $R = 2\ln 40$			**call option** error in $\overline{\Omega_R} \times [0,T]$		error in $[\frac{1}{40}, 40] \times [0,T]$		**put option** error in $\overline{\Omega_R} \times [0,T]$		error in $[\frac{1}{40}, 40] \times [0,T]$	
M	h	k	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$
32	0.23	0.1	2.855	0.592	0.202	0.202	2.855	0.204	0.204	0.204
64	0.12	0.1	2.855	0.144	0.081	0.081	2.855	0.083	0.083	0.083
128	0.058	0.1	2.855	0.207	0.207	0.207	2.855	0.206	0.206	0.206
256	0.029	0.1	2.855	0.206	0.150	0.150	2.855	0.149	0.149	0.149
512	0.014	0.1	2.855	0.225	0.134	0.134	2.855	0.133	0.133	0.133
1024	0.0072	0.1	2.855	0.156	0.131	0.131	2.855	0.129	0.129	0.129
2048	0.0036	0.1	2.855	0.130	0.130	0.130	2.855	0.129	0.129	0.129
4096	0.0018	0.1	2.855	0.130	0.130	0.130	2.855	0.129	0.129	0.129
32	0.23	0.01	2.855	0.613	0.164	0.164	2.855	0.161	0.161	0.161
64	0.12	0.01	2.855	0.157	0.072	0.072	2.855	0.072	0.072	0.072
128	0.058	0.01	2.855	0.164	0.164	0.164	2.855	0.164	0.164	0.164
256	0.029	0.01	2.855	0.091	0.091	0.091	2.855	0.091	0.091	0.091
512	0.014	0.01	2.855	0.055	0.055	0.055	2.855	0.055	0.055	0.055
1024	0.0072	0.01	2.855	0.077	0.044	0.044	2.855	0.044	0.044	0.044
2048	0.0036	0.01	2.855	0.064	0.041	0.041	2.855	0.041	0.041	0.041
4096	0.0018	0.01	2.855	0.041	0.041	0.041	2.855	0.041	0.041	0.041
32	0.23	0.001	2.855	0.615	0.163	0.163	2.855	0.160	0.160	0.160
64	0.12	0.001	2.855	0.158	0.074	0.074	2.855	0.074	0.074	0.074
128	0.058	0.001	2.855	0.148	0.148	0.148	2.855	0.148	0.148	0.148
256	0.029	0.001	2.855	0.074	0.074	0.074	2.855	0.074	0.074	0.074
512	0.014	0.001	2.855	0.039	0.039	0.039	2.855	0.039	0.039	0.039
1024	0.0072	0.001	2.855	0.021	0.021	0.021	2.855	0.021	0.021	0.021
2048	0.0036	0.001	2.855	0.019	0.014	0.014	2.855	0.014	0.014	0.014
4096	0.0018	0.001	2.855	0.024	0.013	0.013	2.855	0.013	0.013	0.013
32	0.23	0.0001	2.855	0.615	0.163	0.163	2.855	0.160	0.160	0.160
64	0.12	0.0001	2.855	0.160	0.074	0.074	2.855	0.074	0.074	0.074
128	0.058	0.0001	2.855	0.147	0.147	0.147	2.855	0.147	0.147	0.147
256	0.029	0.0001	2.855	0.071	0.071	0.071	2.855	0.071	0.071	0.071
512	0.014	0.0001	2.855	0.033	0.033	0.033	2.855	0.033	0.033	0.033
1024	0.0072	0.0001	2.855	0.015	0.015	0.015	2.855	0.015	0.015	0.015
2048	0.0036	0.0001	2.855	0.007	0.007	0.007	2.855	0.007	0.007	0.007
4096	0.0018	0.0001	2.855	0.004	0.004	0.004	2.855	0.004	0.004	0.004

Table 5.5: Results obtained with the implicit Euler method for single-asset options with exercise price $E = 30$, $T = 1$, $\sigma = 0.3$, $r = 0.1$, $R = 2\ln 40$, i.e. $S_{\max} = \$1600$, $\nu_1 = \nu_2 = 7$, $\nu = 1$, and $\omega = \frac{2}{3}$. The error is measured in the maximum norm and with respect to the exact solution obtained by `blsprice`.

Implicit Euler method $R = 2\ln 30$			**call option** error in $\overline{\Omega_R} \times [0,T]$		**call option** error in $[\frac{1}{40}, 40] \times [0,T]$		**put option** error in $\overline{\Omega_R} \times [0,T]$		**put option** error in $[\frac{1}{40}, 40] \times [0,T]$	
M	h	k	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$
32	0.21	0.1	2.855	0.591	0.591	0.591	2.855	0.591	0.591	0.591
64	0.11	0.1	2.855	0.344	0.344	0.344	2.855	0.343	0.343	0.343
128	0.053	0.1	2.855	0.200	0.200	0.200	2.855	0.198	0.198	0.198
256	0.027	0.1	2.855	0.149	0.149	0.149	2.855	0.147	0.147	0.147
512	0.013	0.1	2.855	0.135	0.135	0.135	2.855	0.133	0.133	0.133
1024	0.0066	0.1	2.855	0.131	0.131	0.131	2.855	0.130	0.130	0.130
2048	0.0033	0.1	2.855	0.130	0.130	0.130	2.855	0.129	0.129	0.129
4096	0.0017	0.1	2.855	0.130	0.130	0.130	2.855	0.129	0.129	0.129
32	0.21	0.01	2.855	0.554	0.554	0.554	2.855	0.556	0.556	0.556
64	0.11	0.01	2.855	0.290	0.290	0.290	2.855	0.290	0.290	0.290
128	0.053	0.01	2.855	0.160	0.160	0.160	2.855	0.160	0.160	0.160
256	0.027	0.01	2.855	0.089	0.089	0.089	2.855	0.089	0.089	0.089
512	0.013	0.01	2.855	0.055	0.055	0.055	2.855	0.055	0.055	0.055
1024	0.0066	0.01	2.855	0.045	0.045	0.045	2.855	0.045	0.045	0.045
2048	0.0033	0.01	2.855	0.042	0.042	0.042	2.855	0.042	0.042	0.042
4096	0.0017	0.01	2.855	0.041	0.041	0.041	2.855	0.041	0.041	0.041
32	0.21	0.001	2.855	0.550	0.550	0.550	2.855	0.552	0.552	0.552
64	0.11	0.001	2.855	0.281	0.281	0.281	2.855	0.281	0.281	0.281
128	0.053	0.001	2.855	0.143	0.143	0.143	2.855	0.143	0.143	0.143
256	0.027	0.001	2.855	0.074	0.074	0.074	2.855	0.074	0.074	0.074
512	0.013	0.001	2.855	0.042	0.042	0.042	2.855	0.042	0.042	0.042
1024	0.0066	0.001	2.855	0.023	0.023	0.023	2.855	0.023	0.023	0.023
2048	0.0033	0.001	2.855	0.016	0.016	0.016	2.855	0.016	0.016	0.016
4096	0.0017	0.001	2.855	0.014	0.014	0.014	2.855	0.014	0.014	0.014
32	0.21	0.0001	2.855	0.549	0.549	0.549	2.855	0.551	0.551	0.551
64	0.11	0.0001	2.855	0.280	0.280	0.280	2.855	0.280	0.280	0.280
128	0.053	0.0001	2.855	0.141	0.141	0.141	2.855	0.141	0.141	0.141
256	0.027	0.0001	2.855	0.071	0.071	0.071	2.855	0.071	0.071	0.071
512	0.013	0.0001	2.855	0.036	0.036	0.036	2.855	0.036	0.036	0.036
1024	0.0066	0.0001	2.855	0.019	0.019	0.019	2.855	0.019	0.019	0.019
2048	0.0033	0.0001	2.855	0.011	0.011	0.011	2.855	0.011	0.011	0.011
4096	0.0017	0.0001	2.855	0.006	0.006	0.006	2.855	0.006	0.006	0.006

Table 5.6: Results obtained with the implicit Euler method for single-asset options with exercise price $E = 30$, $T = 1$, $\sigma = 0.3$, $r = 0.1$, $R = 2\ln 30$, i.e. $S_{\max} = \$900$, $\nu_1 = \nu_2 = 7$, $\nu = 1$, and $\omega = \frac{2}{3}$. The error is measured in the maximum norm and with respect to the exact solution obtained by `blsprice`.

$\nu = 1$, $\nu_1 = \nu_2 = 7$

M	$k = 0.1$ multigrid error in $\overline{\Omega_R} \times [0,T]$	$k = 0.1$ multigrid error in $[\frac{1}{40}, 40] \times [0,T]$	$k = 0.01$ multigrid error in $\overline{\Omega_R} \times [0,T]$	$k = 0.01$ multigrid error in $[\frac{1}{40}, 40] \times [0,T]$	$k = 0.001$ multigrid error in $\overline{\Omega_R} \times [0,T]$	$k = 0.001$ multigrid error in $[\frac{1}{40}, 40] \times [0,T]$	$k = 0.0001$ multigrid error in $\overline{\Omega_R} \times [0,T]$	$k = 0.0001$ multigrid error in $[\frac{1}{40}, 40] \times [0,T]$
32	3.8e − 05	9.0e − 07	2.7e − 04	9.4e − 06	2.7e − 03	1.0e − 04	2.7e − 02	1.0e − 03
64	1.0e − 04	3.4e − 06	3.2e − 05	1.0e − 06	2.7e − 04	9.7e − 06	2.7e − 03	9.8e − 05
128	2.5e − 03	6.2e − 05	1.4e − 05	5.7e − 07	2.3e − 05	1.8e − 06	2.1e − 04	1.8e − 05
256	5.1e − 02	2.1e − 04	8.3e − 05	4.1e − 06	3.3e − 06	1.8e − 07	1.6e − 05	1.1e − 06
512	8.5e − 02	2.7e − 04	4.9e − 03	4.3e − 05	3.7e − 06	2.8e − 07	1.4e − 06	1.4e − 07
1024	9.2e − 02	2.9e − 04	2.3e − 02	7.1e − 05	1.6e − 04	3.5e − 06	3.4e − 07	5.2e − 08
2048	6.7e − 02	1.6e − 04	2.9e − 02	8.7e − 05	3.6e − 03	1.9e − 05	3.4e − 06	1.8e − 07
4096	4.1e − 02	2.2e − 04	2.7e − 02	1.3e − 04	8.3e − 03	1.7e − 05	2.0e − 04	2.5e − 06

$\nu = 3$, $\nu_1 = \nu_2 = 15$

M	$k = 0.1$ multigrid error in $\overline{\Omega_R} \times [0,T]$	$k = 0.1$ multigrid error in $[\frac{1}{40}, 40] \times [0,T]$	$k = 0.01$ multigrid error in $\overline{\Omega_R} \times [0,T]$	$k = 0.01$ multigrid error in $[\frac{1}{40}, 40] \times [0,T]$	$k = 0.001$ multigrid error in $\overline{\Omega_R} \times [0,T]$	$k = 0.001$ multigrid error in $[\frac{1}{40}, 40] \times [0,T]$	$k = 0.0001$ multigrid error in $\overline{\Omega_R} \times [0,T]$	$k = 0.0001$ multigrid error in $[\frac{1}{40}, 40] \times [0,T]$
32	6.8e − 13	7.1e − 15	2.3e − 12	8.9e − 15	1.0e − 11	7.6e − 14	6.0e − 10	9.4e − 12
64	9.1e − 13	8.9e − 15	6.6e − 12	9.2e − 14	2.3e − 11	2.8e − 13	4.3e − 10	6.2e − 12
128	8.0e − 13	3.6e − 15	1.8e − 12	2.5e − 14	7.8e − 11	1.1e − 12	6.6e − 10	9.9e − 12
256	1.4e − 09	4.9e − 12	7.7e − 12	6.0e − 14	4.5e − 11	6.7e − 13	6.9e − 10	1.0e − 11
512	6.1e − 07	7.4e − 10	1.0e − 11	1.7e − 13	8.6e − 11	1.3e − 12	1.2e − 10	1.8e − 12
1024	2.4e − 06	4.3e − 09	9.7e − 09	2.3e − 11	5.0e − 11	7.5e − 13	9.5e − 10	1.4e − 11
2048	3.2e − 06	4.0e − 09	3.4e − 07	3.8e − 10	2.2e − 10	3.3e − 12	5.0e − 10	6.6e − 12
4096	2.5e − 06	3.6e − 09	9.0e − 07	1.1e − 09	1.9e − 08	2.8e − 11	1.1e − 09	1.8e − 11

Table 5.7: The error in the multigrid method, calculated as the maximum norm of the difference between the solution obtained by our Crank-Nicolson solver and the solution obtained by applying the MATLAB direct solver on the linear systems (4.3) resulting from the Crank-Nicolson method. The results refer to a single-asset call option with $E = 30$, $T = 1$, $\sigma = 0.3$, $r = 0.1$, $R = 2 \ln 40$ and boundary function $\tilde{g}_2$. In our solver, one full multigrid V-cycle is performed with $\omega = \frac{2}{3}$ and the cycling parameters specified above each table.

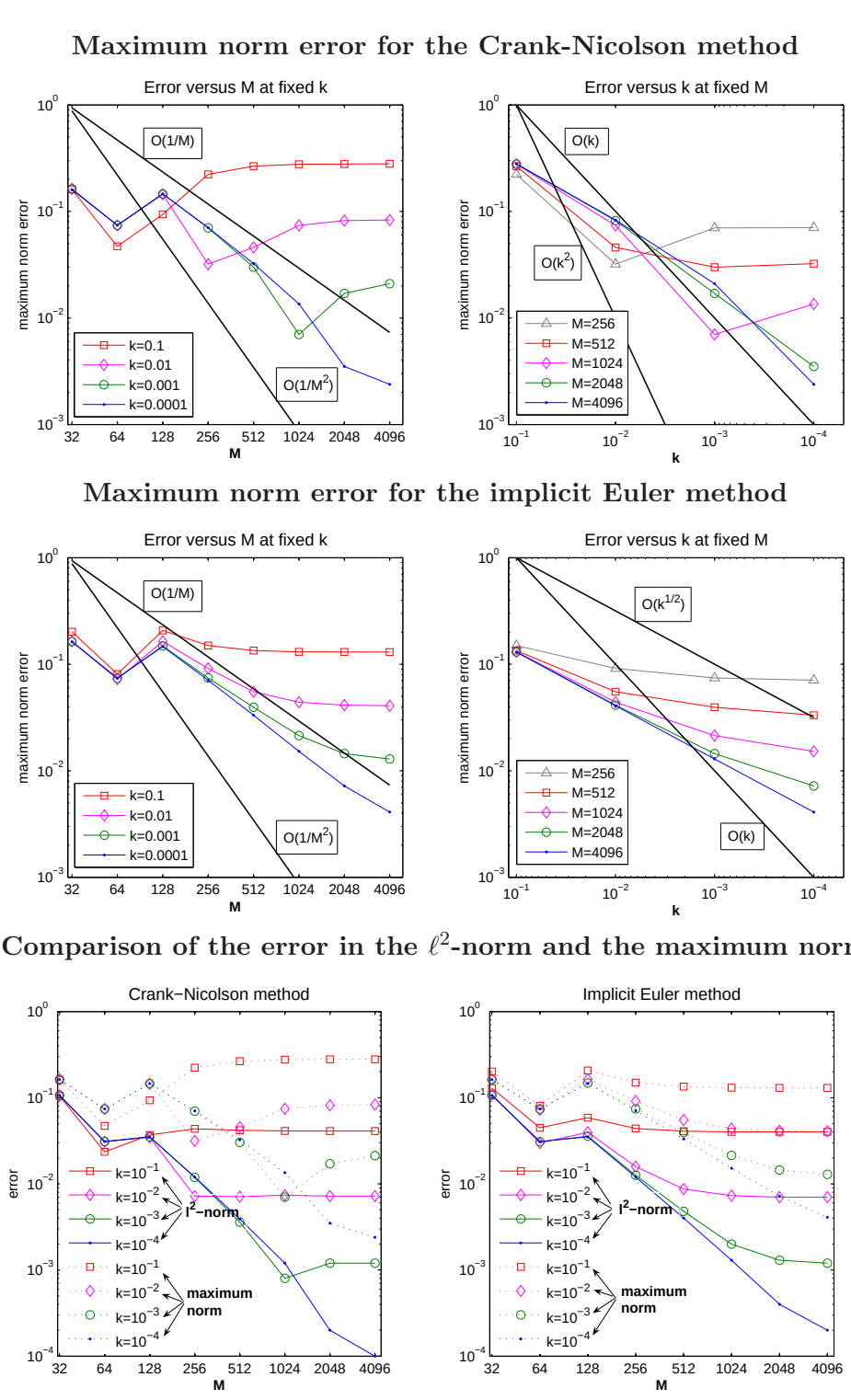

Figure 5.2: Plots of the maximum norm and ℓ^2-norm errors (with respect to the solution of the untruncated Black-Scholes problem) within $[\frac{1}{40}, 40] \times [0, T]$ obtained for a single-asset call option with parameters as in Tables 5.3 and 5.5 against the space or time grid parameter, for the Crank-Nicolson and the implicit Euler methods.

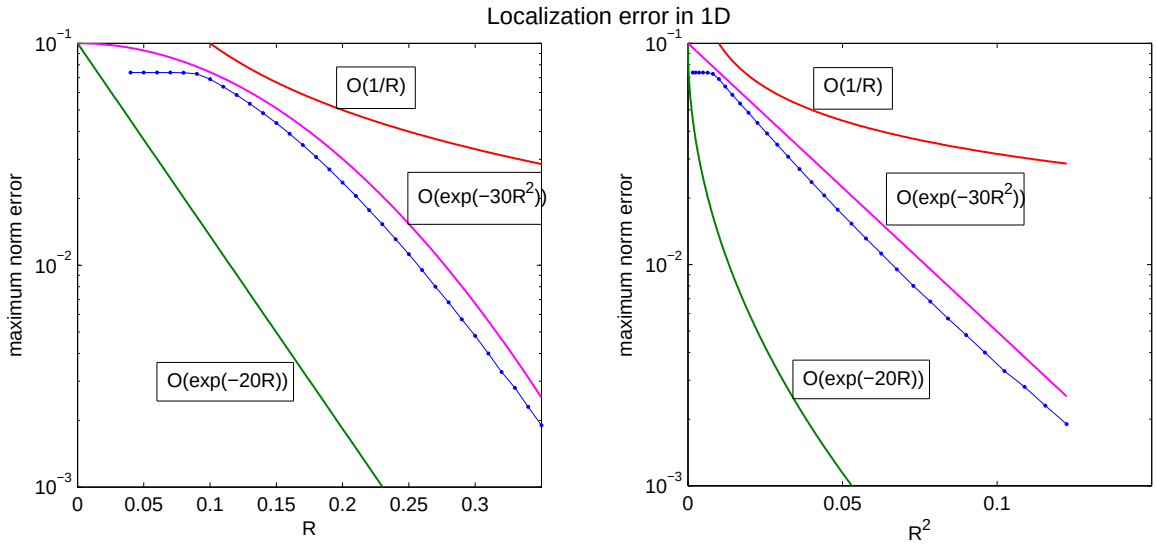

Figure 5.3: Plots of the error within $\overline{\Omega_{0.005}} \times [0,T]$ versus the size R of the computational domain, for single-asset call options with $E = 1$, $T = 1$, $\sigma = 0.3$, $r = 0.1$, and boundary function $\tilde{g}_2$. We use $M = 1024$, $k = 0.001$, where the error empirically remains constant when the grid is further refined, the implicit Euler method, and multigrid parameters $\nu_1 = \nu_2 = 7$, $\nu = 1$, and $\omega = \frac{2}{3}$.

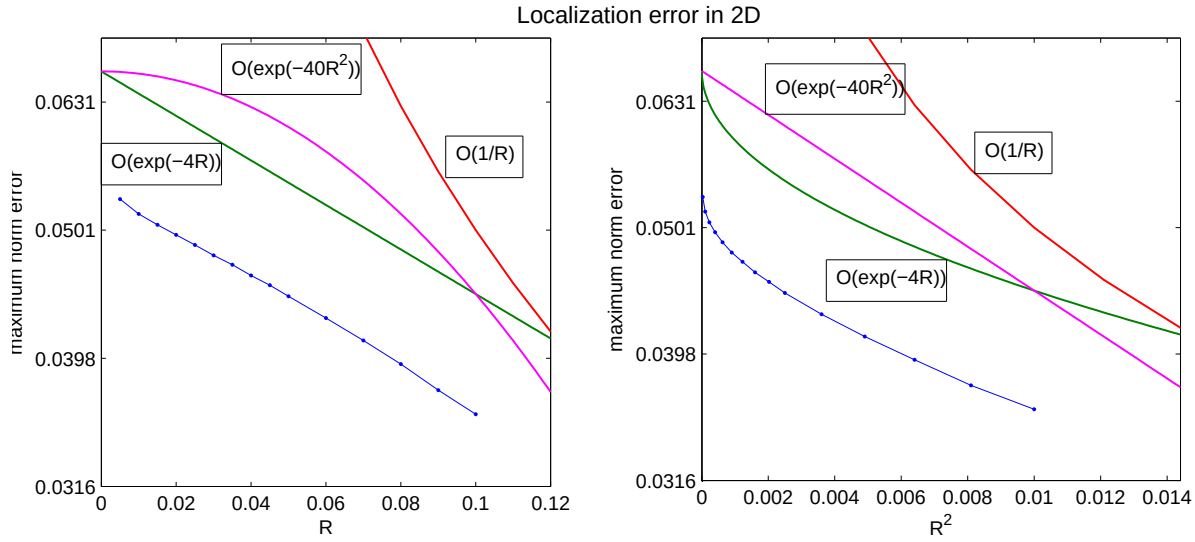

Figure 5.4: Plots of the error within $\overline{\Omega_{0.005}} \times [0,T]$ versus the size R of the computational domain, for exchange options with $q_1 = q_2 = 1$, $T = 1$, $\sigma_1 = 0.25$, $\sigma_2 = 0.3$, $\rho = 0.9$, and $r = 0.1$. We use $M = 128$, $k = 0.01$, where the error empirically remains constant when the grid is further refined, the implicit Euler method, and multigrid parameters $\nu_1 = \nu_2 = 10$, $\nu = 1$, and $\omega = 0.8$.

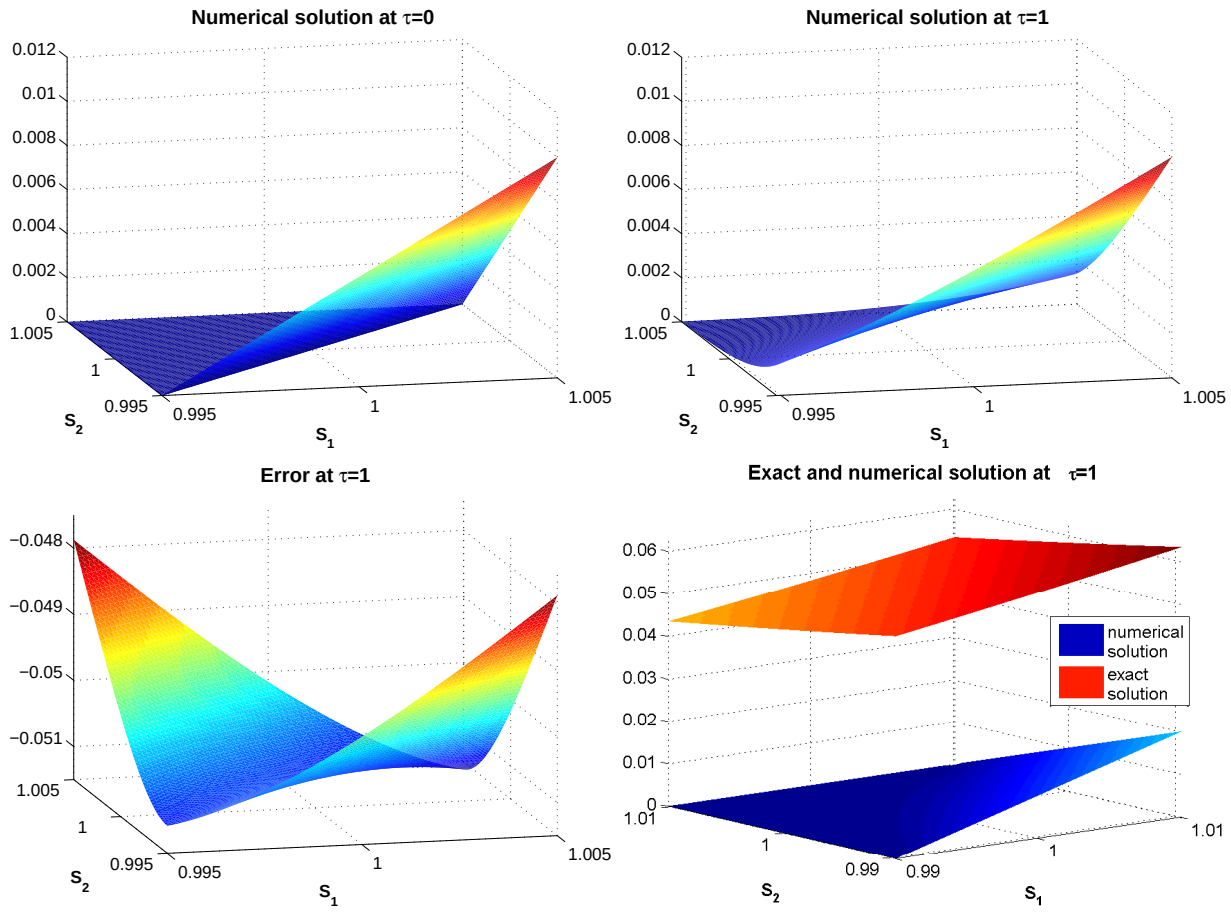

Figure 5.5: Plots of the numerical solution and its point-wise error for an exchange option with $q_1 = q_2 = 1$, $T = 1$, $\sigma_1 = 0.25$, $\sigma_2 = 0.3$, $\rho = 0.9$, and $r = 0.1$. The numerical solution is obtained using the implicit Euler method with $R = 0.01$, i.e. $S_{\max} \approx \$1.01$, $M = 64$, $k = 0.01$, boundary function $\tilde{g}_1 = \tilde{g}_2$, and multigrid parameters $\nu_1 = \nu_2 = 10$, $\nu = 1$, and $\omega = 0.8$.

Implicit Euler $R = 2\ln 40$			**call on the maximum** error in $\overline{\Omega_R}$ $\times (0:0.1:1)$		**call on the maximum** error in $[\frac{1}{40}, 40]^2$ $\times (0:0.1:1)$		**call on the minimum** error in $\overline{\Omega_R}$ $\times (0:0.1:1)$		**call on the minimum** error in $[\frac{1}{40}, 40]^2$ $\times (0:0.1:1)$	
M	h	k	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$
16	0.46	0.1	87.234	84.379	1.039	1.039	81.524	84.379	0.714	0.714
32	0.23	0.1	87.234	84.379	0.751	0.751	81.524	84.379	0.781	0.781
64	0.12	0.1	87.234	84.379	0.407	0.407	81.524	84.379	0.431	0.431
128	0.058	0.1	87.234	84.379	0.220	0.220	81.524	84.379	0.248	0.248
256	0.029	0.1	87.234	84.379	0.159	0.159	81.524	84.379	0.178	0.178
512	0.014	0.1	87.234	84.379	0.139	0.139	81.524	84.379	0.147	0.147
1024	0.0072	0.1	87.234	84.379	0.132	0.132	81.524	84.379	0.141	0.141
16	0.46	0.01	87.234	84.379	1.111	1.111	81.524	84.379	0.745	0.745
32	0.23	0.01	87.234	84.379	0.809	0.809	81.524	84.379	0.803	0.803
64	0.12	0.01	87.234	84.379	0.450	0.450	81.524	84.379	0.432	0.432
128	0.058	0.01	87.234	84.379	0.207	0.207	81.524	84.379	0.207	0.207
256	0.029	0.01	87.234	84.379	0.095	0.095	81.524	84.379	0.094	0.094
512	0.014	0.01	87.234	84.379	0.042	0.042	81.524	84.379	0.041	0.041
1024	0.0072	0.01	87.234	84.379	0.018	0.018	81.524	84.379	0.022	0.022
16	0.46	0.001	87.234	84.379	1.118	1.118	81.524	84.379	0.749	0.749
32	0.23	0.001	87.234	84.379	0.815	0.815	81.524	84.379	0.805	0.805
64	0.12	0.001	87.234	84.379	0.454	0.454	81.524	84.379	0.433	0.433
128	0.058	0.001	87.234	84.379	0.212	0.212	81.524	84.379	0.211	0.211
256	0.029	0.001	87.234	84.379	0.103	0.103	81.524	84.379	0.102	0.102
512	0.014	0.001	87.234	84.379	0.042	0.042	81.524	84.379	0.042	0.042

Table 5.8: Results obtained with the implicit Euler method for options on the maximum and minimum of two assets with exercise price $E = 30$, $T = 1$, $\sigma_1 = 0.25$, $\sigma_2 = 0.3$, $\rho = 0.9$, $r = 0.1$, $R = 2\ln 40$, i.e. $S_{\max} = \$1600$, $\nu_1 = \nu_2 = 10$, $\nu = 1$, and $\omega = 0.8$. The error is with respect to the solution of the untruncated Black-Scholes problem and measured in the maximum norm.

Implicit Euler $R = 2\ln 30$			**call on the maximum** error in $\overline{\Omega_R}$ $\times (0:0.1:1)$		**call on the maximum** error in $[\frac{1}{40}, 40]^2$ $\times (0:0.1:1)$		**call on the minimum** error in $\overline{\Omega_R}$ $\times (0:0.1:1)$		**call on the minimum** error in $[\frac{1}{40}, 40]^2$ $\times (0:0.1:1)$	
M	h	k	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_1$	$\tilde{g}_2$
16	0.43	0.1	50.318	47.463	1.042	1.042	44.608	47.463	1.758	1.758
32	0.21	0.1	50.318	47.463	0.595	0.595	44.608	47.463	0.829	0.829
64	0.11	0.1	50.318	47.463	0.352	0.352	44.608	47.463	0.418	0.418
128	0.053	0.1	50.318	47.463	0.212	0.212	44.608	47.463	0.237	0.237
256	0.027	0.1	50.318	47.463	0.157	0.157	44.608	47.463	0.173	0.173
512	0.013	0.1	50.318	47.463	0.138	0.138	44.608	47.463	0.146	0.146
1024	0.0066	0.1	50.318	47.463	0.134	0.134	44.608	47.463	0.141	0.141
16	0.43	0.01	50.318	47.463	1.016	1.016	44.608	47.463	1.756	1.756
32	0.21	0.01	50.318	47.463	0.629	0.629	44.608	47.463	0.822	0.822
64	0.11	0.01	50.318	47.463	0.346	0.346	44.608	47.463	0.405	0.405
128	0.053	0.01	50.318	47.463	0.185	0.185	44.608	47.463	0.185	0.185
256	0.027	0.01	50.318	47.463	0.085	0.085	44.608	47.463	0.083	0.083
512	0.013	0.01	50.318	47.463	0.038	0.038	44.608	47.463	0.039	0.039
1024	0.0066	0.01	50.318	47.463	0.018	0.018	44.608	47.463	0.021	0.021
16	0.43	0.001	50.318	47.463	1.013	1.013	44.608	47.463	1.756	1.756
32	0.21	0.001	50.318	47.463	0.634	0.634	44.608	47.463	0.822	0.822
64	0.11	0.001	50.318	47.463	0.351	0.351	44.608	47.463	0.404	0.404
128	0.053	0.001	50.318	47.463	0.190	0.190	44.608	47.463	0.189	0.189
256	0.027	0.001	50.318	47.463	0.092	0.092	44.608	47.463	0.091	0.091
512	0.013	0.001	50.318	47.463	0.037	0.037	44.608	47.463	0.037	0.037

Table 5.9: Results obtained with the implicit Euler method for options on the maximum and minimum of two assets with exercise price $E = 30$, $T = 1$, $\sigma_1 = 0.25$, $\sigma_2 = 0.3$, $\rho = 0.9$, $r = 0.1$, $R = 2\ln 30$, i.e. $S_{\max} = \$900$, $\nu_1 = \nu_2 = 10$, $\nu = 1$, and $\omega = 0.8$. The error is with respect to the solution of the untruncated Black-Scholes problem and measured in the maximum norm.

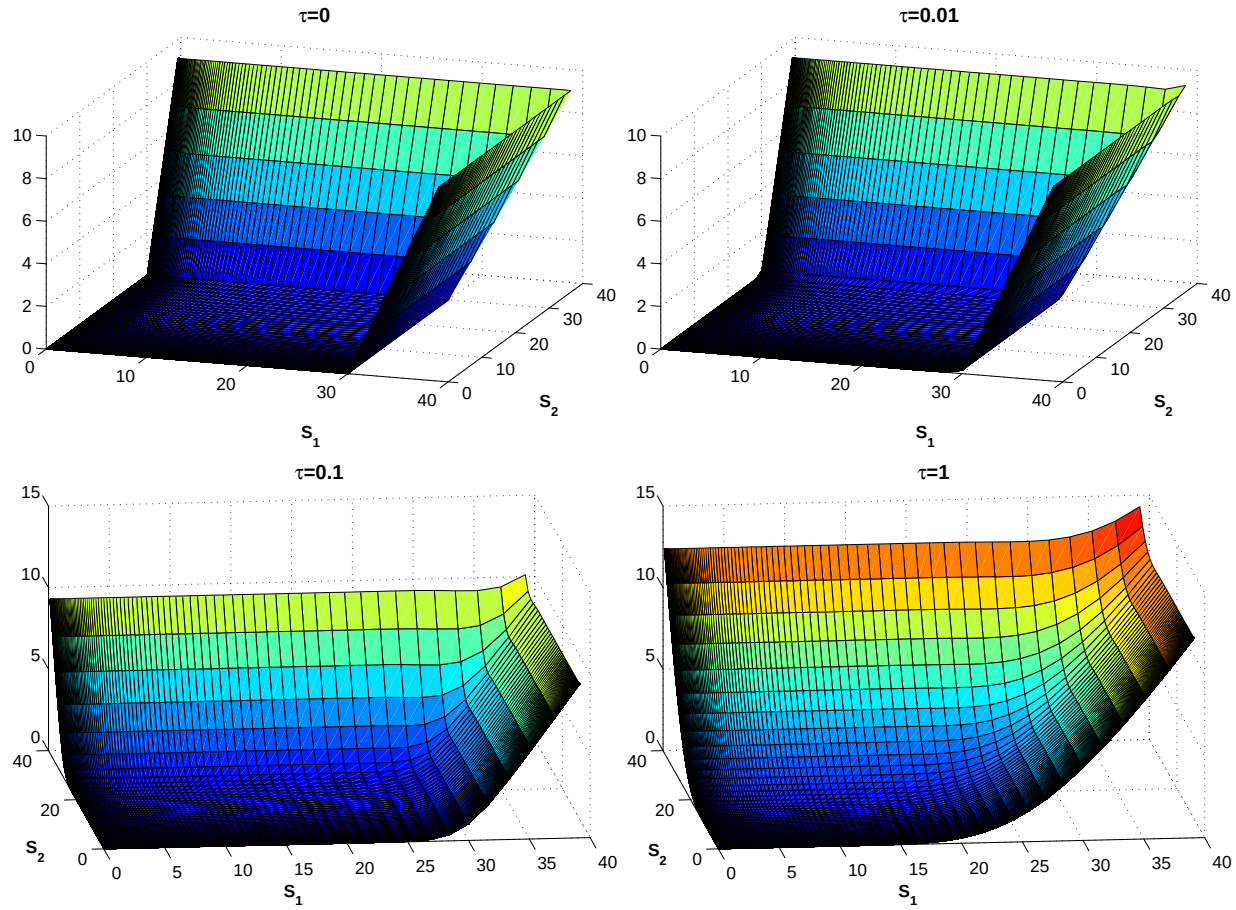

Figure 5.6: Plot of the numerical solution for a call on the maximum of two assets with $E = 30$, $T = 1$, $\sigma_1 = 0.25$, $\sigma_2 = 0.3$, $\rho = 0.9$, $r = 0.1$, obtained by the implicit Euler method with computational parameters $M = 128$, $R = 2\ln 30$, $k = 0.01$, boundary function $\tilde{g}_2$, $\nu_1 = \nu_2 = 10$, $\nu = 1$, and $\omega = 0.8$. The solution in the domain $[\frac{1}{40}, 40]^2$ is shown at different time steps.

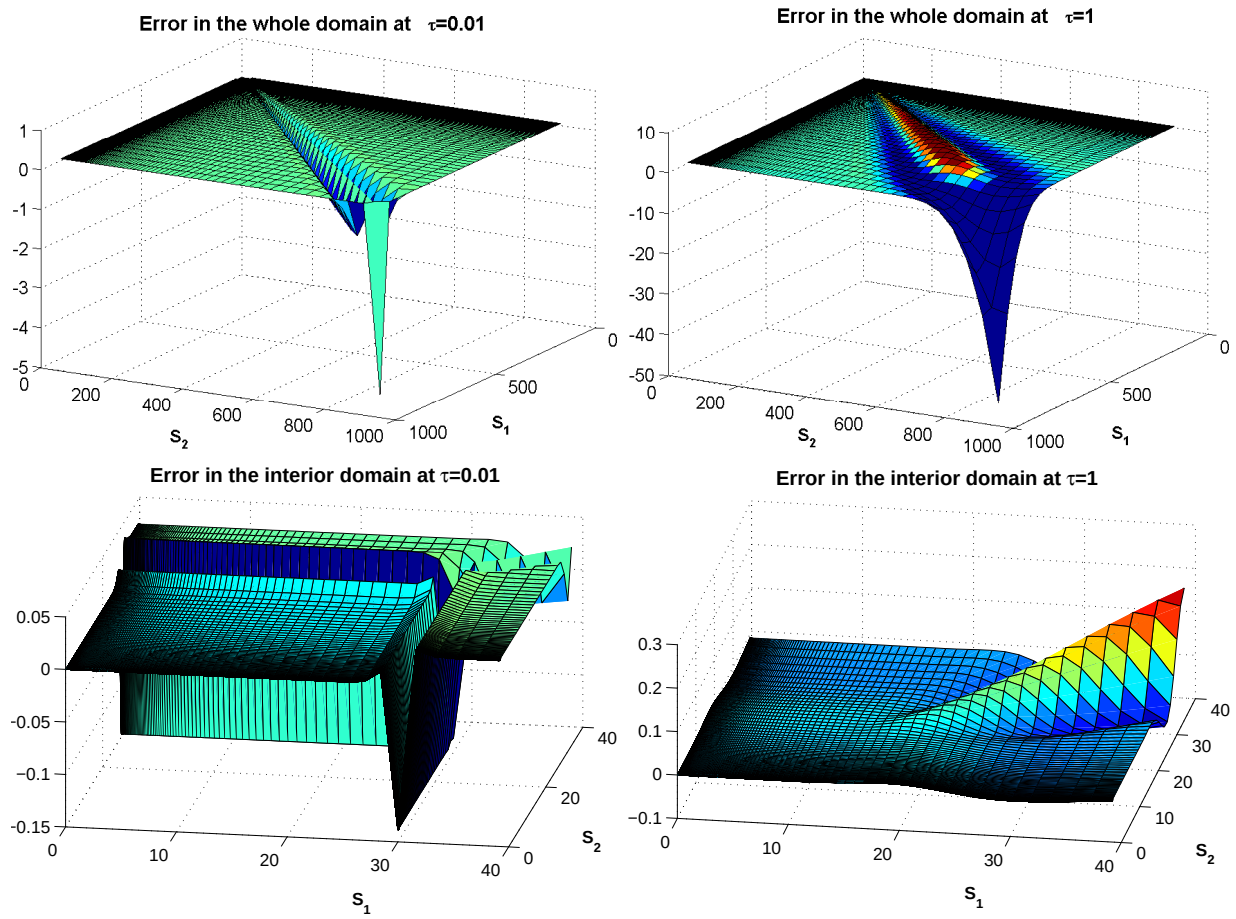

Figure 5.7: Illustration of the point-wise error (with respect to the solution of the untruncated Black-Scholes problem) in the numerical solution for a call on the maximum of two assets with $E = 30$, $T = 1$, $\sigma_1 = 0.25$, $\sigma_2 = 0.3$, $\rho = 0.9$, $r = 0.1$, obtained by the implicit Euler method with computational parameters $M = 128$, $R = 2\ln 30$, $k = 0.01$, boundary function $\tilde{g}_2$, $\nu_1 = \nu_2 = 10$, $\nu = 1$, and $\omega = 0.8$. The error is shown at the first non-zero (left) and the last (right) time step, in the whole computational domain (top) and within the domain $[\frac{1}{40}, 40]^2$ (bottom), respectively.

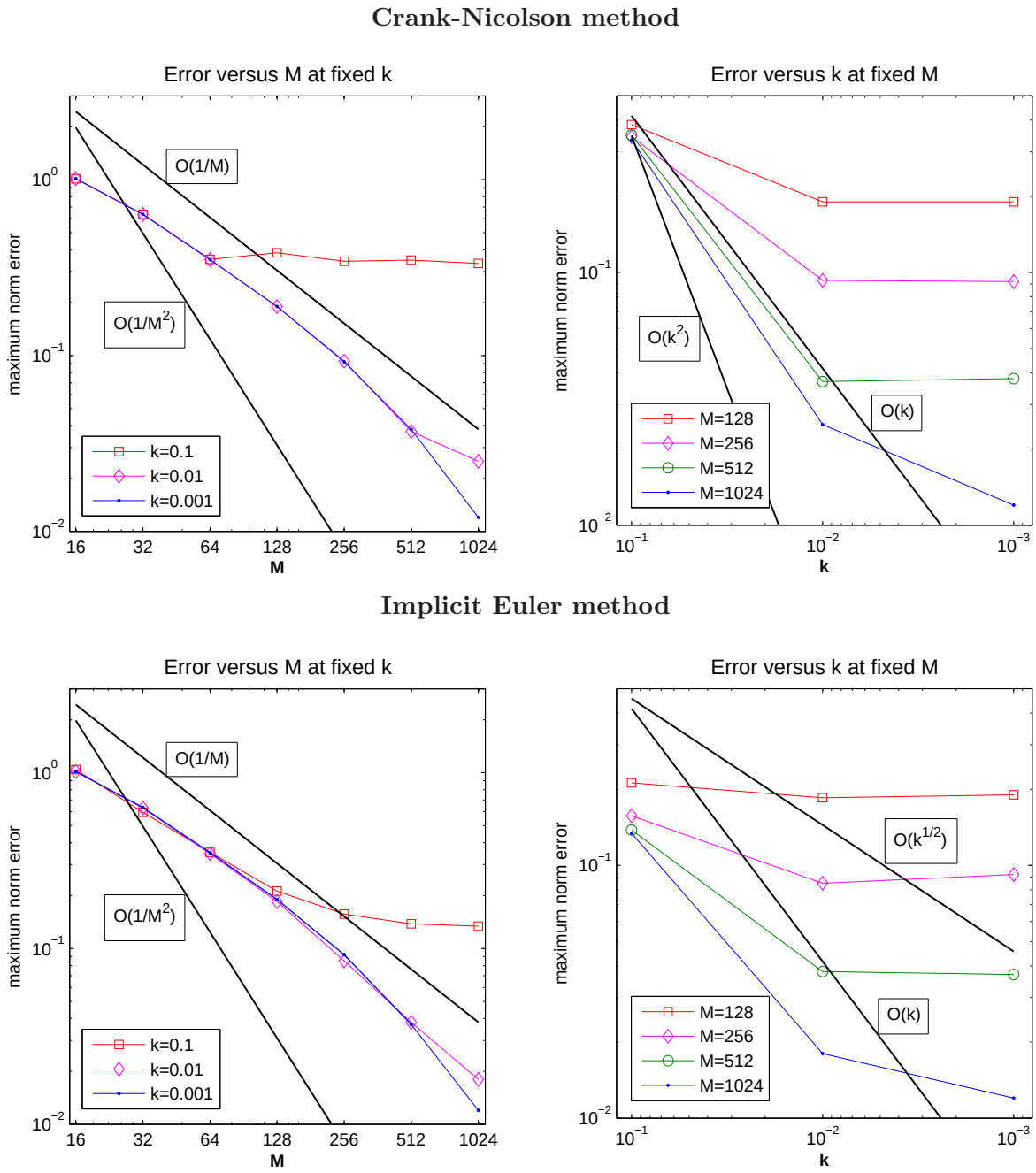

Figure 5.8: Plots of the maximum norm errors (with respect to the solution of the untruncated Black-Scholes problem) within $[\frac{1}{40}, 40]^2 \times (0 : 0.1 : 1)$ obtained for an option on the maximum of two assets with parameters as in Table 5.9 (i.e. $E = 30$, $\sigma_1 = 0.25$, $\sigma_2 = 0.3$, $\rho = 0.9$, $r = 0.1$, $R = 2 \ln 30$, $\nu_1 = \nu_2 = 10$, $\nu = 1$, and $\omega = 0.8$) against the space and time grid parameter, respectively, for the Crank-Nicolson and the implicit Euler methods. Compared to Table 5.9, the experiment with parameters $M = 1024$, $k = 0.001$ is added.

M	h	$k = 0.1$ multigrid error in $\overline{\Omega_R} \times [0,T]$	$k = 0.1$ multigrid error in $[\frac{1}{40}, 40]^2 \times [0,T]$	$k = 0.01$ multigrid error in $\overline{\Omega_R} \times [0,T]$	$k = 0.01$ multigrid error in $[\frac{1}{40}, 40]^2 \times [0,T]$	$k = 0.001$ multigrid error in $\overline{\Omega_R} \times [0,T]$	$k = 0.001$ multigrid error in $[\frac{1}{40}, 40]^2 \times [0,T]$
4	1.84	8.2e − 11	1.5e − 11	8.5e − 10	2.1e − 10	8.5e − 09	2.2e − 09
8	0.92	5.6e − 11	3.8e − 12	5.3e − 10	3.0e − 11	5.3e − 09	2.9e − 10
16	0.46	5.2e − 11	1.7e − 12	2.4e − 10	3.3e − 12	2.3e − 09	3.0e − 11
32	0.23	3.3e − 09	1.2e − 10	8.1e − 11	2.2e − 12	5.3e − 10	1.3e − 11
64	0.12	1.8e − 05	6.8e − 07	2.4e − 10	7.3e − 12	1.1e − 10	3.0e − 12
128	0.058	3.4e − 02	8.9e − 04	5.3e − 08	1.6e − 09	5.9e − 11	1.6e − 12
256	0.029	9.4e − 01	8.3e − 03	2.7e − 04	6.2e − 06		
512	0.014	3.1e + 00	1.1e − 02				

Table 5.10: The error in the multigrid method, calculated as the maximum norm of the difference between the solution obtained by our Crank-Nicolson solver and the solution obtained by applying the MATLAB direct solver on the linear systems (4.3) resulting from the Crank-Nicolson method. The results refer to an option on the maximum of two assets with $E = 30$, $T = 1$, $\sigma_1 = 0.25$, $\sigma_2 = 0.3$, $\rho = 0.9$, $r = 0.1$, $R = 2 \ln 40$ and boundary function $\tilde{g}_2$. The multigrid method is used with the parameters $\nu_1 = \nu_2 = 10$, $\nu = 1$, and $\omega = 0.8$.

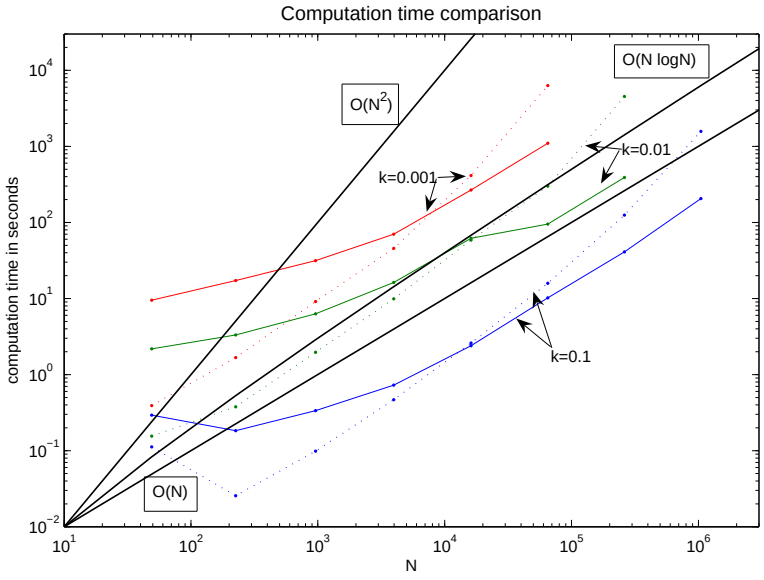

Figure 5.9: Computation time for the numerical solution with our solver (solid lines) and with the MATLAB built-in direct solver (dashed lines) plotted against the number $N = (2M-1)^2$ of unknowns in the linear system for the experiments listed in Table 5.10. The computation time is measured with the MATLAB built-in stopwatch timer functions `tic` and `toc` (see [MATLAB 7.7.0 Help, 2008]).

call on the maximum

Implicit Euler $R = 2\ln 40$			error in $\overline{\Omega_R}$ $\times (0 : 0.1 : 1)$	error in $[\frac{1}{40}, 40]^3$ $\times (0 : 0.1 : 1)$	value at $\mathbf{S} = (40, 40, 40)$, $\tau = 1$ (exact: **20.153**)	error at $\mathbf{S} = (40, 40, 40)$, $\tau = 1$
M	h	k				
8	0.92	0.1	289.878	1.885	20.131	−0.022
8	0.92	0.01	289.878	1.879	20.216	+0.063
8	0.92	0.001	289.878	1.879	20.225	+0.072
8	0.92	0.0001	289.878	1.879	20.226	+0.073
16	0.46	0.1	289.878	1.237	20.149	−0.004
16	0.46	0.01	289.878	1.181	20.282	+0.129
16	0.46	0.001	289.878	1.176	20.296	+0.143
16	0.46	0.0001	289.878	1.176	20.297	+0.144
32	0.23	0.1	289.878	0.803	20.237	+0.084
32	0.23	0.01	289.878	0.649	20.346	+0.193
32	0.23	0.001	289.878	0.630	20.357	+0.204
64	0.12	0.1	289.878	0.436	20.138	−0.015
64	0.12	0.01	289.878	0.203	20.232	+0.079

call on the minimum

Implicit Euler $R = 2\ln 40$			error in $\overline{\Omega_R}$ $\times (0 : 0.1 : 1)$	error in $[\frac{1}{40}, 40]^3$ $\times (0 : 0.1 : 1)$	value at $\mathbf{S} = (40, 40, 40)$, $\tau = 1$ (exact: **7.172**)	error at $\mathbf{S} = (40, 40, 40)$, $\tau = 1$
M	h	k				
8	0.92	0.1	271.770	2.404	9.021	+1.849
8	0.92	0.01	271.770	2.393	8.998	+1.826
8	0.92	0.001	271.770	2.392	8.996	+1.824
8	0.92	0.0001	271.770	2.392	8.996	+1.824
16	0.46	0.1	271.770	1.807	8.100	+0.928
16	0.46	0.01	271.770	1.756	8.028	+0.856
16	0.46	0.001	271.770	1.750	8.020	+0.848
16	0.46	0.0001	271.770	1.750	8.019	+0.847
32	0.23	0.1	271.770	0.955	7.235	+0.063
32	0.23	0.01	271.770	0.813	7.176	+0.004
32	0.23	0.001	271.770	0.795	7.170	−0.002
64	0.12	0.1	271.770	0.514	7.207	+0.035
64	0.12	0.01	271.770	0.250	7.169	−0.003

Table 5.11: Results obtained with the implicit Euler method for options on the maximum and the minimum of three assets with exercise price $E = 30$, $T = 1$, $\sigma_1 = 0.25$, $\sigma_2 = 0.3$, $\sigma_3 = 0.35$, $\rho_{12} = \rho_{23} = 0.6$, $\rho_{13} = 0.4$, $r = 0.1$, $R = 2\ln 40$, i.e. $S_{\max} = \$1600$, boundary function $\tilde{g}_2$, $\nu_1 = \nu_2 = 12$, $\nu = 1$, and $\omega = \frac{6}{7}$. The error is with respect to the solution of the untruncated Black-Scholes problem and measured in the maximum norm within domains.

Implicit Euler method $R = 2\ln 100$			**call on the arithmetic average**	**call on the geometric average**
M	h	k	value at $\mathbf{S} = (100, 100, 100)$, $\tau = 1$	value at $\mathbf{S} = (100, 100, 100)$, $\tau = 1$
8	1.15	0.1	10.999	8.129
8	1.15	0.01	11.024	8.147
8	1.15	0.001	11.027	8.149
8	1.15	0.0001	11.027	8.149
16	0.58	0.1	10.621	9.589
16	0.58	0.01	10.678	9.647
16	0.58	0.001	10.684	9.653
16	0.58	0.0001	10.684	9.654
32	0.29	0.1	11.675	11.008
32	0.29	0.01	11.784	11.119
32	0.29	0.001	11.796	11.130
32	0.29	0.0001	11.797	11.131
64	0.14	0.1	11.965	11.398
64	0.14	0.01	12.048	11.483
64	0.14	0.001	12.056	11.492
128	0.072	0.1	11.994	11.469
128	0.072	0.01	12.072	11.551

Table 5.12: Point-wise results obtained with the implicit Euler method for calls on the arithmetic and the geometric average of three assets with exercise price $E = 100$, $T = 1$, $\sigma_i = 0.2$ for all i, $\rho_{ij} = 0.2$ for all $i \neq j$, $r = 0.1$, $R = 2\ln 100$, i.e. $S_{\max} = \$10000$, boundary function $\tilde{g}_1$, $\nu_1 = \nu_2 = 12$, $\nu = 1$, and $\omega = \frac{6}{7}$. For the option on the geometric average, [Boyle et al., 1989] provide the exact solution 11.581.

Chapter 6

Conclusion

This thesis has developed a numerical solution method for the multivariate Black-Scholes PDE based on a finite-difference discretisation of a log-transform of the Black-Scholes PDE after truncation to a bounded domain, and on iterative solution of the resulting linear systems using the full multigrid V-cycle method. We have proven the existence of a unique solution of the original and the truncated differential problem and provided an analytical estimate of the localisation error by generalising the approach by [Kangro and Nicolaides, 2000]. We found that the truncated problem is not regular enough to ensure convergence of the finite-difference scheme according to the standard convergence theory of such numerical schemes. However, numerical experiments in MATLAB in one, two, and three dimensions demonstrated that the implicit Euler discretisation employed with a suitable choice of computational parameters has satisfactory convergence properties even if the theoretically optimal convergence order is not attained. Hence, we were able to price realistic multi-asset options to an accuracy of few cents.

A main objective of our work was to design a numerical method that can simultaneously be used for any number of underlying assets, and we have shown that all required algorithms can indeed be expressed with the number of underlyings as an a-priori unknown parameter, and presented a corresponding MATLAB implementation.

In theory, our method is hence applicable for an arbitrary number of underlying assets. In practice, however, applicability of the method is limited by the computational capacity available. As we compute the numerical solution on the grid points of a uniform grid in space and time, the number of unknowns is exponential in the space dimension. This "curse of dimensionality", as [Lötstedt et al., 2007] call it, constitutes a serious limitation to any finite-difference or finite element based numerical method. Alternatively, methods based on the second approach to the Black-Scholes model mentioned in the introduction can be used, where the option value at a specified point $(\mathbf{S}, \tau)$ is given in terms of a multi-dimensional integral that can be numerically evaluated using, e.g., Monte-Carlo methods such as the one presented in [Barraquand, 1995]. With such methods, options on as many as several hundred underlying assets can be valued, albeit only at a single point $(\mathbf{S}, \tau)$ at a time. PDE-based methods, in contrast, simultaneously yield the option price on every grid point of a numerical grid. Thus, even though the finite-difference approach is impracticable in high dimensions, [Lötstedt et al., 2007] declare that they consider it the best choice for problems in up to five space dimensions, amongst other reasons because the finite-difference solution is smooth enough to allow for the calculation of derivatives of the option price. These derivatives, called "the Greeks", are needed in hedging strategies (see also, e.g., [Wilmott et al.,

1995]). Both types of methods thus have their justification and applications.

The performance of the numerical solver presented here might be enhanced in several ways.

Using an adaptive method in generating the computational grid, as done, e.g., by [Persson and von Sydow, 2007] and [Lötstedt et al., 2007], could reduce the computational effort necessary to achieve a certain accuracy. Adaptive grids in space are not well compatible with the multigrid method used here, yet introducing an adaptive time stepping method should not cause major complications and would allow to deal with the initial disturbances caused by the non-smooth initial condition without imposing an excessively fine time step k during the whole time interval.

Furthermore, more sophisticated variants of the multigrid method might be considered, involving, e.g., different smoothing operations, intergrid transfer operators, or adaptive regulation of the number of smoothing operations performed on each grid. We refer to the vast literature on multigrid methods for possible modifications.

This list of possible enhancements is by no means complete. The main aim of this thesis was to present how an algorithm involving the space dimension as a mere parameter can be devised. For detailed information, the full code is given in the appendix.

Appendix A

MATLAB Code

Listing A.1: MATLAB implementation of the Black-Scholes solver using the Crank-Nicolson method (Algorithm 10)

```
function W = CN( n,M,R,k,T,r,sigma,w0,g,nu1,nu2,nu,omega )
% This function performs the Crank-Nicolson method for the transformed
% Black-Scholes problem for an n-asset option with transformed pay-off w0
% and boundary function g, truncated to [-R,R]^n x [0,T], on a mesh of
% spatial mesh width R/M and time step k.
% The resulting linear systems are numerically solved with the full
% multigrid V-cycle method based on weighted Jacobi iterations, n-linear
% interpolation and full weighting restriction.
%
% Claudia Satke 2009

K = T/k;
W = zeros((2*M+1)^n,K+1);

% set W0 by evaluating w0 at all grid points:
y = repmat( (0:(2*M+1)^n-1)', 1,n);
aux = y./repmat( (2*M+1).^(n:-1:1), (2*M+1)^n,1);
coord = R/M*floor((2*M+1)*aux) - R/M*(2*M+1)*floor(aux) - R;
W(:,1) = w0(coord);

% find interior and boundary grid point indices:
M0 = repmat( (0:((2*M+1)^(n-1)-1))', 1,n);
pow = (2*M+1).^(n-1:-1:0);
M4 = repmat( pow, (2*M+1)^(n-1),1);
leftbd = 2*M* M4.*floor( M0./M4 ) + M0 + 1;
bdind = reshape( [leftbd; leftbd+2*M*M4], 2*n*(2*M+1)^(n-1),1);
bdind = unique(bdind);
intind = setdiff((1:(2*M+1)^n)', bdind);

% find boundary coordinates for g evaluations
% and evaluate boundary values at t=0:
bdcoord = coord(bdind,:);
g2 = zeros((2*M+1)^n,1);
g2(bdind) = g(bdcoord,0);

```

```
% generate the row entries of the matrix A
    h = R/M;        % = mesh width
    increments = [0, pow, -pow];
    vals = [ r + sum(diag(sigma))/h^2, ...                          % diagonal
        -diag(sigma)'/(2*h^2) + diag(sigma)'/(4*h) - r/(2*h), ...   % 1-step neighbours
        -diag(sigma)'/(2*h^2) - diag(sigma)'/(4*h) + r/(2*h)];
    if n>1          % increments corresponding to 2-step neighbours:
        dimensions = zeros( 2, n*(n-1)/2);          % initialise the matrix
                                                    % of all (i,j) combinations
        temp = repmat(1:n,n,1);
        dimensions(1,:) = temp(tril(temp,-1)>0);    % lower triangular part,
                    % starting from the first subdiagonal, then the
                    % non-zero values are extracted as a row vector
                    % (automatically column-wise ordered)
        temp = temp';
        dimensions(2,:) = temp(tril(temp,-1)>0);    % instead of triu + row-wise
                    % extracting: tril of transposed matrix and column-wise extracting
        inc = (2*M+1).^(n-dimensions);
        increments = [increments, sum(inc,1), - sum(inc,1), ...
            inc(1,:) - inc(2,:), - inc(1,:) + inc(2,:)];
        vals = [vals, ...
          - sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2), ...
          - sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2), ...
            sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2), ...
            sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2)];
    end
% now: vals contains the row entries of A(intind,incrind)
    kB0vals = - k/2*vals;
    kB0vals(1) = kB0vals(1) + 1;
% now: kB0vals contains the row entries of (I-k/2*A)(intind,incrind)

%generate the indices to extract the corresponding entries of W
N = (2*M-1)^n;      % length of intind
incrind = repmat( intind, 1, 2*n^2 + 1 ) + repmat( increments, N,1);

% Crank-Nicolson scheme
for i=1:K
    % generate the boundary value vectors g^l and g^(l+1):
    g1 = g2;
    g2 = zeros((2*M+1)^n,1);
    g2(bdind) = g(bdcoord,i*k);
    % generate the right-hand side:
    Wi = zeros((2*M+1)^n,1);
    Wi(intind)=W(intind,i);
    RHS = Wi(incrind) * kB0vals' - k/2*( g1(incrind) + g2(incrind) ) * vals';
    % call the full multigrid V-cycle function to compute W^(l+1):
    W(:,i+1) = FMG( n,M,R,k,r,sigma,RHS,g2,nu1,nu2,nu,omega );
end
```

Listing A.2: MATLAB implementation of the full multigrid V-cycle method for use in the CN function (Algorithm 11)

```
function W = FMG( n,M,R,k,r,sigma,RHS,BVs,nu1,nu2,nu,omega )
% This function performs the full multigrid V-cycle for the equation
% (I+k/2*A_h(ii)) W,int = RHS,  W,bd = BVs
% on the n-dimensional grid of spatial mesh width h=R/M, using weighted
% Jacobi smoothing, n-linear interpolation and full weighting restriction.
%
% Claudia Satke 2009

W = zeros((2*M+1)^n,1);

% find interior and boundary grid point indices:
M0 = repmat( (0:((2*M+1)^(n-1)-1))', 1,n);
M4 = repmat( (2*M+1).^(n-1:-1:0), (2*M+1)^(n-1),1);
leftbd = 2*M* M4.*floor( M0./M4 ) + M0 + 1;
bdind = reshape( [leftbd; leftbd+2*M*M4], 2*n*(2*M+1)^(n-1),1);
bdind = unique(bdind);
intind = setdiff(1:(2*M+1)^n, bdind);

% full multigrid V-cycle algorithm:

if M > 1
    % generate the prolongation matrix P:
        % apply Algorithm 8 with the parameter M+2:
        pow = (M+1).^(n-1:-1:0);
        ind = floor( rem( repmat( (0:((M+1)^n-1))', 1,n),...
                          repmat( (M+1)*pow, (M+1)^n,1) )...
                    ./ repmat( pow, (M+1)^n,1)            ) + 1;
        ind = ind';
        intcoarse = ((  M+3).^(n-1:-1:0))*  ind + 1;
        intfine   = ((2*M+5).^(n-1:-1:0))*2*ind + 1;
        comb = floor( rem( repmat( (0:(3^n -1))', 1,n),...
                           repmat( 3.^(n:-1:1), 3^n,1) )...
                      ./ repmat( 3.^(n-1:-1:0), 3^n,1)   ) - 1;
        incr = comb*((2*M+5).^(n-1:-1:0)');
        I = repmat( intfine, 3^n,1) + repmat( incr, 1,(M+1)^n);
        J = repmat( intcoarse, 3^n,1);
        Pvals = 2.^(-sum(abs(comb),2));
        P = sparse( I(:), J(:), repmat( Pvals, (M+1)^n,1), (2*M+5)^n, (M+3)^n );
        % delete rows corresp. to boundary grid points in the mesh with M+2:
        M0 = repmat( (0:((2*M+5)^(n-1)-1))', 1,n);
        M4 = repmat( (2*M+5).^(n-1:-1:0), (2*M+5)^(n-1),1);
        leftbd = 2*(M+2)* M4.*floor( M0./M4 ) + M0 + 1;
        bdind2 = reshape( [leftbd; leftbd+2*(M+2)*M4], 2*n*(2*M+5)^(n-1),1);
        intind2 = setdiff(1:(2*M+5)^n, bdind2);
        P = P(intind2,:);
        % delete rows corresp. to boundary grid points in the mesh with M+1:
        M0 = repmat( (0:((2*M+3)^(n-1)-1))', 1,n);
        M4 = repmat( (2*M+3).^(n-1:-1:0), (2*M+3)^(n-1),1);
        leftbd = 2*(M+1)* M4.*floor( M0./M4 ) + M0 + 1;
        bdind2 = reshape( [leftbd; leftbd+2*(M+1)*M4], 2*n*(2*M+3)^(n-1),1);
        intind2 = setdiff(1:(2*M+3)^n, bdind2);
        P = P(intind2,:);
        % delete columns corresp. to boundary grid points in the mesh with M/2+1:
```

```
54          M0 = repmat( (0:((M+3)^(n-1)-1))', 1,n);
55          M4 = repmat( (M+3).^(n-1:-1:0), (M+3)^(n-1),1);
56          leftbd = 2*(M/2+1)* M4.*floor( M0./M4 ) + M0 + 1;
57          bdind2 = reshape( [leftbd; leftbd+2*(M/2+1)*M4], 2*n*(M+3)^(n-1),1);
58          intind2 = setdiff(1:(M+3)^n, bdind2);
59          P = P(:,intind2);
60      % use only the part corresponding to interior grid points to restrict
61      % RHS to the coarser mesh Omega_2h:
62          M0 = repmat( (0:((M+1)^(n-1)-1))', 1,n);
63          M4 = repmat( (M+1).^(n-1:-1:0), (M+1)^(n-1),1);
64          leftbd = M* M4.*floor( M0./M4 ) + M0 + 1;
65          bdind2h_coarse = reshape( [leftbd; leftbd+M*M4], 2*n*(M+1)^(n-1),1);
66          bdind2h_coarse = unique(bdind2h_coarse);
67          intind2h = setdiff(1:(M+1)^n, bdind2h_coarse);
68      RHS2h = P(intind,intind2h)'*RHS/2^n;
69      % extract the correct boundary values on the coarse grid from BVs:
70          % find fine-grid indices of the boundary grid points which belong to
71          % the coarse grid as well:
72          y = repmat( bdind2h_coarse-1, 1,n);
73          aux = y./repmat( (M+1).^(n:-1:1),length(y),1 );
74          alpha = floor((M+1)*aux) - (M+1)*floor(aux) - M/2;
75          bdind2h_fine = ((2*M+1).^(n-1:-1:0))*(2*alpha'+M) + 1;
76      BVs2h = zeros((M+1)^n,1);
77      BVs2h(bdind2h_coarse(:)) = BVs(bdind2h_fine(:));
78      % call FMG on the coarse grid and correct W by the interpolated result:
79      W2h = FMG( n,M/2,R,k,r,sigma,RHS2h,BVs2h,nu1,nu2,nu,omega );
80      W = W + P*W2h;
81  end
82  % set the boundary values of W right again:
83  W(bdind) = BVs(bdind);
84
85  for i=1:nu
86      W(intind) = MG(n,M,R,k,r,sigma,W(intind),RHS,nu1,nu2,1,omega);
87  end
```

Listing A.3: MATLAB implementation of the multigrid μ-cycle method for the linear system resulting from the Crank-Nicolson discretisation (Algorithm 12)

```
function W = MG(n,M,R,k,r,sigma,W0,RHS,nu1,nu2,mu,omega)
% This function performs the multigrid mu-cycle for the equation
% (I+k/2*A)W,int = RHS,int
% on the n-dimensional grid of spatial mesh width h=R/M, using weighted
% Jacobi smoothing, n-linear interpolation and full weighting restriction.
%
% Claudia Satke 2009

if M == 1           % on the coarsest grid: only 1 interior component,
                    % 1x1 linear system is explicitly solved:
    W = RHS /( 1 + k/2* (r + sum(diag(sigma))*M^2/R^2) );
else
    % initialising
        W = W0;
    % pre-smoothing
        % find interior and boundary grid point indices:
        M0 = repmat( (0:((2*M+1)^(n-1)-1))', 1,n);
        pow = (2*M+1).^(n-1:-1:0);
        M4 = repmat( pow, (2*M+1)^(n-1),1);
        leftbd = 2*M* M4.*floor( M0./M4 ) + M0 + 1;
        bdind = reshape( [leftbd; leftbd + 2*M*M4], 2*n*(2*M+1)^(n-1),1);
        intind = setdiff((1:(2*M+1)^n)', bdind);
        % generate entries in the intind-rows of B1:
        h = R/M;        % = mesh width
        increments = [0, pow, -pow];
        vals = [ r + sum(diag(sigma))/h^2, ...                      % diagonal
        -diag(sigma)'/(2*h^2) + diag(sigma)'/(4*h) - r/(2*h), ...   % 1-step neighbours
        -diag(sigma)'/(2*h^2) - diag(sigma)'/(4*h) + r/(2*h)];
        if n>1          % increments corresponding to 2-step neighbours:
            dimensions = zeros( 2, n*(n-1)/2);      % initialise the matrix
                                                    % of all (i,j) combinations
            temp = repmat(1:n,n,1);
            dimensions(1,:) = temp(tril(temp,-1)>0); % lower triangular part,
                    % starting from the first subdiagonal, then the
                    % non-zero values are extracted as a row vector
                    % (automatically column-wise ordered)
            temp = temp';
            dimensions(2,:) = temp(tril(temp,-1)>0); % instead of triu + row-wise
                    % extracting: tril of transposed matrix and column-wise extracting
            inc = (2*M+1).^(n-dimensions);
            increments = [increments, sum(inc,1), - sum(inc,1), ...
                inc(1,:) - inc(2,:), - inc(1,:) + inc(2,:)];
            vals = [vals, ...
                - sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2), ...
                - sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2), ...
                  sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2), ...
                  sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2)];
        end
        % now: vals contains the row entries of A(intind,incrind)
        kB1vals = k/2*vals;
        kB1vals(1) = kB1vals(1) + 1;
        % now: kB1vals contains the row entries of (I+k/2*A)(intind,incrind)

```

```
54          % extend W by zero boundary values
55          W_BV = zeros((2*M+1)^n,1);
56          W_BV( intind ) = W;
57          %generate the indices to extract the corresponding entries of W_BV
58          N = (2*M-1)^n;       % length of intind
59          incrind = repmat( intind, 1, 2*n^2 + 1 ) + repmat( increments, N,1);
60
61          % execute nu1 weighted Jacobi steps:
62          for j = 1:nu1
63              res = RHS - W_BV( incrind ) * kB1vals';
64              W = W + omega/kB1vals(1) * res;
65              W_BV( intind ) = W;
66          end
67
68      % generate the prolongation matrix P:
69          pow = (M-1).^(n-1:-1:0);
70          ind = floor( rem( repmat( (0:((M-1)^n-1))', 1,n),...
71                            repmat( (M-1)*pow, (M-1)^n,1) )...
72                       ./ repmat( pow, (M-1)^n,1)           ) + 1;
73          ind = ind';
74          intcoarse = ((  M+1).^(n-1:-1:0))*  ind + 1;
75          intfine   = ((2*M+1).^(n-1:-1:0))*2*ind + 1;
76          comb = floor( rem( repmat( (0:(3^n -1))', 1,n),...
77                             repmat( 3.^(n:-1:1), 3^n,1) )...
78                        ./ repmat( 3.^(n-1:-1:0), 3^n,1)      ) - 1;
79          incr = comb*((2*M+1).^(n-1:-1:0)');
80          I = repmat( intfine, 3^n,1) + repmat( incr, 1,(M-1)^n);
81          J = repmat( intcoarse, 3^n,1);
82          Pvals = 2.^(-sum(abs(comb),2));
83          P = sparse( I(:), J(:), repmat( Pvals, (M-1)^n,1), (2*M+1)^n, (M+1)^n );
84          % retain only the components corresponding to interior grid points:
85          P = P(intind,intcoarse);
86      % calculate the residual and restrict it to the coarse grid Omega_2h:
87          res = RHS - W_BV( incrind ) * kB1vals';
88          res = P'*res/2^n;
89      % initialise the error E2h:
90          E2h = zeros((M-1)^n,1);
91      % call MG mu times for the error E2h on the coarse grid Omega_2h and
92      % the restricted residual as RHS:
93          for i = 1:mu
94              E2h = MG(n,M/2,R,k,r,sigma,E2h,res,nu1,nu2,mu,omega);
95          end
96      % correcting W with the interpolated error estimate
97          W = W + P*E2h;
98      % post-smoothing:
99          %execute nu2 weighted Jacobi steps
100         W_BV( intind ) = W;
101         for j = 1:nu2
102             res = RHS - W_BV( incrind ) * kB1vals';
103             W = W + omega/kB1vals(1) * res;
104             W_BV( intind ) = W;
105         end
106 end
```

Listing A.4: MATLAB implementation of the Black-Scholes solver using the implicit Euler method (Algorithm 13)

```
function W = IE( n,M,R,k,T,r,sigma,w0,g,nu1,nu2,nu,omega )
% This function performs the implicit Euler method for the transformed
% Black-Scholes problem for an n-asset option with transformed pay-off w0
% and boundary function g, truncated to [-R,R]^n x [0,T], on a mesh of
% spatial mesh width R/M and time step k.
% The resulting linear systems are numerically solved with the full
% multigrid V-cycle method based on weighted Jacobi iterations, n-linear
% interpolation and full weighting restriction.
%
% Claudia Satke 2009

K = T/k;
W = zeros((2*M+1)^n,K+1);

% set W0 by evaluating w0 at all grid points:
y = repmat( (0:(2*M+1)^n-1)', 1,n);
aux = y./repmat( (2*M+1).^(n:-1:1), (2*M+1)^n,1);
coord = R/M*floor((2*M+1)*aux) - R/M*(2*M+1)*floor(aux) - R;
W(:,1) = w0(coord);

% find interior and boundary grid point indices:
M0 = repmat( (0:((2*M+1)^(n-1)-1))', 1,n);
pow = (2*M+1).^(n-1:-1:0);
M4 = repmat( pow, (2*M+1)^(n-1),1);
leftbd = 2*M* M4.*floor( M0./M4 ) + M0 + 1;
bdind = reshape( [leftbd; leftbd+2*M*M4], 2*n*(2*M+1)^(n-1),1);
bdind = unique(bdind);
intind = setdiff((1:(2*M+1)^n)', bdind);

% find boundary coordinates for g evaluations:
bdcoord = coord(bdind,:);

% generate the row entries of the matrix A
    h = R/M;        % = mesh width
    increments = [0, pow, -pow];
    vals = [ r + sum(diag(sigma))/h^2, ...                          % diagonal
        -diag(sigma)'/(2*h^2) + diag(sigma)'/(4*h) - r/(2*h), ...   % 1-step neighbours
        -diag(sigma)'/(2*h^2) - diag(sigma)'/(4*h) + r/(2*h)];
    if n>1          % increments corresponding to 2-step neighbours:
        dimensions = zeros( 2, n*(n-1)/2);          % initialise the matrix
                                                    % of all (i,j) combinations
        temp = repmat(1:n,n,1);
        dimensions(1,:) = temp(tril(temp,-1)>0);    % lower triangular part,
                    % starting from the first subdiagonal, then the
                    % non-zero values are extracted as a row vector
                    % (automatically column-wise ordered)
        temp = temp';
        dimensions(2,:) = temp(tril(temp,-1)>0);    % instead of triu + row-wise
                    % extracting: tril of transposed matrix and column-wise extracting
        inc = (2*M+1).^(n-dimensions);
        increments = [increments, sum(inc,1), - sum(inc,1), ...
            inc(1,:) - inc(2,:), - inc(1,:) + inc(2,:)];
        vals = [vals, ...
```

```
          - sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2), ...
          - sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2), ...
            sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2), ...
            sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2)];
    end
% now: vals contains the row entries of A(intind,incrind) (see below)

%generate the indices to extract the corresponding entries of W
N = (2*M-1)^n;      % length of intind
incrind = repmat( intind, 1, 2*n^2 + 1 ) + repmat( increments, N,1);

% Implicit Euler scheme
for i=1:K
    % generate the boundary value vector g^(l+1):
    g2 = zeros((2*M+1)^n,1);
    g2(bdind) = g(bdcoord,i*k);
    % generate the right-hand side:
    RHS = W(intind,i) - k*g2(incrind)* vals';
    % call the full multigrid V-cycle function to compute W^(l+1):
    W(:,i+1) = FMG_IE( n,M,R,k,r,sigma,RHS,g2,nu1,nu2,nu,omega );
end
```

Listing A.5: MATLAB implementation of the full multigrid V-cycle method for use in the IE function

```
function W = FMG_IE( n,M,R,k,r,sigma,RHS,BVs,nu1,nu2,nu,omega )
% This function performs the full multigrid V-cycle for the equation
% (I+k*A_h(ii)) W,int = RHS,  W,bd = BVs
% on the n-dimensional grid of spatial mesh width h=R/M, using weighted
% Jacobi smoothing, n-linear interpolation and full weighting restriction.
%
% Claudia Satke 2009

W = zeros((2*M+1)^n,1);

% find interior and boundary grid point indices:
M0 = repmat( (0:((2*M+1)^(n-1)-1))', 1,n);
M4 = repmat( (2*M+1).^(n-1:-1:0), (2*M+1)^(n-1),1);
leftbd = 2*M* M4.*floor( M0./M4 ) + M0 + 1;
bdind = reshape( [leftbd; leftbd+2*M*M4], 2*n*(2*M+1)^(n-1),1);
bdind = unique(bdind);
intind = setdiff(1:(2*M+1)^n, bdind);

% full multigrid V-cycle algorithm:

if M > 1
    % generate the prolongation matrix P:
        % apply Algorithm 8 with the parameter M+2:
        pow = (M+1).^(n-1:-1:0);
        ind = floor( rem( repmat( (0:((M+1)^n-1))', 1,n),...
                          repmat( (M+1)*pow, (M+1)^n,1) )...
                    ./ repmat( pow, (M+1)^n,1)            ) + 1;
        ind = ind';
        intcoarse = ((  M+3).^(n-1:-1:0))*  ind + 1;
        intfine   = ((2*M+5).^(n-1:-1:0))*2*ind + 1;
        comb = floor( rem( repmat( (0:(3^n -1))', 1,n),...
                           repmat( 3.^(n:-1:1), 3^n,1) )...
                      ./ repmat( 3.^(n-1:-1:0), 3^n,1)   ) - 1;
        incr = comb*((2*M+5).^(n-1:-1:0)');
        I = repmat( intfine, 3^n,1) + repmat( incr, 1,(M+1)^n);
        J = repmat( intcoarse, 3^n,1);
        Pvals = 2.^(-sum(abs(comb),2));
        P = sparse( I(:), J(:), repmat( Pvals, (M+1)^n,1), (2*M+5)^n, (M+3)^n );
        % delete rows corresp. to boundary grid points in the mesh with M+2:
        M0 = repmat( (0:((2*M+5)^(n-1)-1))', 1,n);
        M4 = repmat( (2*M+5).^(n-1:-1:0), (2*M+5)^(n-1),1);
        leftbd = 2*(M+2)* M4.*floor( M0./M4 ) + M0 + 1;
        bdind2 = reshape( [leftbd; leftbd+2*(M+2)*M4], 2*n*(2*M+5)^(n-1),1);
        intind2 = setdiff(1:(2*M+5)^n, bdind2);
        P = P(intind2,:);
        % delete rows corresp. to boundary grid points in the mesh with M+1:
        M0 = repmat( (0:((2*M+3)^(n-1)-1))', 1,n);
        M4 = repmat( (2*M+3).^(n-1:-1:0), (2*M+3)^(n-1),1);
        leftbd = 2*(M+1)* M4.*floor( M0./M4 ) + M0 + 1;
        bdind2 = reshape( [leftbd; leftbd+2*(M+1)*M4], 2*n*(2*M+3)^(n-1),1);
        intind2 = setdiff(1:(2*M+3)^n, bdind2);
        P = P(intind2,:);
        % delete columns corresp. to boundary grid points in the mesh with M/2+1:
```

```
54          M0 = repmat( (0:((M+3)^(n-1)-1))', 1,n);
55          M4 = repmat( (M+3).^(n-1:-1:0), (M+3)^(n-1),1);
56          leftbd = 2*(M/2+1)* M4.*floor( M0./M4 ) + M0 + 1;
57          bdind2 = reshape( [leftbd; leftbd+2*(M/2+1)*M4], 2*n*(M+3)^(n-1),1);
58          intind2 = setdiff(1:(M+3)^n, bdind2);
59          P = P(:,intind2);
60      % use only the part corresponding to interior grid points to restrict
61      % RHS to the coarser mesh Omega_2h:
62          M0 = repmat( (0:((M+1)^(n-1)-1))', 1,n);
63          M4 = repmat( (M+1).^(n-1:-1:0), (M+1)^(n-1),1);
64          leftbd = M* M4.*floor( M0./M4 ) + M0 + 1;
65          bdind2h_coarse = reshape( [leftbd; leftbd+M*M4], 2*n*(M+1)^(n-1),1);
66          bdind2h_coarse = unique(bdind2h_coarse);
67          intind2h = setdiff(1:(M+1)^n, bdind2h_coarse);
68      RHS2h = P(intind,intind2h)'*RHS/2^n;
69      % extract the correct boundary values on the coarse grid from BVs:
70          % find fine-grid indices of the boundary grid points which belong to
71          % the coarse grid as well:
72          y = repmat( bdind2h_coarse-1, 1,n);
73          aux = y./repmat( (M+1).^(n:-1:1),length(y),1 );
74          alpha = floor((M+1)*aux) - (M+1)*floor(aux) - M/2;
75          bdind2h_fine = ((2*M+1).^(n-1:-1:0))*(2*alpha'+M) + 1;
76      BVs2h = zeros((M+1)^n,1);
77      BVs2h(bdind2h_coarse(:)) = BVs(bdind2h_fine(:));
78      % call FMG on the coarse grid and correct W by the interpolated result:
79      W2h = FMG_IE( n,M/2,R,k,r,sigma,RHS2h,BVs2h,nu1,nu2,nu,omega );
80      W = W + P*W2h;
81  end
82  % set the boundary values of W right again:
83  W(bdind) = BVs(bdind);
84
85  for i=1:nu
86      W(intind) = MG_IE(n,M,R,k,r,sigma,W(intind),RHS,nu1,nu2,1,omega);
87  end
```

Listing A.6: MATLAB implementation of the multigrid μ-cycle method for the linear system resulting from the implicit Euler discretisation

```
function W = MG_IE(n,M,R,k,r,sigma,W0,RHS,nu1,nu2,mu,omega)
% This function performs the multigrid mu-cycle for the equation
% (I+k*A)W,int = RHS,int
% on the n-dimensional grid of spatial mesh width h=R/M, using weighted
% Jacobi smoothing, n-linear interpolation and full weighting restriction.
%
% Claudia Satke 2009

if M == 1           % on the coarsest grid: only 1 interior component,
                    % 1x1 linear system is explicitly solved:
    W = RHS /( 1 + k* (r + sum(diag(sigma))*M^2/R^2) );
else
    % initialising
        W = W0;
    % pre-smoothing
        % find interior and boundary grid point indices:
        M0 = repmat( (0:((2*M+1)^(n-1)-1))', 1,n);
        pow = (2*M+1).^(n-1:-1:0);
        M4 = repmat( pow, (2*M+1)^(n-1),1);
        leftbd = 2*M* M4.*floor( M0./M4 ) + M0 + 1;
        bdind = reshape( [leftbd; leftbd + 2*M*M4], 2*n*(2*M+1)^(n-1),1);
        intind = setdiff((1:(2*M+1)^n)', bdind);
        % generate entries in the intind-rows of A:
        h = R/M;        % = mesh width
        increments = [0, pow, -pow];
        vals = [ r + sum(diag(sigma))/h^2, ...                      % diagonal
        -diag(sigma)'/(2*h^2) + diag(sigma)'/(4*h) - r/(2*h), ...   % 1-step neighbours
        -diag(sigma)'/(2*h^2) - diag(sigma)'/(4*h) + r/(2*h)];
        if n>1          % increments corresponding to 2-step neighbours:
            dimensions = zeros( 2, n*(n-1)/2);      % initialise the matrix
                                                    % of all (i,j) combinations
            temp = repmat(1:n,n,1);
            dimensions(1,:) = temp(tril(temp,-1)>0); % lower triangular part,
                    % starting from the first subdiagonal, then the
                    % non-zero values are extracted as a row vector
                    % (automatically column-wise ordered)
            temp = temp';
            dimensions(2,:) = temp(tril(temp,-1)>0); % instead of triu + row-wise
                    % extracting: tril of transposed matrix and column-wise extracting
            inc = (2*M+1).^(n-dimensions);
            increments = [increments, sum(inc,1), - sum(inc,1), ...
                inc(1,:) - inc(2,:), - inc(1,:) + inc(2,:)];
            vals = [vals, ...
                - sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2), ...
                - sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2), ...
                  sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2), ...
                  sigma(n*dimensions(2,:)-n+dimensions(1,:))/(4*h^2)];
        end
        % now: vals contains the row entries of A(intind,incrind)
        kB1vals = k*vals;
        kB1vals(1) = kB1vals(1) + 1;
        % now: kB1vals contains the row entries of (I+k*A)(intind,incrind)

```

```
54          % extend W by zero boundary values
55          W_BV = zeros((2*M+1)^n,1);
56          W_BV( intind ) = W;
57          %generate the indices to extract the corresponding entries of W_BV
58          N = (2*M-1)^n;      % length of intind
59          incrind = repmat( intind, 1, 2*n^2 + 1 ) + repmat( increments, N,1);
60
61          % execute nu1 weighted Jacobi steps:
62          for j = 1:nu1
63              res = RHS - W_BV( incrind ) * kB1vals';
64              W = W + omega/kB1vals(1) * res;
65              W_BV( intind ) = W;
66          end
67
68      % generate the prolongation matrix P:
69          pow = (M-1).^(n-1:-1:0);
70          ind = floor( rem( repmat( (0:((M-1)^n-1))', 1,n),...
71                            repmat( (M-1)*pow, (M-1)^n,1) )...
72                       ./ repmat( pow, (M-1)^n,1)           ) + 1;
73          ind = ind';
74          intcoarse = ((  M+1).^(n-1:-1:0))*  ind + 1;
75          intfine   = ((2*M+1).^(n-1:-1:0))*2*ind + 1;
76          comb = floor( rem( repmat( (0:(3^n -1))', 1,n),...
77                             repmat( 3.^(n:-1:1), 3^n,1) )...
78                        ./ repmat( 3.^(n-1:-1:0), 3^n,1)      ) - 1;
79          incr = comb*((2*M+1).^(n-1:-1:0)');
80          I = repmat( intfine, 3^n,1) + repmat( incr, 1,(M-1)^n);
81          J = repmat( intcoarse, 3^n,1);
82          Pvals = 2.^(-sum(abs(comb),2));
83          P = sparse( I(:), J(:), repmat( Pvals, (M-1)^n,1), (2*M+1)^n, (M+1)^n );
84          % retain only the components corresponding to interior grid points:
85          P = P(intind,intcoarse);
86      % calculate the residual and restrict it to the coarse grid Omega_2h:
87          res = RHS - W_BV( incrind ) * kB1vals';
88          res = P'*res/2^n;
89      % initialise the error E2h:
90          E2h = zeros((M-1)^n,1);
91      % call MG mu times for the error E2h on the coarse grid Omega_2h and
92      % the restricted residual as RHS:
93          for i = 1:mu
94              E2h = MG_IE(n,M/2,R,k,r,sigma,E2h,res,nu1,nu2,mu,omega);
95          end
96      % correcting W with the interpolated error estimate
97          W = W + P*E2h;
98      % post-smoothing:
99          %execute nu2 weighted Jacobi steps
100         W_BV( intind ) = W;
101         for j = 1:nu2
102             res = RHS - W_BV( incrind ) * kB1vals';
103             W = W + omega/kB1vals(1) * res;
104             W_BV( intind ) = W;
105         end
106 end
```

Bibliography

[Barraquand, 1995] Barraquand, J. (1995). Numerical valuation of high dimensional multivariate European securities. *Management Science*, 41(12):1882–1891.

[Barraquand and Martineau, 1995] Barraquand, J. and Martineau, D. (1995). Numerical valuation of high dimensional multivariate American securities. *Journal of Financial and Quantitative Analysis*, 30(3):383–405.

[Black and Scholes, 1973] Black, F. and Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy*, 81:637–654.

[Boyle et al., 1989] Boyle, P. P., Evnine, J., and Gibbs, S. (1989). Numerical evaluation of multivariate contingent claims. *Review of Financial Studies*, 2(2):241–250.

[Boyle and Tse, 1990] Boyle, P. P. and Tse, Y. K. (1990). An algorithm for computing values of options on the maximum or minimum of several assets. *Journal of Financial and Quantitative Analysis*, 25(2):215–227.

[Brennan and Schwartz, 1978] Brennan, M. J. and Schwartz, E. S. (1978). Finite difference methods and jump processes arising in the pricing of contingent claims: A synthesis. *Journal of Financial and Quantitative Analysis*, 13(3):461–474.

[Briggs, 1987] Briggs, W. L. (1987). *A multigrid tutorial*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

[BusinessDictionary.com, 2009] BusinessDictionary.com (2007-2009). Available at: `http://www.businessdictionary.com` [Accessed 5th May 2009].

[Cox et al., 1979] Cox, J., Ross, S., and Rubinstein, M. (1979). cited in [Boyle et al., 1989].

[Engelmann and Schwendner, 1998] Engelmann, B. and Schwendner, P. (1998). The pricing of multi-asset options using a Fourier grid method. *Journal of Computational Finance*, 1(4):53–61.

[Friedman, 1964] Friedman, A. (1964). *Partial differential equations of parabolic type*. Prentice-Hall, Inc., Englewood Cliffs, N.J.

[Günther and Jüngel, 2003] Günther, M. and Jüngel, A. (2003). *Finanzderivate mit MATLAB®-Mathematische Modellierung und numerische Simulation*. Vieweg, Wiesbaden.

[Hackbusch, 1985] Hackbusch, W. (1985). *Multigrid methods and applications*, volume 4 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin.

[Hackbusch, 1993] Hackbusch, W. (1993). *Iterative Lösung großer schwachbesetzter Gleichungssysteme*. Teubner, Stuttgart, second edition.

[Hilber et al., 2004] Hilber, N., Matache, A.-M., and Schwab, C. (2004). Sparse wavelet methods for option pricing under stochastic volatility. Research Report No. 2004-07, Eidgenössische Technische Hochschule, Zurich, Switzerland.

[Johnson, 1987] Johnson, H. (1987). Options on the maximum or the minimum of several assets. *Journal of Financial and Quantitative Analysis*, 22(3):277–283.

[Kangro and Nicolaides, 2000] Kangro, R. and Nicolaides, R. (2000). Far field boundary conditions for Black–Scholes equations. *SIAM Journal on Numerical Analysis*, 38(4):1357–1368.

[Lötstedt et al., 2007] Lötstedt, P., Persson, J., von Sydow, L., and Tysk, J. (2007). Space-time adaptive finite difference method for European multi-asset options. *Computers and Mathematics with Applications*, 53(8):1159–1180.

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

[MATLAB 7.7.0 Help, 2008] MATLAB 7.7.0 Help (1984-2008). The MathWorks, Inc.

[Morton and Mayers, 1994] Morton, K. and Mayers, D. (1994). *Numerical solution of partial differential equations*. Cambridge University Press, Cambridge.

[Persson and von Sydow, 2007] Persson, J. and von Sydow, L. (2007). Pricing European multi-asset options using a space-time adaptive FD-method. *Computing and Visualization in Science*, 10:173–183.

[Schwartz, 1977] Schwartz, E. S. (1977). The valuation of warrants: Implementing a new approach. *Journal of Financial Economics*, 4:79–93.

[Seydel, 2004] Seydel, R. (2004). *Tools for Computational Finance*. Springer-Verlag, Berlin, second edition.

[Smith, 1978] Smith, G. D. (1978). *Numerical solution of partial differential equations - Finite Difference methods*. Oxford University Press, Oxford, second edition.

[Stulz, 1982] Stulz, R. M. (1982). Options on the minimum or the maximum of two risky assets: Analysis and applications. *Journal of Financial Economics*, 10:161–185.

[Wilmott, 1998] Wilmott, P. (1998). *Derivatives: the theory and practice of financial engineering*. John Wiley & Sons, Chichester.

[Wilmott et al., 1995] Wilmott, P., Howison, S., and Dewynne, J. (1995). *The mathematics of financial derivatives - A student introduction*. Cambridge University Press, Cambridge.