Zurich University of Applied Sciences
School of Management and Law

Department Banking, Finance, Insurance

Bachelor's Thesis

Evaluation of lookback options in the Heston model using the finite difference method in Python

Submitted by
Moritz Leon Schroeter

Supervised by
Dr. Norbert Hilber

Winterthur, May 31st, 2023

Management Summary

Financial markets consist not only of exchanges for publicly traded assets but also comprise decisive private financial transactions with various participants. The fair value of some assets can be determined directly by the market through supply and demand or by simple calculations. On the other hand, the complexity of price determination is significantly exacerbated for the issuance of over-the-counter products without liquid markets, such as lookback options. The value of these options is path-dependent and considers the future movement of the underlying asset's price. Thus, the option price depends on future uncertainty but must be determined at the time of emission. Although an explicit analytical solution for the evaluation is known, the calculation of this solution is highly complex and unwarranted in terms of practicality and efficiency. Thus, the option price must be approximated by numerical-algorithmic processes in a stochastic volatility model such as the Heston model. Despite notable advancements in recent research, the study of lookback options in the Heston model exhibits a limited body of literature and a scarcity of practical implementations.

Hence, this thesis lays the foundation for the evaluation of lookback options by expounding the functionality and payoff of options, the Black-Scholes model with its deficiencies, and the Heston model as an extension. Furthermore, the finite difference approach is introduced as a numerical approximation method for one and two-dimensional problems. First, the numerical method is converted into matrix notation for further application and fundamental properties such as multiple, combinable boundary conditions are covered. Subsequently, an analytical partial differential equation for the pricing of lookback options is transformed from four to three dimensions with a new domain satisfying the requirement of the finite difference method, and a modified payoff function is defined. The implementation of the evaluation entails the construction of univariate functions and the development of the finite difference model in the software Python. Furthermore, the option value is interpolated, and node points at the lower boundary are extrapolated to receive the price of floating lookback options. The price for fixed lookback options is obtained by converting floating lookback option prices through a specific put-call parity. Moreover, the processing time is significantly reduced by applying the Craig-Sneyd scheme with specialised algorithms, and a graphic user interface is developed for enhanced usability.

The developed model is verified on the basis of a Monte Carlo simulation and exhibits accurate results up to the second decimal place with expeditious discretisation processes. In addition, this thesis proposes amended option prices for an erroneous study conducted in 2019, thereby contributing to the field of quantitative finance. The codes for the developed model, graphical user interface, and complex graphics are disclosed in the appendix for extended transparency and further research.

Table of Contents

List of Figures

List of Tables

Notation

$S(t)$	Price of an underlying asset
V	Option's value
K	Strike of option
t	Generic time
T	Expiration date of an option
τ	Term of maturity
r	Continuous risk-free interest rate
q	Continuous dividend yield
g	Payoff function
S_{min}	Minimum underlying asset's price during the lookback period
S_{max}	Maximum underlying asset's price during the lookback period
S^*	Represents S_{min} for call options and S_{max} for put options
λ	Coefficient of partial payoff
$V(t)$	Variance process of the Heston model
$W(t)$	Wiener process (Brownian motion)
μ	Drift coefficient of the Wiener process
θ_d	Theta coefficient
$\mathcal{A}$	Infinitesimal generator
x	Logarithmic ratio of S^* to $S(t)$
y	Variance of the Heston model after transformation
v	Instantaneous variance
σ	Volatility of the instantaneous variance
κ	Mean reversion speed of the variance
ρ	Correlation of Wiener processes
θ_m	Mean reversion level of the variance
N	Number of grid points
M	Number of time steps

1. Introduction

Economists assume efficient financial markets with fair price settings through supply and demand. Therefore, market prices should reflect the fair price, and investors would be able to value financial assets accurately. While this hypothesis holds for most market events, anomalies and distortions in prices aggravate the evaluation of fair prices for market participants. The value of certain financial assets, such as futures contracts, can be directly determined through simple calculations or the usage of formulas. On the other hand, many financial assets, such as stocks or bonds, are dependent on uncertain variables such as management, earnings, default probability, and other economic factors. Similarly, the price of options is reliant on the upon-agreed conditions and unknown future prices of the underlying.

Lookback options are barrier-determined options which pay the difference between the price at maturity and the most beneficial price of the underlying during the term of the option. This characteristic increases the likelihood and the amount of a payoff for an investor but complicates the evaluation of fair prices. Thus, the payoff is dependent on the future movement of the underlying asset's price. The known analytical solution for lookback options in the Heston model is given in the form of

$$\mathbf{w}(t) = e^{-\mathbf{A}t}\mathbf{g}$$

with the matrix $\mathbf{A}$ in the exponent of e. Thus, the option price can be defined with a unique solution, but the calculation for a simple case with few steps already requires extensive processing power, and the complex process is far too time-consuming for this purpose. Other closed-end formulas likewise fall short in the valuation of barrier-determined options because of inaccuracy or additional assumptions. Therefore, the finite difference method is used to approximate the prices of lookback options efficiently. Consequently, the thesis of this paper is the evaluation of lookback options in the Heston model using the finite difference method in Python. The prospective model aims to generate an accurate price estimate up to the second decimal place by using the calibrated parameters of a lookback option as inputs. Moreover, the aim of the model is to evaluate prices expeditiously for floating and fixed lookback call and put options as well as partial lookback options. In addition, this thesis strives to present reliable option prices for a scientific paper with erroneous values.

The focus of the thesis is on the pricing of European lookback options, the Heston model and its underlying model, and the finite difference method. The model is designed to use continuous monitoring and is applicable to lookback options with all types of underlying. The prices of fixed lookback options are determined based on a put-call parity using prices of floating lookback options. The intention is to create a transparent model that contributes to the evaluation of lookback option prices and strengthens research in this field.

This thesis provides an explanation for the functionality of financial options in general and for lookback options specifically in the second chapter. The third chapter introduces option pricing models, including the Black-Scholes model and elucidates its shortcomings. Furthermore, the Black-Scholes model is extended to the Heston model. Chapter four expounds on the finite difference method, which will be used for the discretisation of differential equations in two dimensions. Moreover, the grid and matrices for the approximation as well as the boundary conditions are introduced and explained. Chapter five deduces the applicable formulas, performs necessary transformations, and depicts the implementation of the model in Python. Additionally, the Craig-Sneyd scheme is presented and applied, and a graphical user interface is created. Subsequently, the developed model is verified by a Monte Carlo simulation with calibrated parameters, and the results are compared in chapter six. Furthermore, this chapter provides amended option prices and serves as a correction for the article of De Gennaro Aquino & Bernard (2019, p. 738). Finally, chapter seven concludes the thesis and discusses the decisive findings and limitations.

2. Financial Options and Lookback Options

An option is a financial contract with a counterparty which gives the buyer the right to buy or sell an underlying asset S at the strike price K before or at a specified expiration date. The option contract is only binding for the emitter of the option in case of delivery of the underlying, called execution of an option. The option's premium compensates the emitter for incurring the inherent risk of a loss. A European option concedes the buyer the right to execute the option at the expiration date T. In contrast, the buyer of an American option is allowed to execute the option at any time during the specified term (Bodie, Kane & Marcus, 2019, p. 475).

2.1. Call & Put Options

The buyer of a call option speculates on a rising price of the underlying, whereas the buyer of a put option speculates on a decreasing price. The payoff function of an option is given by the function g through

$$g\big(S(T)\big) = \max(\omega(S(T) - K), 0 \tag{2.1}$$

with $\omega = 1$ for a call option and $\omega = -1$ for a put option (Hilber, 2023, p. 3). The option expires worthless if $S(t)$ is below K in the case of a call option or $S(t)$ is above K for a put option. The execution of the option would be unfavourable in this case because the underlying could be acquired on the market for a lower price or sold on the market for a higher price. Hence, the execution would lead to a loss for the option holder. The option's premium consists of the intrinsic value, which is determined by the payoff plus the time value, corresponding to the likelihood of an option to generate a higher payoff.

2.2. Lookback Options

Lookback options are path-dependent options where the payoff depends on the price development of the underlying over a specified term, called the lookback period. The payoff of lookback options depends on the reached minimum S_{min} in the case of lookback call options and the maximum S_{max} in the case of lookback put options. Lookback options are divided into floating lookback options without a strike K and fixed lookback options with a strike K.

The payoff of a floating lookback option with S^* representing S_{min} or S_{max} is given by

$$V_{fl} = \max(\omega(S(T) - S^*), 0) \tag{2.2}$$

with $\omega = 1$ for a floating lookback call option C_{fl} and $\omega = -1$ for a floating lookback put option P_{fl}. The payoff of a fixed lookback call option C_{fix} with $\omega = 1$ and fixed lookback put option P_{fix} with $\omega = -1$ are given by

$$V_{fix} = \max(\omega(S(T)_{max} - K), 0). \tag{2.3}$$

The attractive characteristic of lookback options to take the most beneficial price enables the buyer to realise a higher or at least identical payoff in comparison to conventional options and to speculate on price fluctuations of the underlying asset (Leung, 2013, p. 145). The downside of the lookback option is the significantly higher premium in comparison to a conventional option because of a higher payoff and the elevated inherent risk for the issuer.

In addition, the price of a lookback option can be reduced by reducing the payoff partially by the parameter λ. A partial lookback option is generally a floating lookback option with a scaled payoff. The parameter λ typically takes the value $\lambda \geq 1$ for call options and $0 < \lambda \leq 1$ for put options. A partial lookback option with $\lambda = 1$ is identical to a conventional floating lookback option. Thus, the payoff is given by the formula

$$V_{fl} = \max(\omega(S(T) - \lambda S^*), 0), \tag{2.4}$$

which reduces the payoff by decreasing the difference between $S(T)$ and the minimum or maximum S^* (Hilber, 2023, p. 366).

3. Option Pricing Models

The value of an option can be derived from option pricing models through the application of the theory of probability and stochastic processes. First, the Black-Scholes equation is derived by a partial differential equation (PDE), and the Black-Scholes formula is given afterwards. Subsequently, the shortcomings of the Black-Scholes model are identified, and the Heston model is presented to address these deficiencies.

3.1. Black-Scholes Model

The Black-Scholes model is the established standard option pricing model for the evaluation of European call and put options. Black and Scholes (1973, p. 640-641) assume that (a) the short-term interest rate r is known and constant, (b) the price of the underlying follows a continuous random walk and has a log-normal distribution, (c) the underlying does not pay a dividend, (d) there are no transaction costs for trading the underlying, (e) any fraction of the price of the underlying can be borrowed at the short-term interest rate, and (f) there are no limitations to short selling. Hence, the option value depends only on the price of the underlying, time, and known variables and assumes a no-arbitrage price for European call and put options.

Let $V(s,t)$ denote the price of a European option at time t and let Π_t denote the portfolio value at time t, which contains one option $V(s,t)$ and a short position consisting of a number δ of the underlying asset multiplied by the asset price $s = S(t)$.

$$\Pi_t = V(s,t) - \delta s \tag{3.1}$$

The change in portfolio value can be found by the derivative of (3.1), given by

$$\mathrm{d}\Pi_t = \mathrm{d}V(s,t) - \delta \mathrm{d}s \tag{3.2}$$

with $\mathrm{d}V(s,t)$ being the change in the option's value. The change in the underlying price, denoted $\mathrm{d}s$, is given by

$$\mathrm{d}s = \mu s \mathrm{d}t + \sigma s \mathrm{d}W(t) \tag{3.3}$$

with $\mu s \mathrm{d}t$ describing the drift of the underlying as a deterministic component and $\sigma s \mathrm{d}W(t)$ describing the volatility. The term $\mathrm{d}W(t)$ denotes the geometric Brownian motion, also known as the Wiener process. Given that $W(t)$ is normally distributed, the

equation (3.3) is a special case of the Itô process and is log-normal distributed (Albrecher, Binder & Mayer, 2009, p. 49). The term $V(s,t)$ is the function of a stochastic process. Therefore, the dynamics of $\mathrm{d}V(s,t)$ are described by Itô's Lemma

$$\mathrm{d}V = \partial_t V * \mathrm{d}t + \partial_s V * \mathrm{d}s + \frac{1}{2}\partial_{ss} V * \mathrm{d}s^2. \tag{3.4}$$

By substituting the $\mathrm{d}s^2$ term of (3.3) in (3.4), most terms cancel out because of

$$\mathrm{d}t * \mathrm{d}t = 0, \qquad \mathrm{d}t * \mathrm{d}W = 0, \qquad \mathrm{d}W * \mathrm{d}W = \mathrm{d}t.$$

The modified equation can be written as

$$\mathrm{d}V = \partial_t V * \mathrm{d}t + \partial_s V * \mathrm{d}s + \frac{1}{2}\sigma^2 s^2 \partial_{ss} V \mathrm{d}t. \tag{3.5}$$

The equation (3.5) can be inserted into (3.2) for $\mathrm{d}V$ and rearranged into $\mathrm{d}t$ terms and $\mathrm{d}s$ terms to receive

$$\mathrm{d}\Pi_t = \left(\partial_t V + \frac{1}{2}\sigma^2 s^2 \partial_{ss} V\right)\mathrm{d}t + (\partial_s V - \delta)\mathrm{d}s.$$

By choosing $\delta = \partial_s V$ for delta, the stochastic component $(\partial_s V - \delta)\mathrm{d}s$ cancels out, and the equation simplifies to

$$\mathrm{d}\Pi_t = \left(\partial_t V + \frac{1}{2}\sigma^2 s^2 \partial_{ss} V\right)\mathrm{d}t. \tag{3.6}$$

This step eliminates the risk, and the resulting portfolio is deterministic. Under the Q-measure, the portfolio is risk-free and should yield the risk-free interest rate r. The new portfolio is given by

$$\mathrm{d}\Pi_t = r * \Pi \mathrm{d}t = (rV - rs\partial_s V)\mathrm{d}t. \tag{3.7}$$

The equations (3.6) and (3.7) are set equal

$$\left(\partial_t V + \frac{1}{2}\sigma^2 s^2 \partial_{ss} V\right)\mathrm{d}t = (rV - rs\partial_s V)\mathrm{d}t,$$

and the terms are rearranged on the left side to consolidate the equation.

The result is complemented with the payoff function to yield in the system of equations

$$\begin{cases} \partial_t V + \frac{1}{2}\sigma^2 s^2 \partial_{ss} V + rs\partial_s V - rV = 0 & \text{in } \mathbb{R}^+ \times [0, T[\\ V(s,T) = g(s) & \text{in } \mathbb{R}^+ \end{cases}, \tag{3.8}$$

which is the Black-Scholes equation (Albrecher et al., 2009, p. 53-55).

The Black-Scholes equation (3.8) is a PDE and is satisfied for European derivatives. For specific derivates to be calculated, the desired derivative's end and boundary condition must first be defined. In the case of the European call and put option, the end condition is given by the payoff function and the boundary condition is given by the no-arbitrage limitations. It becomes apparent that the equation is akin to the heat equation, and the Feynman-Kac theorem can be used to solve the equation analytically (Albrecher et al., 2009, p. 54). The result is the closed-end Black-Scholes formula, which can be extended with the addition of Hilber (2023, p. 8) to include the continuous dividend yield q. This approach replaces μ with $(r - q)$ under a risk-neutral valuation with constant parameters r, q, and σ. The extended Black-Scholes formula is given by

$$\begin{cases} V(s,t,T,K,\sigma,r,q,\omega) = \omega\left(se^{-q(T-t)}\Phi_{0,1}(\omega d_1) - Ke^{-r(T-t)}\Phi_{0,1}(\omega d_2)\right) \\ d_1 = \frac{1}{\sigma\sqrt{T-t}}\left(\ln\left(\frac{s}{K}\right) + \left(r - q + \frac{\sigma^2}{2}\right)(T-t)\right) \\ d_2 = d_1 - \sigma\sqrt{T-t} \end{cases} \tag{3.9}$$

with $\Phi_{0,1}$ being the normal distribution function (Hilber, 2023, p. 8-9). The probability density function of the normal distribution is given by

$$\varphi_{\mu,\sigma}(x) := \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

In the following intercept, the non-constant coefficients of the Black-Scholes equation are converted into constant coefficients to facilitate the calculation of option prices. First, a transition from time t to the term of maturity of the option, denoted $\tau = T - t$, is performed by defining the function

$$v(s,\tau) := V(s, T - \tau), \tag{3.10}$$

which replaces $V(s,t)$ in the Black-Scholes equation (3.8).

The risk-neutral valuation, as shown in (3.9), is implemented with $(r - q)$ replacing μ. Applying the chain rule $\partial_\tau v = -\partial_t V$ changes the signs in the equation and yields in

$$\begin{cases} \partial_\tau v - \frac{1}{2}\sigma^2 s^2 \partial_{ss} v - (r-q) s \partial_s v + rv = 0 & \text{in } \mathbb{R}^+ \times]0, T] \\ v(s,0) = g(s) & \text{in } \mathbb{R}^+ \end{cases}. \tag{3.11}$$

The transformation of the differential equation is achieved by setting the variable $x := \ln(s)$ and replacing $s = e^x$ to result in the modified function

$$v(s,\tau) = v(e^x, \tau).$$

The new function u with the natural logarithm x replacing the underlying value s is defined for

$$u(x,\tau) := v(e^x, \tau),$$

and the new equation exhibiting constant coefficients is given by

$$\begin{cases} \partial_\tau u - \frac{1}{2}\sigma^2 \partial_{xx} u - \left(r - q - \frac{\sigma^2}{2}\right) \partial_x u + ru = 0 & \text{in } \mathbb{R} \times]0, T] \\ u(x,0) = g(e^x) & \text{in } \mathbb{R} \end{cases}.$$

For reasons of simplicity, the term of maturity τ is renamed to t, and the equation is simplified by defining the coefficients a, b, and c with

$$a := -\frac{1}{2}\sigma^2, \qquad b := -r + q + \frac{\sigma^2}{2}, \qquad c := r.$$

The remodelled Black-Scholes equation with option price $u = u(x,t)$ is defined as

$$\begin{cases} \partial_t u + a\partial_{xx} u + b\partial_x u + cu = 0 & \text{in } \mathbb{R} \times]0, T] \\ u(x,0) = g(e^x) & \text{in } \mathbb{R} \end{cases} \tag{3.12}$$

and builds the basis for further calculations (Hilber, 2023, p. 82-83).

3.2. Deficiencies of the Black-Scholes Model

The Black-Scholes model calculates European call and put option values under the assumption of normally distributed logarithmic returns. On the other hand, empirical log returns do not exhibit a normal distribution. The logarithmic daily returns of the Swiss Market Index (SMI) from 1. January 2013 to 31. December 2022 were retrieved from Refinitiv (2023) and used as an example to illustrate the deviation.

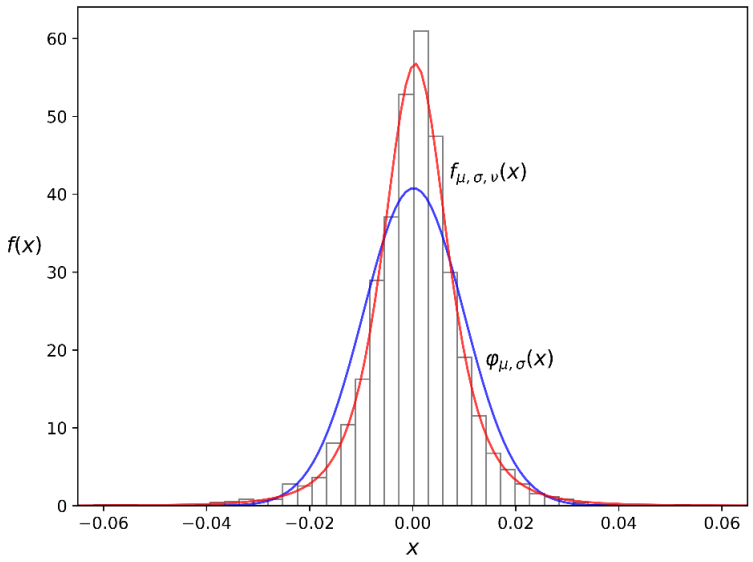

Figure 3.1: Histogram of logarithmic daily SMI returns with its normal distribution and the Student's t-distribution in reference to Hilber (2023, p. 12).

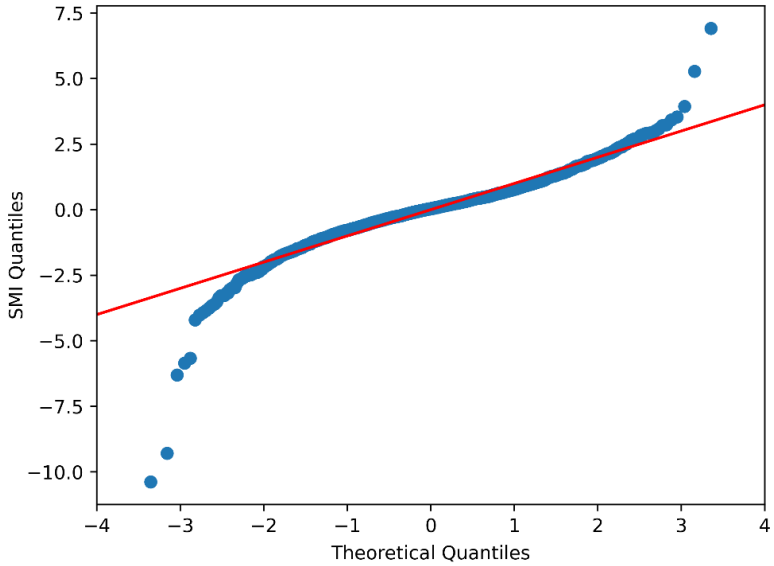

Figure 3.2: QQ-plot of logarithmic daily SMI returns illustrates high kurtosis and leptokurtic distribution.

The normal distribution has a mean $\mu = 0$, a standard deviation $\sigma = 1$, a skewness of 0 and excess kurtosis of 0. In contrast, the empirical log returns have $\mu \approx \bar{r} \doteq 0.0001689$, $\sigma \approx s_r \doteq 0.0097648$, a skewness of -0.9985240, and an excess kurtosis of 7.8901749. The SMI sample exhibits a negative skewness and a significant excess kurtosis as a leptokurtic distribution, meaning that the Black-Scholes model significantly underapproximates the probability of extreme returns.

The Student's t-distribution is more leptokurtic and exhibits fatter tails than the normal distribution. Thus, the Student's t-distribution approximates the distribution of the SMI better (Hilber, 2023, p. 12). The probability density function of the Student's t-distribution is given by

$$f_{\mu,\sigma,v}(x) = c_v \frac{1}{\sigma\sqrt{v\pi}}\left(1 + \frac{1}{v}\left(\frac{x-\mu}{\sigma}\right)^2\right)^{-\frac{v+1}{2}}.$$

A further deficiency of the Black-Scholes model is the assumption of constant volatility with respect to the term of maturity and the strike K. Every parameter for the Black-Scholes formula is determined by either the option contract itself or no-arbitrage considerations except the volatility as an arbitrary parameter. The volatility can be estimated from data of options published by exchanges such as the EUREX or financial databases such as Refinitiv or Bloomberg but is always assumed to be constant. Conversely, the published option price can be used to calculate the volatility as the only missing parameter. In this case, the calculated probability is referred to as implied volatility σ^i.

The constancy of volatility is subsequently tested with published prices of European call options on the DAX in EUR with an expiry date 23. June 2023 and term of maturity $\tau = T - t = \frac{112}{360}$. The market values V_I^M of the European call options are given for selected strike prices K_I in index points. The DAX closed at 15'305.02, and the data is drawn from EUREX with reference date 1. March 2023. Moreover, the applied risk-free rate is given by the EURIBOR money market rate quoted at 2.783%. The EURIBOR is the decisive reference interest rate for unsecured funds in the euro interbank market. Hence, the option values for selected strikes are listed below in Table 3.1, and the implied volatility is plotted in Figure 3.3.

K_I	V_I^M	K_I	V_I^M	K_I	V_I^M	K_I	V_I^M
11’600	3’878.2	13’600	2’002.3	15’600	492.1	17’600	17.9
12’000	3’492.6	14’000	1’653.3	16’000	301.9	18’000	8.0
12’400	3’110.8	14’400	1’321.3	16’400	167.2	18’500	3.0
12’800	2’733.9	14’800	1’011.9	16’800	83.6	19’000	1.2
13’200	2’363.7	15’200	732.3	17’200	39.5	19’500	0.5

Table 3.1: Market value V_I^M of European call options on the DAX with selected strike price K_I in index points and expiry date on 23. June 2023 (EUREX, 2023).

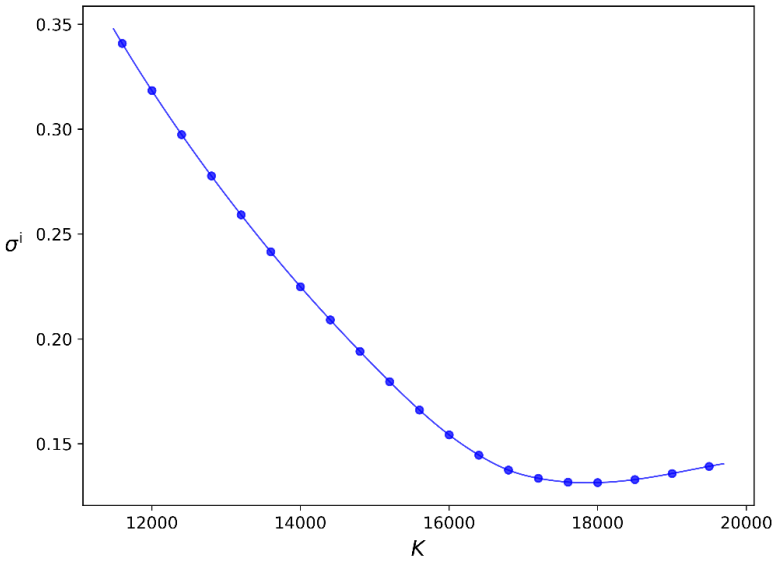

Figure 3.3: Visualisation of the implied volatility σ^i for different strikes of European call options in reference to Hilber (2023, S. 19).

The assumed implied volatility in the Black-Scholes model should be identical for all strikes and displayed as a horizontal line. However, the graph displays a downwards-sloping curve, the so-called volatility smile. Therefore, it infers that the implied volatility in the Black-Scholes formula is changing for varying strikes, consequently not being constant. These shortcomings can be addressed by extending the Black-Scholes model to the Heston model. This model reproduces the volatility as a stochastic process, accounting for the volatility smile observed in the European call options. Thus, the actual volatility is more accurately estimated, the additional stochastic process affects the distribution of the underlying’s price, and the model yields a more precise price estimation of the fair value of an option.

3.3. Heston Model

The Heston model extends the Black-Scholes model by replacing the constant volatility σ in (3.3) with a stochastic volatility $\sigma(t)$. The adapted stochastic differential equation is thereby given through

$$\mathrm{d}S(t) = \mu S(t)\mathrm{d}t + \sigma(t)S(t)\mathrm{d}W_1(t)$$

with the stochastic volatility defined by

$$\sigma(t) = \sqrt{V(t)}.$$

The process $V(t)$ represents the variance in the Heston model (Hilber, 2023, p. 377-378) and the stochastic differential equation of $V(t)$ is described by

$$\mathrm{d}V(t) = \kappa\big(\theta - V(t)\big)\mathrm{d}t + \delta\sqrt{V(t)}\mathrm{d}W_2(t) \tag{3.13}$$

with κ as the mean reversion speed of the variance, θ the mean reversion level of the variance, $\delta > 0$ as the volatility of the variance, and $W_2(t)$ the Wiener process of the variance $V(t)$. The Wiener processes $W_1(t)$ and $W_2(t)$ are assumed to be correlated

$$E^{\mathbb{P}}[\mathrm{d}W_1(t)\mathrm{d}W_2(t)] = \rho\mathrm{d}t$$

with correlation coefficient $\rho \in [-1,1]$. For evaluating the prices of financial assets, the risk-neutral Q-measure is subsequently applied instead of the historical P-measure (Rouah, 2015, p. 2). Girsanov's theorem is applied to the stochastic differential equations, and the risk-neutral equations are states as

$$\begin{aligned} \mathrm{d}S(t) &= (r-q)S(t)\mathrm{d}t + \sqrt{V(t)}S(t)\mathrm{d}W_1(t), \\ \mathrm{d}V(t) &= \kappa\big(\theta - V(t)\big)\mathrm{d}t + \delta\sqrt{V(t)}\mathrm{d}W_2(t) \end{aligned} \tag{3.14}$$

with μ being replaced by $(r-q)$ and the correlation of the Wiener processes given by

$$\begin{aligned} E^{\mathbb{Q}}[\mathrm{d}W_1(t)\mathrm{d}W_2(t)] &= \rho\mathrm{d}t, \\ W_2(t) &= \rho W_1(t) + \sqrt{1-\rho^2}W_2(t). \end{aligned}$$

The variance process $V(t)$ follows a stochastic differential equation, also known as the Cox-Ingersoll-Ross (CIR) model, which initially depicts the movements of interest rates.

A property of the variance process $V(t)$ is the Feller condition. It states that the drift is significant enough to ensure the variance process remains positive and does not reach zero if the condition $2\kappa\theta > \delta^2$ is satisfied (Rouah, 2013, p. 2-4).

The differential equations of the Heston model can be simplified under the risk-neutral Q-measure. Therefore, the vector process $\mathbf{X}(t) = \big(S(t), V(t)\big)^{\mathsf{T}}$ is defined by

$$\begin{pmatrix} \mathrm{d}S(t) \\ \mathrm{d}V(t) \end{pmatrix} = \underbrace{\begin{pmatrix} (r-q)S(t) \\ \kappa\big(\theta - V(t)\big) \end{pmatrix}}_{\boldsymbol{\mu}(\mathbf{X}(t))} \mathrm{d}t + \underbrace{\begin{pmatrix} \sqrt{V(t)}S(t) & 0 \\ \rho\delta\sqrt{V(t)} & \sqrt{1-\rho^2}\delta\sqrt{V(t)} \end{pmatrix}}_{\boldsymbol{\sigma}(\mathbf{X}(t))} \begin{pmatrix} \mathrm{d}W_1(t) \\ \mathrm{d}W_2(t) \end{pmatrix}$$

with the covariance matrix $\mathbf{Q}$ for $\big(\mathbf{x} = (s, v)\big)$

$$\mathbf{Q}(\mathbf{s}) = \boldsymbol{\sigma}(\mathbf{s})\boldsymbol{\sigma}(\mathbf{s})^{\mathsf{T}} = \begin{pmatrix} s^2 v & \rho\delta s v \\ \rho\delta s v & \delta^2 v \end{pmatrix}.$$

Thus, the Heston model can be written with

$$\mathbf{X}(t) = \begin{pmatrix} \mathrm{d}S(t) \\ \mathrm{d}V(t) \end{pmatrix}, \qquad \boldsymbol{\mu}(\mathbf{x}, t) = \begin{pmatrix} (r-q)S(t) \\ \kappa\big(\theta - V(t)\big) \end{pmatrix},$$

$$\boldsymbol{\sigma}(\mathbf{x}, t) = \begin{pmatrix} \sqrt{V(t)}S(t) & 0 \\ \rho\delta\sqrt{V(t)} & \sqrt{1-\rho^2}\delta\sqrt{V(t)} \end{pmatrix}, \qquad \mathbf{W}(t) = \begin{pmatrix} \mathrm{d}W_1(t) \\ \mathrm{d}W_2(t) \end{pmatrix}$$

as the differential equation

$$\mathrm{d}\mathbf{X}(t) = \boldsymbol{\mu}(\mathbf{X}(t), t)\mathrm{d}t + \boldsymbol{\sigma}(\mathbf{X}(t), t)\mathrm{d}\mathbf{W}(t).$$

Consequently, the multidimensional case does not differ from the one-dimensional case in form (Hilber, 2023, p. 326-327, 378-379).

4. Finite Difference Method

This chapter explains the finite difference method as an approach to numerically solve PDEs. First, the approximation of finite differences is explained for one-dimensional differential equations, including extended formulas with smaller estimating errors. Secondly, the finite difference grid and the fundamental matrices are defined. The theta method is introduced, and multiple combinable boundary conditions are given. Subsequently, the case is extended to PDEs with two dimensions.

4.1. Finite Differences in One Dimension

This section introduces the finite difference method (FDM) and the corresponding grid for one dimension. Afterwards, the finite difference equations are transformed into matrix form and the theta method as well as boundary conditions are presented and applied.

4.1.1. Finite Difference Approximation

The slope of the secant at a given point of a threefold continuously differentiable function $f(x)$ can be determined by the limit of

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

to obtain the derivative $f'(x)$ of a function. This approach can be used to approximate the value of the derivative for small Δx and is the basis for the finite difference method.

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

The Taylor expansion of $f(x)$ is given by

$$f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{1}{2!} f''(x)\Delta x^2 + \frac{1}{3!} f'''(x)\Delta x^3 + \cdots \qquad (4.15)$$

which can be rewritten to the derivative of $f(x)$

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} + \frac{1}{2!} f''(x)\Delta x + \frac{1}{3!} f'''(x)\Delta x^2 + \cdots.$$

The derivation between the first-degree approximation and the actual value is described by the error term $\mathcal{O}(\Delta x)$, which represents the subsequent terms. The quotient obtained

in (4.16) is called the forward difference, which follows the explicit Euler's method. Explicit methods determine the value of the next increment based on the initial or current value. In this case, the approximation $f(x + \Delta x)$ is calculated based on $f(x)$ without requiring the solution of algebraic equations. Thus, this method enables a simple and fast calculation of the following value but exhibits only conditional stability.

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} + \mathcal{O}(\Delta x) \tag{4.16}$$

The error term $\mathcal{O}(\Delta x)$ in (4.16) can be omitted to receive the approximation

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}. \tag{4.17}$$

Alternatively, the Taylor expansion can be computed at the point $(x - \Delta x)$, called the backward difference, which follows the implicit Euler's method. In contrast to the explicit method, the implicit method finds the solution of the approximation by solving an equation involving the current and advanced value. The disadvantage of using this method is the higher computation time required to solve the equation. Nonetheless, the benefits of this approach include enhanced convergence, improved numerical stability, and the ability to use larger increments Δx when solving equations. The Taylor expansion (4.15) for $(x - \Delta x)$ is given by

$$f(x - \Delta x) = f(x) - f'(x)\Delta x + \frac{1}{2!}f''(x)\Delta x^2 - \frac{1}{3!}f'''(x)\Delta x^3 + \cdots \tag{4.18}$$

with the derivative adjusting to

$$f'(x) = \frac{f(x) - f(x - \Delta x)}{\Delta x} - \frac{1}{2!}f''(x)\Delta x + \frac{1}{3!}f'''(x)\Delta x^2 + \cdots.$$

The polynomial can be simplified to the backward difference quotient

$$f'(x) = \frac{f(x) - f(x - \Delta x)}{\Delta x} + \mathcal{O}(\Delta x) \tag{4.19}$$

(Hilber, 2023, p. 99-102), which is first-order accurate $\mathcal{O}(\Delta x)$ and unconditionally stable.

Omitting the error term $\mathcal{O}(\Delta x)$ in (4.19) leads to the approximation

$$f'(x) \approx \frac{f(x) - f(x - \Delta x)}{\Delta x}. \tag{4.20}$$

The forward and backward difference have some variation and are not second-order accurate, while only the backward difference is unconditionally stable. To address this problem, the Crank-Nicolson method can be used to obtain a more precise result. The method is implemented by subtracting polynomial (4.18) from (4.15) to obtain

$$f(x + \Delta x) - f(x - \Delta x) = 2f'(x)\Delta x + 2\frac{1}{3!}f'''(x)\Delta x^3 + \cdots. \tag{4.21}$$

The polynomial (4.21) can be rewritten as the derivative

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} - \frac{1}{3!}f'''(x)\Delta x^2 + \cdots,$$

adjusted to result in the central difference quotient

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} + \mathcal{O}(\Delta x^2),$$

and the approximation with omitted error term $\mathcal{O}(\Delta x^2)$ is defined as

$$f'(x) \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x}. \tag{4.22}$$

Thus, the central difference results in a more minute result than the forward or backward difference individually. The accuracy of the first-order central difference quotient can be further improved by taking the second-order central difference.

The quotient can be derived by adding (4.18) to (4.15) as subsequently shown by

$$f(x + \Delta x) + f(x - \Delta x) = 2f(x) + 2\frac{1}{2!}f''(x)\Delta x^2 + 2\frac{1}{4!}f^{(4)}(x)\Delta x^4 + \cdots$$

and rewriting to

$$f''(x) = \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{\Delta x^2} - 2\frac{1}{4!}f^{(4)}(x)\Delta x^2 + \cdots. \tag{4.23}$$

The polynomial (4.23) can be simplified to the central difference quotient

$$f''(x) = \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2)$$

which is second-order accurate $\mathcal{O}(\Delta x^2)$ and unconditionally stable (Seydel, 2017, p. 134-146). The omission of the error term $\mathcal{O}(\Delta x^2)$ leads to the approximation

$$f''(x) \approx \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2}. \tag{4.24}$$

4.1.2. Finite Difference Grid and Matrices

The quotients derived above can be applied on an interval $G_x =]x_{min}, x_{max}[$ at defined steps to approximate the values at these points. Thus, the equidistant finite difference grid

$$G_x := \{x_i \mid i = 1, \dots, N\}$$

is defined with minimum value $x_{min} = x_1$ and maximum value $x_{max} = x_N$. Let Δx denote the discretisation increments of variables x with

$$x_i = x_{min} + i\Delta x, \qquad \Delta x := \frac{x_{max} - x_{min}}{N-1}.$$

The application of the finite difference method is demonstrated in a generalised example. The differential equation

$$\begin{cases} au''(x) + bu'(x) + cu(x) = f(x) & \text{in } G_x \\ u(x_{min}) = u_{min} \\ u(x_{max}) = u_{max} \end{cases}$$

approximates the function $u(x)$ (Hilber, 2023, p. 110-114) in a one-dimensional equidistant grid for every point x_i

$$x_{min} = x_1 < x_2 < \cdots < x_{i-1} < x_i < x_{i+1} < \cdots < x_{N-1} < x_N = x_{max}.$$

The first-order central difference quotient in (4.22) can be rewritten with the grid notation and by omitting the error term $\mathcal{O}(\Delta x^2)$ to receive the approximation for a grid point x_i specified as

$$u'(x_i) \approx \partial_{\Delta x} u(x_i) = \frac{u(x_{i+1}) - u(x_{i-1})}{2\Delta x} \tag{4.25}$$

and rewriting the second-order central difference quotient in (4.24) results in

$$u''(x_i) \approx \partial^2_{\Delta x} u(x_i) = \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{\Delta x^2}. \tag{4.26}$$

The derivatives $u''(x_i)$ and $u'(x_i)$ are replaced by quotient (4.25) and (4.26) at every x_i to receive

$$a\frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{\Delta x^2} + b\frac{u(x_{i+1}) - u(x_{i-1})}{2\Delta x} + cu(x_i) \approx f(x_i).$$

To enforce equality in the above approximations, the function values $u(x_{i+1}), u(x_i)$, and $u(x_{i-1})$ are replaced by the approximative values u_{i+1}, u_i, and u_{i-1} to enable the calculation of the equation. Hence, the revised equation is given by

$$a\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta u^2} + b\frac{u_{i+1} - u_{i-1}}{2\Delta u} + cu_i = f(x_i) \tag{4.27}$$

with $i = 2, \dots, N-1$. Thus, the equation for the points $u_2, \dots, u_{N-1}$ are given by

$$\begin{aligned}
&\left(-\frac{2a}{\Delta x^2} + c\right)u_2 + \left(\frac{a}{\Delta x^2} + \frac{b}{2\Delta x}\right)u_3 = f(x_2) - \left(\frac{a}{\Delta x^2} - \frac{b}{2\Delta x}\right)u_{min},\\
&\left(\frac{a}{\Delta x^2} - \frac{b}{2\Delta x}\right)u_{i-1} + \left(-\frac{2a}{\Delta x^2} + c\right)u_i + \left(\frac{a}{\Delta x^2} + \frac{b}{2\Delta x}\right)u_{i+1} = f(x_i),\\
&\left(\frac{a}{\Delta x^2} - \frac{b}{2\Delta x}\right)u_{N-2} + \left(-\frac{2a}{\Delta x^2} + c\right)u_{N-1}\\
&\qquad = f(x_{N-1}) - \left(\frac{a}{\Delta x^2} - \frac{b}{2\Delta x}\right)u_{max}
\end{aligned} \tag{4.28}$$

with the upper and lower equation for the boundary points and the middle equation for points $u = 2, \dots, N-2$.

The equations possess the same terms, which can be defined as variables

$$\alpha := \frac{a}{\Delta x^2} - \frac{b}{2\Delta x}, \qquad \beta := -\frac{2a}{\Delta x^2} + c, \qquad \gamma := \frac{a}{\Delta x^2} + \frac{b}{2\Delta x}. \tag{4.29}$$

Thus, the equations can be written in a matrix with the defined variables as

$$\begin{array}{ccccccccccc}
\beta u_2 & + & \gamma u_3 & & & & & & & = & f(x_2) - \alpha u_{min}\\
\alpha u_2 & + & \beta u_3 & + & \gamma u_4 & & & & & = & f(x_3)\\
 & & \alpha u_3 & + & \beta u_4 & + & \gamma u_5 & & & = & f(x_4)\\
 & & & & \ddots & & & & & \vdots & \\
 & & & & \alpha u_{N-3} & + & \beta u_{N-2} & + & \gamma u_{N-1} & = & f(x_{N-2})\\
 & & & & & + & \alpha u_{N-2} & + & \beta u_{N-1} & = & f(x_{N-1}) - \gamma u_{max}
\end{array}.$$

Hence, the equations are rewritten as $(N-2)\times(N-2)$ matrix $\mathbf{A}$, the vector of the points $\mathbf{u}$ and the vector of the function values $\mathbf{f}$

$$\mathbf{A} := \begin{pmatrix} \beta & \gamma & & & & \\ \alpha & \beta & \gamma & & & \\ & \alpha & \beta & \gamma & & \\ & & & \ddots & & \\ & & & \alpha & \beta & \gamma \\ & & & & \alpha & \beta \end{pmatrix}, \qquad \mathbf{u} := \begin{pmatrix} u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{pmatrix}, \tag{4.30}$$

$$\mathbf{f} := \begin{pmatrix} f(x_2) - \alpha u_{min} \\ f(x_3) \\ \vdots \\ f(x_{N-2}) \\ f(x_{N-1}) - \gamma u_{max} \end{pmatrix}$$

to represent the equation in matrix notation

$$\mathbf{Au} = \mathbf{f}.$$

To apply the derivation in later problems, matrix $\mathbf{A}$ can be written as a sum of three matrices. These matrices are created by the discretisations of the terms au'', bu', and cu.

First, assume that $b = c = 0$ and the differential equation is stated as

$$au''(x) = f(x)$$

with $\alpha = \frac{a}{\Delta x^2}$, $\beta = -\frac{2a}{\Delta x^2}$, and $\gamma = \frac{a}{\Delta x^2}$. The matrix $\mathbf{A}$ for this case is given by

$$\mathbf{M}_a^{(2)} := \frac{1}{\Delta x^2}\begin{pmatrix} -2a & a & & & & \\ a & -2a & a & & & \\ & a & -2a & a & & \\ & & & \ddots & & \\ & & & a & -2a & a \\ & & & & a & -2a \end{pmatrix}.$$

Secondly, assume that $a = c = 0$ and the differential equation is stated as

$$bu'(x) = f(x)$$

with $\alpha = -\frac{b}{2\Delta x}$, $\beta = 0$, and $\gamma = \frac{b}{2\Delta x}$. The matrix $\mathbf{A}$ for this case is given by

$$\mathbf{M}_b^{(1)} := \frac{1}{2\Delta x}\begin{pmatrix} 0 & b & & & & \\ -b & 0 & b & & & \\ & -b & 0 & b & & \\ & & & \ddots & & \\ & & & -b & 0 & b \\ & & & & -b & 0 \end{pmatrix}.$$

Thirdly, assume that $a = b = 0$ and the differential equation is stated as

$$cu(x) = f(x)$$

with $\alpha = 0$, $\beta = c$, and $\gamma = 0$. The matrix **A** for this case is given by

$$\mathbf{M}_c^{(0)} := \begin{pmatrix} c & 0 & & & & \\ 0 & c & 0 & & & \\ & 0 & c & 0 & & \\ & & & \ddots & & \\ & & & 0 & c & 0 \\ & & & & 0 & c \end{pmatrix}.$$

Consequently, matrix **A** is the sum of

$$\mathbf{A} = \mathbf{M}_a^{(2)} + \mathbf{M}_b^{(1)} + \mathbf{M}_c^{(0)}.$$

The general definition of this structure is given for function y as $(N-2) \times (N-2)$ matrix $\mathbf{M}_y^{(k)}$ in interval G with length $\overline{G}$ and $k = 0, 1, 2$, and $y_i := y(x_i)$ with $N-2$ equidistant points x_i and $\Delta x = \frac{\overline{G}}{N-1}$. The superscript (k) determines the order of the derivative ($k = 2$ for the second, $k = 1$ for the first, and $k = 0$ for the zeroth derivative). The general form with variable y is given by

$$\mathbf{M}_a^{(2)} := \frac{1}{\Delta x^2} \begin{pmatrix} -2y_2 & y_2 & & & & & \\ y_3 & -2y_3 & y_3 & & & & \\ & y_4 & -2y_4 & y_4 & & & \\ & & & \ddots & & & \\ & & & y_{N-2} & -2y_{N-2} & y_{N-2} & \\ & & & & y_{N-1} & -2y_{N-1} \end{pmatrix},$$

$$\mathbf{M}_b^{(1)} := \frac{1}{2\Delta x} \begin{pmatrix} 0 & y_2 & & & & & \\ -y_3 & 0 & y_3 & & & & \\ & -y_4 & 0 & y_4 & & & \\ & & & \ddots & & & \\ & & & -y_{N-2} & 0 & y_{N-2} & \\ & & & & -y_{N-1} & 0 \end{pmatrix}, \tag{4.31}$$

$$\mathbf{M}_c^{(0)} := \begin{pmatrix} y_2 & & & & & \\ & y_3 & & & & \\ & & y_4 & & & \\ & & & \ddots & & \\ & & & & y_{N-2} & \\ & & & & & y_{N-1} \end{pmatrix},$$

and builds the foundation for later approximations (Hilber, 2023, p. 110-114).

4.1.3. Theta Method with Application

This section introduces the theta method in order to solve finite difference problems. The method is introduced on the basis of the remodelled Black-Scholes equation in (3.12). The function $u(x,t)$ is replaced with function $w(x,t)$ in $G_x =]x_{min}, x_{max}[$ and the differential equation is given by

$$\begin{cases} \partial_t w(x,t) + \mathrm{a}\partial_{xx} w(x,t) + \mathrm{b}\partial_x w(x,t) & \\ \qquad + \mathrm{c} w(x,t) = 0 & \text{in } G_x \times]0,T] \\ w(x_{min},t) = 0 & \text{in }]0,T] \\ w(x_{max},t) = 0 & \text{in }]0,T] \\ w(x,0) = g(e^x) & \text{in } G_x \end{cases} \tag{4.32}$$

Additionally, finite differences approximate the solution of a function in a finite grid with the limits given by boundary conditions. For this example, the boundaries are defined by $w_{min}(t) = w_{max}(t) = 0$ (Hilber, 2023, p. 124). The partial derivatives of equation (4.28) are replaced by the central difference quotients as in (4.27) for $x_i, i = 2, \dots, N-1$

$$\partial_t w(x_i,t) + a\frac{w(x_{i+1},t) - 2w(x_i,t) + w(x_{i-1},t)}{\Delta x^2} + b\frac{w(x_{i+1},t) - w(x_{i-1},t)}{2\Delta x} + cw(x_i,t) \approx 0.$$

The equality in the approximation is enforced by replacing the function values $w(x_{i+1},t), w(x_i,t)$, and $w(x_{i-1},t)$ by the approximative values $w_{i+1}(t), w_i(t)$, and $w_{i-1}(t)$. Thus, the new quotient is given as

$$\partial_t w_i(t) + a\frac{w_{i+1}(t) - 2w_i(t) + w_{i-1}(t)}{\Delta x^2} + b\frac{w_{i+1}(t) - w_{i-1}(t)}{2\Delta x} + cw_i(t) = 0. \tag{4.33}$$

The resulting equations of the points possess the same terms as in the time-independent variables in (4.29) and are given by

$$\alpha := \frac{a}{\Delta x^2} - \frac{b}{2\Delta x}, \qquad \beta := -\frac{2a}{\Delta x^2} + c, \qquad \gamma := \frac{a}{\Delta x^2} + \frac{b}{2\Delta x}.$$

The boundary conditions for the time t are given by

$$w(x_0,t) = w_0(t) = 0 \text{ and } w(x_N,t) = w_N(t) = 0$$

with the simplified notation $w_i'(t) = \partial_t w_i(t)$ and leads to

$$\begin{array}{rcrcrcrcrcrcl}
w_2'(t) & + & \beta w_2 & + & \gamma w_3 & & & & & & & = & 0 \\
w_3'(t) & + & \alpha w_2 & + & \beta w_3 & + & \gamma w_4 & & & & & = & 0 \\
w_4'(t) & + & & & \alpha w_3 & + & \beta w_4 & + & \gamma w_4 & & & = & 0 \\
\vdots & & & & & & \ddots & & & & & \vdots & \\
w_{N-2}'(t) & + & & & & & \alpha w_{N-3} & + & \beta w_{N-2} & + & \gamma w_{N-1} & = & 0 \\
w_{N-1}'(t) & + & & & & & & + & \alpha w_{N-2} & + & \beta w_{N-1} & = & 0
\end{array}.$$

The above system of equations can be converted into matrix notation by using the matrix **A** of (4.30) and the vectors

$$\mathbf{w}(t) := \begin{pmatrix} w_2(t) \\ w_3(t) \\ \vdots \\ w_{N-1}(t) \end{pmatrix}, \qquad \mathbf{w}'(t) := \begin{pmatrix} w_2'(t) \\ w_3'(t) \\ \vdots \\ w_{N-1}'(t) \end{pmatrix} \tag{4.34}$$

to obtain the equation

$$\mathbf{w}'(t) + \mathbf{A}\mathbf{w}(t) = \mathbf{0}. \tag{4.35}$$

The equation (4.35) must now be completed with the option's payoff function to reflect the option value. For this purpose, the initial condition $\mathbf{w}(0) = \mathbf{g}$ determines the payoff at $t = 0$, and the vector **g** contains the function values of the payoff function at the grid points. The vector of grid points is stated as

$$\mathbf{g} := \begin{pmatrix} g(e^{x_2}) \\ g(e^{x_3}) \\ \vdots \\ g(e^{x_{N-1}}) \end{pmatrix}.$$

The resulting system of equations can be solved explicitly and is defined as

$$\begin{cases} \mathbf{w}'(t) + \mathbf{A}\mathbf{w}(t) = \mathbf{0} \\ \mathbf{w}(0) = \mathbf{g} \end{cases}, \tag{4.36}$$

which forms the complex equation in the introduction (Hilber, 2023, p. 125-127).

The following intercept illustrates the theta method to derive the calculation rule. For this purpose, the equidistant grid

$$G_t := \{t_j \mid j = 1, \dots, M\}, \qquad 0 = t_1 < t_2 < \cdots < t_{M-1} < t_M = T$$

in the interval $G_t = [0, T]$ is defined.

The grid points t_j and discretisation increments of variables t are given by

$$t_j = jk, \qquad j = 1, \dots, M, \qquad \Delta t := \frac{T}{M-1}.$$

The first derivative $w'(t)$ of equation (4.36) can be approximated by the forward difference quotient from (4.17) at the points $j = 1, \dots, M-1$ by

$$\frac{w(t_j + \Delta t) - w(t_j)}{\Delta t} + Aw(t_j) \approx 0. \tag{4.37}$$

The equality in the approximation is enforced by replacing the function values $w(t_j)$ with the approximative values w_j. Thus, the equation with $w(t_j) \approx w_j$ is given by

$$\frac{w_{j+1} - w_j}{\Delta t} + Aw_j = 0. \tag{4.38}$$

Equation (4.38) can be multiplied with Δt and rearranged to

$$w_{j+1} = w_j + \underbrace{\Delta t(-A)w_j}_{\Delta w} \tag{4.39}$$

with the increment $\Delta w = \Delta t(-A)w_j$ and approximated slope $-Aw_j$. The equation in (4.39) is the calculation rule for the explicit Euler's method and can further be simplified to

$$w_{j+1} = (1 - \Delta tA)w_j$$

to show the growth factor $(1 - \Delta tA)$. The explicit method is only conditionally stable as mentioned in Chapter 4.1.1. Therefore, the implicit Euler's method is derived below. The forward difference quotient in equation (4.37) is replaced by the backwards difference quotient in accordance with (4.20) to yield

$$\frac{w(t_j) - w(t_j - \Delta t)}{\Delta t} + Aw(t_j) \approx 0, \qquad j = 2, \dots, M.$$

The equality in the approximation is enforced by replacing the function values $w(t_j)$ by values w_j with $w(t_j) \approx w_j$ to

$$\frac{w_j - w_{j-1}}{\Delta t} + Aw_j = 0, \qquad j = 2, \dots, M.$$

Aligning the index to the index of the forward difference quotient with index j starting at 1 instead of 2 results in

$$\frac{w_{j+1} - w_j}{\Delta t} + Aw_j = 0, \qquad j = 1, \dots, M-1.$$

The equation can be rearranged analogously to (4.39) to

$$w_{j+1} = w_j + \underbrace{\Delta t(-A)w_{j+1}}_{\Delta w} \tag{4.40}$$

with the increment $\Delta w = \Delta t(-A)w_{j+1}$ and approximated slope $-Aw_{j+1}$. The equation (4.40) is resolved according to w_{j+1} to the geometric sequence

$$w_{j+1} = \frac{1}{1 + \Delta tA} w_j, \qquad j = 1, \dots, M-1.$$

After deriving the explicit and implicit Euler's method, both approximations can be combined in one equation with parameter θ_d determining the convex combination of the two increments $\Delta t(-A)w_j$ and $\Delta t(-A)w_{j+1}$.

The calculation rule is given by

$$w_{j+1} = w_j + (1 - \theta_d)\Delta t(-A)w_j + \theta_d \Delta t(-A)w_{j+1}, \quad j = 1, \dots, M-1 \tag{4.41}$$

and represents the so-called theta method. The equation (4.41) can be assorted to

$$(1 + \Delta t\theta_d A)w_{j+1} = (1 - \Delta t(1 - \theta_d)A)w_j, \qquad j = 1, \dots, M-1.$$

The parameter θ_d denotes the implicitness of Euler's method, with $\theta_d = 0$ denoting the explicit method, $\theta_d = 1$ denoting the implicit Euler's method, and $\theta_d = 0.5$ denoting the Crank-Nicolson method. The unique characteristic of $\theta_d = 0.5$ is that the Crank-Nicolson method is second-order accurate $\mathcal{O}(\Delta x^2)$, while for $\theta_d \neq 0.5$, the approximation is only first-order accurate $\mathcal{O}(\Delta x)$. Thus, halving the time step Δt, the error term for $\theta_d = 0.5$ is reduced quadratically by a factor of four, while the error is only linearly reduced, by factor two, for $\theta_d \neq 0.5$ (Hilber, 2023, p. 128-138).

4.1.4. Boundary Conditions

The restricted grid for the finite difference method is defined by its boundary conditions, which are determined by the characteristic of the financial product. The boundary conditions can be defined either as Dirichlet boundary condition, which defines the boundary for specified function values or as Neumann boundary condition, which specifies the boundary for a normal derivative of the solution. Additionally, boundary conditions are referred to as homogeneous if the value of a function or derivative at the boundary is equal to zero. Otherwise, the boundary condition is inhomogeneous (Seydel, 2017, p. 147-149).

The boundary conditions of the differential equation in (4.32) were defined as homogenous Dirichlet boundary conditions with $w_{min}(t) = w_{max}(t) = 0$. Subsequently, different boundary conditions are derived, including their universal implementation, by replacing the homogenous boundary conditions of (4.32) with the variable $w^{(n)}$ and replacing the logarithmic payoff by the payoff $g(x)$ in

$$\begin{cases} \partial_t w(x,t) + \mathrm{a}\partial_{xx} w(x,t) + \mathrm{b}\partial_x w(x,t) & \\ \qquad + \mathrm{c}w(x,t) = 0 & \text{in } G_x \times]0,T] \\ w^{(n_{min})}(x_{min},t) = w_{min}(\mathrm{t}) & \text{in }]0,T] \\ w^{(n_{max})}(x_{max},t) = w_{max}(\mathrm{t}) & \text{in }]0,T] \\ w(x,0) = g(x) & \text{in } G_x \end{cases}.$$

The lower boundary point is given by $x_1 = x_{min}$, the upper boundary point is given by $x_N = x_{max}$, and the interior points are given by $x_i, i = 2, \dots, N-1$. The superscript (n) represents the order of derivatives for $n = 0, 1, 2$ with n_{min} for the lower boundary and n_{max} for the upper boundary of the interval G_x. First, the Dirichlet boundary conditions are addressed, representing the function value without derivatives, denoted as $n_{min} = n_{max} = 0$. The discretisation of the differential equations similar to equation (4.33) originates in the system of equations

$$\begin{aligned} w'_i(t) + a(x_i)\frac{w_{i+1}(t) - 2w_i(t) + w_{i-1}(t)}{\Delta x^2} & \\ + b(x_i)\frac{w_{i+1}(t) - w_{i-1}(t)}{2\Delta x} + c(x_i)w_i(t) = 0. & \end{aligned} \tag{4.42}$$

Applying the equation for the interior points yields the identical equation from the previous chapter, which is given by

$$w'_i(t) + \left(\frac{a(x_i)}{\Delta x^2} - \frac{b(x_i)}{2\Delta x}\right) w_{i-1}(t) + \left(-\frac{2a(x_i)}{\Delta x^2} + c(x_i)\right) w_i(t) + \left(\frac{a(x_i)}{\Delta x^2} + \frac{b(x_i)}{2\Delta x}\right) u_{i+1}(t) = 0. \tag{4.43}$$

The equation for the first point $w_{min} = w_1(t)$ is attained by inserting the first nodes into the equation and taking the terms for w_{min} to the right side of the equation by

$$w'_2(t) + a(x_2)\frac{w_3(t) - 2w_2(t)}{\Delta x^2} + b(x_2)\frac{w_3(t)}{2\Delta x} + c(x_2)w_2(t) = \left(-\frac{a(x_2)}{\Delta x^2} + \frac{b(x_2)}{2\Delta x}\right) w_{min}(t) \tag{4.44}$$

and taking the last point $w_{max} = w_N(t)$ and rewriting the equation results in

$$w'_{N-1}(t) + a(x_{N-1})\frac{-2w_{N-1}(t) + w_{N-2}(t)}{\Delta x^2} + b(x_{N-1})\frac{-w_{N-2}(t)}{2\Delta x} + c(x_{N-1})w_{N-1}(t) = \left(-\frac{a(x_{N-1})}{\Delta x^2} - \frac{b(x_{N-1})}{2\Delta x}\right) w_{max}(t).$$

Rewriting the equations into matrix notation, as in the previous chapter, results in

$$\mathbf{w}'(t) + \mathbf{A}\mathbf{w}(t) = \mathbf{f}(t), \qquad \boldsymbol{w}(1) = \mathbf{g} \tag{4.45}$$

with the vector $\mathbf{w}(t)$ as seen in (4.34) and matrix $\mathbf{A}$ as in (4.30) with the modified vector $\mathbf{f}(t)$ with length $N - 2$, which incorporates the boundary conditions

$$\mathbf{f}(t) = -\begin{pmatrix} \alpha_2 w_{min}(t) \\ 0 \\ \vdots \\ 0 \\ \gamma_{N-1} w_{min}(t) \end{pmatrix}$$

and can be written by the sum of matrices with the superscript bc for boundary condition and the left subscript d for Dirichlet condition

$$\mathbf{f}(t) = -\left({}_{\boldsymbol{d}}\mathbf{M}_a^{(2),bc} + {}_{\boldsymbol{d}}\mathbf{M}_b^{(1),bc}\right)\mathbf{w}^{bc}(t),$$

and continuing the notation for the general function y from (4.31) for

$$
{}_{\boldsymbol{d}}\mathbf{M}_{y}^{(2),bc} := \frac{1}{\Delta x^2}\begin{pmatrix} y_2 & 0 & \dots & 0 & 0 \\ 0 & 0 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 0 & 0 \\ 0 & 0 & \dots & 0 & y_{N-1} \end{pmatrix},
$$

$$
{}_{\boldsymbol{d}}\mathbf{M}_{y}^{(1),bc} := \frac{1}{2\Delta x}\begin{pmatrix} -y_2 & 0 & \dots & 0 & 0 \\ 0 & 0 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 0 & 0 \\ 0 & 0 & \dots & 0 & -y_{N-1} \end{pmatrix},
$$

$$
\mathbf{w}^{bc}(t) := \begin{pmatrix} w_{min}(t) \\ 0 \\ \vdots \\ 0 \\ w_{max}(t) \end{pmatrix}.
$$

Approximating the value by applying the theta method analogous to (4.41) yields

$$
\mathbf{w}_{j+1} - \mathbf{w}_j + \Delta t(1-\theta_d)\mathbf{A}\mathbf{w}_j + \Delta t\theta_d\mathbf{A}\mathbf{w}_{j+1} = \Delta t\mathbf{f}_j
$$

with vector $\mathbf{f}_j$ given by

$$
\mathbf{f}_j := \mathbf{f}\big(t_j + \Delta t\theta_d\big) = -\Big(\mathbf{M}_a^{(2),bc} + \mathbf{M}_b^{(1),bc}\Big)\mathbf{w}^{bc}\big(t_j + \Delta t\theta_d\big).
$$

Thus, the equation can be written by the $N \times N$ identity matrix $\mathbf{I}$ to result in M systems of equations

$$
(\mathbf{I} + \Delta t\theta_d\mathbf{A})w_{j+1} = (\mathbf{I} - \Delta t(1-\theta_d)\mathbf{A})\mathbf{w}_j + \Delta t\mathbf{f}_j, \qquad j = 1, \dots, M-1 \quad (4.46)
$$

with $\mathbf{w}_1 = \mathbf{g}$ (Hilber, 2023, p. 140, 176-178).

Secondly, the Neumann boundary conditions are addressed with $n_{min} = n_{max} = 1$, which results in the first derivative of $w(x,t)$. The first derivatives $\partial_x w(x_{min},t) = w_{min}(t)$ and $\partial_x w(x_{max},t) = w_{max}(t)$ must first be calculated whereby only points in the grid can be used. Consequently, the boundary point itself is found by discretisation and the applicable different quotient used for the discretisation is given by

$$
f'(x) = \frac{\pm 3f(x) \mp 4f(x \mp \Delta x) \pm f(x \mp 2\Delta x)}{2\Delta x} + \mathcal{O}(\Delta x^2). \qquad (4.47)
$$

The error term $\mathcal{O}(\Delta x^2)$ of equation (4.47) can be omitted to result in the approximation, and the grid notation $\partial_x w(x_{max},t) \approx w_{max}(t)$ is inserted to enforce equality. Thus, the upper boundary point $w_{max}(t) = w_N(t)$ is approximated by

$$w_{max}(t) = \frac{3w_N(t) - 4w_{N-1}(t) + w_{N-2}(t)}{2\Delta x}.$$

Rewriting the equation according to $w_N(t)$ leads to

$$w_N(t) = \frac{2}{3}\Delta x w_{max}(t) + \frac{4}{3}w_{N-1}(t) - \frac{1}{3}w_{N-2}(t).$$

Consequently, the expression can be inserted into the equation from (4.33) for the last node point to be described by

$$w'_{N-1}(t) + a(x_{N-1})\frac{\frac{2}{3}\Delta x w_{max}(t) - \frac{2}{3}w_{N-1}(t) + \frac{2}{3}w_{N-2}(t)}{\Delta x^2}$$
$$+b(x_{N-1})\frac{\frac{2}{3}\Delta x w_{max}(t) + \frac{4}{3}w_{N-1}(t) - \frac{4}{3}w_{N-2}(t)}{2\Delta x} + c(x_{N-1})w_{N-1}(t) = 0.$$

Hence, the term $w_N(t)$ is omitted and the terms can be assorted to

$$w'_{N-1}(t) + \left(\frac{2a(x_{N-1})}{3\Delta x^2} - \frac{2b(x_{N-1})}{3\Delta x}\right)w_{N-2}(t)$$
$$+\left(-\frac{2a(x_{N-1})}{3\Delta x^2} + \frac{2b(x_{N-1})}{3\Delta x} + c(x_{N-1})\right)w_{N-1}(t)$$
$$= -\left(\frac{2a(x_{N-1})}{3\Delta x} + \frac{b(x_{N-1})}{3}\right)w_{max}(t)$$

The equation for the first point on the grid $w_{min}(t) = w_1(t)$ is derived by the same procedure and results in

$$w'_2(t) + \left(-\frac{2a(x_2)}{3\Delta x^2} - \frac{2b(s_2)}{3\Delta x} + c(x_2)\right)w_2(t) + \left(\frac{2a(x_2)}{3\Delta x^2} + \frac{2b(s_2)}{3\Delta x}\right)w_3(t)$$
$$= \left(\frac{2a(x_2)}{3\Delta x} - \frac{b(s_2)}{3}\right)w_{min}(t).$$

The equation for the interior points is the identical equation as in (4.43), and the equations can be rewritten as

$$\mathbf{w}'(t) + \mathbf{A}\mathbf{w}(t) = \mathbf{f}(t), \qquad \mathbf{w}(1) = \mathbf{g}$$

with matrix $\mathbf{A}$ and vector $\mathbf{f}(t)$ given by

$$\mathbf{A} = {}_{n1}\mathbf{M}_a^{(2)} + {}_{n1}\mathbf{M}_b^{(1)} + \mathbf{M}_c^{(0)},$$
$$\mathbf{f}(t) = -\left({}_{n1}\mathbf{M}_a^{(2),bc} + {}_{n1}\mathbf{M}_b^{(1),bc}\right)\mathbf{w}^{bc}(t)$$

with the left subscript $n1$ for the first-order Neumann boundary condition. The general case with function y results in matrices

$$
{}_{n1}\mathbf{M}_y^{(2)} := \frac{1}{\Delta x^2}\begin{pmatrix} -\frac{2}{3}y_2 & \frac{2}{3}y_2 & & & \\ y_3 & -2y_3 & y_3 & & \\ & \ddots & & \ddots & \\ & & y_{N-2} & -2y_{N-2} & y_{N-2} \\ & & & \frac{2}{3}y_{N-1} & -\frac{2}{3}y_{N-1} \end{pmatrix},
$$

$$
{}_{n1}\mathbf{M}_y^{(1)} := \frac{1}{2\Delta x}\begin{pmatrix} -\frac{4}{3}y_2 & \frac{4}{3}y_2 & & & \\ -y_3 & 0 & y_3 & & \\ & \ddots & & \ddots & \\ & & -y_{N-2} & 0 & y_{N-2} \\ & & & -\frac{4}{3}y_{N-1} & \frac{4}{3}y_{N-1} \end{pmatrix},
$$

$$
\boldsymbol{{}_{n1}\mathbf{M}_y^{(2),bc}} := \frac{2}{3\Delta x}\begin{pmatrix} -y_2 & 0 & \dots & 0 & 0 \\ 0 & 0 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 0 & 0 \\ 0 & 0 & \dots & 0 & y_{N-1} \end{pmatrix},
$$

$$
\boldsymbol{{}_{n1}\mathbf{M}_y^{(1),bc}} := \frac{1}{3}\begin{pmatrix} y_2 & 0 & \dots & 0 & 0 \\ 0 & 0 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 0 & 0 \\ 0 & 0 & \dots & 0 & y_{N-1} \end{pmatrix}.
$$

Discretising the equations with the theta method, as in (4.46), results in

$$
(\mathbf{I} + \Delta t\theta_d\mathbf{A})w_{j+1} = (\mathbf{I} - \Delta t(1-\theta_d)\mathbf{A})\mathbf{w}_j + \Delta t\mathbf{f}_j, \qquad j = 1, \dots, M-1
$$

with vector $\mathbf{f}_j$, matrix $\mathbf{A}$, and initial vector $\mathbf{w}_1 = \mathbf{g}$.

Thirdly, the second-order derivative as Neumann boundary conditions is addressed with $n_{min} = n_{max} = 2$. The different quotient for the second derivative is given by

$$
f''(x) = \frac{2f(x) - 5f(x \pm \Delta x) + 4f(x \pm 2\Delta x) - f(x \pm 3\Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2)
$$

with $+$ for the upper boundary and $-$ for the lower boundary. Following the same process as above leads to the discretisation with vector $\mathbf{f}_j = \mathbf{f}\big(t_j + \Delta t\theta_d\big)$, and the matrix $\mathbf{A}$ and vector $\mathbf{f}(t)$ given by

$$\mathbf{A} = {}_{n2}\mathbf{M}_a^{(2)} + {}_{n2}\mathbf{M}_b^{(1)} + \mathbf{M}_c^{(0)},$$
$$\mathbf{f}(t) = -\left({}_{n2}\mathbf{M}_a^{(2),bc} + {}_{n2}\mathbf{M}_b^{(1),bc}\right)\mathbf{w}^{bc}(t).$$

The left subscript $n2$ denotes the second-order Neumann condition with $\mathbf{w}_1 = \mathbf{g}$ and the matrices

$${}_{n2}\mathbf{M}_y^{(2)} := \frac{1}{\Delta x^2}\begin{pmatrix} \frac{1}{2}y_2 & -y_2 & \frac{1}{2}y_2 & & \\ y_3 & -2y_3 & y_3 & & \\ & \ddots & & \ddots & \\ & & y_{N-2} & -2y_{N-2} & y_{N-2} \\ & & \frac{1}{2}y_{N-1} & -y_{N-1} & \frac{1}{2}y_{N-1} \end{pmatrix},$$

$${}_{n2}\mathbf{M}_y^{(1)} := \frac{1}{2\Delta x}\begin{pmatrix} -\frac{5}{2}y_2 & 3y_2 & -\frac{1}{2}y_2 & & \\ -y_3 & 0 & y_3 & & \\ & \ddots & & \ddots & \\ & & -y_{N-2} & 0 & y_{N-1} \\ & & \frac{1}{2}y_{N-1} & -3y_{N-1} & \frac{5}{2}y_{N-1} \end{pmatrix},$$

$$\boldsymbol{{}_{n2}\mathbf{M}_y^{(2),bc}} := \frac{1}{2}\begin{pmatrix} y_2 & 0 & \dots & 0 & 0 \\ 0 & 0 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 0 & 0 \\ 0 & 0 & \dots & 0 & y_{N-1} \end{pmatrix},$$

$$\boldsymbol{{}_{n2}\mathbf{M}_y^{(1),bc}} := \frac{\Delta x}{4}\begin{pmatrix} -y_2 & 0 & \dots & 0 & 0 \\ 0 & 0 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 0 & 0 \\ 0 & 0 & \dots & 0 & y_{N-1} \end{pmatrix}$$

for the second-order Neumann boundary conditions (Hilber, 2023, p. 176-182).

Alternatively, certain differential equations possess intrinsic boundary conditions such as the Feller condition and no explicit Dirichlet or Neumann boundary conditions must be defined. For instance, the differential equation with assumed intrinsic boundary conditions

$$\begin{cases} \partial_t w(x,t) + \mathrm{a}\partial_{xx} w(x,t) + \mathrm{b}\partial_x w(x,t) + \mathrm{c}w(x,t) = 0 & \text{in } G_i \times]0,T] \\ w(x,0) = g(x) & \text{in } G_i \end{cases}$$

can be discretised in the interval $G_i = [x_{min}, x_{max}]$ without further boundary conditions. The second-order discretisation is performed analogously to the previous discretisation with the system of equations given by (4.42). The equations of the boundary points are

formed with the difference quotients from (4.47) for the first-order derivative and (4.55) for the second-order derivative. The lower boundary is given by

$$w'_1(t) + a(x_1)\frac{-w_4(t) + 4w_3(t) - 5w_2(t) + 2w_1(t)}{\Delta x^2} + b(s_1)\frac{-w_3(t) + 4w_2(t) - 3w_1(t)}{2\Delta x} + c(s_1)w_1(t) = 0$$

while the upper boundary point is given by

$$w'_N(t) + a(x_N)\frac{2w_N(t) - 5w_{N-1}(t) + 4w_{N-2}(t) - w_{N-3}(t)}{\Delta x^2} + b(x_n)\frac{3w_N(t) - 4w_{N-1}(t) + w_{N-2}(t)}{2\Delta x} + c(x_N)w_N(t) = 0$$

The resulting system of equations consists of $N-2$ intrinsic equation plus two additional equations for the two boundary points. Thus, N equations are written as

$$\mathbf{w}'(t) + \mathbf{A}\mathbf{w}(t) = \mathbf{f}(t)$$

with the $N \times N$ matrix

$$\mathbf{A} = {}_i\mathbf{M}_a^{(2)} + {}_i\mathbf{M}_b^{(1)} + {}_i\mathbf{M}_c^{(0)}$$

with the subscript i denoting the intrinsic boundary condition. The general definition is

$${}_i\mathbf{M}_y^{(2)} := \frac{1}{\Delta x^2}\begin{pmatrix} 2y_1 & -5y_1 & 4y_1 & -y_1 & & & \\ y_2 & -2y_2 & y_2 & & & & \\ & y_3 & -2y_3 & y_3 & & & \\ & & \ddots & & \ddots & & \\ & & & y_{N-1} & -2y_{N-1} & y_{N-1} & \\ & & -y_N & 4y_N & -5y_N & 2y_N & \end{pmatrix},$$

$${}_i\mathbf{M}_y^{(1)} := \frac{1}{2\Delta x}\begin{pmatrix} -3y_1 & 4y_1 & -y_1 & & & & \\ -y_2 & 0 & y_2 & & & & \\ & -y_3 & & y_3 & & & \\ & & \ddots & & \ddots & & \\ & & & -y_{N-1} & & y_{N-1} & \\ & & & y_N & -4y_N & 3y_N & \end{pmatrix},$$

$${}_i\mathbf{M}_y^{(0)} := \begin{pmatrix} y_1 & & & & \\ & y_2 & & & \\ & & \ddots & & \\ & & & y_{N-1} & \\ & & & & y_N \end{pmatrix}.$$

The discussed boundary conditions, namely the Dirichlet, Neumann, and intrinsic boundary conditions, enable the setting of an accurate grid for option pricing. The different types of boundary conditions can individually be combined to account for differing characteristics. The equation for interior grid points is identical, while the equations for the first and last points change depending on the boundary condition. Hence, the system of differential equations $\mathbf{w}'(t) + \mathbf{A}\mathbf{w}(t) = \mathbf{f}(t)$ only alters for matrix $\mathbf{A}$ and vector $\mathbf{f}(t)$ for different boundary conditions (Hilber, 2023, p. 191-194).

4.2. Finite Differences in Two Dimensions

The Feynman-Kac theorem establishes a link between derivative prices as expectation values with PDEs. This connection is demonstrated by considering the Black-Scholes differential equation

$$\begin{cases} \partial_t V + \mathcal{A}V - r(t)V = -\Delta t(s,t) & \text{in } G \times [0,T[\\ V(s,T) = g(s) & \text{in } G \end{cases}$$

with the infinitesimal generator $\mathcal{A}$ given by

$$\mathcal{A}f = \frac{1}{2}\sigma(s,t)^2 \partial_{ss} f + \mu(s,t)\partial_s f.$$

Consequently, the case is analysed for two processes $S_1(t)$ and $S_2(t)$, representing two different stochastic differential equations. The vector of the two processes is

$$\mathbf{S}(t) = \begin{pmatrix} S_1(t) \\ S_2(t) \end{pmatrix} \in G \subset \mathbb{R}^d$$

with the corresponding stochastic differential equations for both processes

$$\mathrm{d}S_i(t) = \mu_i(\mathbf{S}(t),t)\mathrm{d}t + \sum_{k=1}^{2} \sigma_{ik}(\mathbf{S}(t),t)\mathrm{d}W_k(t).$$

The functions μ_i, σ_{ik}, and W_k are compiled with $\big(\mathbf{s} = (s_1, s_2)\big)$ in

$$\boldsymbol{\mu}(\mathbf{s},t) = \begin{pmatrix} \mu_1(\mathbf{s},t) \\ \mu_1(\mathbf{s},t) \end{pmatrix}, \qquad \boldsymbol{\sigma}(\mathbf{s},t) = \begin{pmatrix} \sigma_{11}(\mathbf{s},t) & \sigma_{12}(\mathbf{s},t) \\ \sigma_{21}(\mathbf{s},t) & \sigma_{22}(\mathbf{s},t) \end{pmatrix}, \qquad \mathbf{W}(t) = \begin{pmatrix} W_1(t) \\ W_2(t) \end{pmatrix}$$

and can be written as a one-dimensional differential equation

$$\mathrm{d}\mathbf{S}(t) = \boldsymbol{\mu}(\mathbf{S}(t),t)\mathrm{d}t + \boldsymbol{\sigma}(\mathbf{S}(t),t)\mathrm{d}\mathbf{W}(t).$$

The 2×2 covariance matrix $\mathbf{Q}$ is given by $\mathbf{Q}(\mathbf{s},t) = \boldsymbol{\sigma}(\mathbf{s},t)\boldsymbol{\sigma}(\mathbf{s},t)^\top$, and the infinitesimal generator $\mathcal{A}$ of the processes is

$$\mathcal{A}f = \frac{1}{2}\mathrm{tr}[\mathbf{Q}(\mathbf{s},t)D^2 f] + \boldsymbol{\mu}(\mathbf{s},t)^\top D^1 f$$

with "tr" denoting "trace". Therefore, the trace of a square matrix is the sum of its diagonal elements. When matrix $\mathbf{A} = \left(A_{ij}\right)$ is a $d \times d$ matrix, then $\mathrm{tr}[\mathbf{A}] = \sum_{i=1}^{d} A_{ii}$. The first and second-order partial derivatives are compiled in

$$D^1 f(\mathbf{s}) = \begin{pmatrix} \partial_{s_1} f(\mathbf{s}) \\ \partial_{s_2} f(\mathbf{s}) \end{pmatrix}, \qquad D^2 f(\mathbf{s}) = \begin{pmatrix} \partial_{s_1 s_1} f(\mathbf{s}) & \partial_{s_1 s_2} f(\mathbf{s}) \\ \partial_{s_2 s_1} f(\mathbf{s}) & \partial_{s_2 s_2} f(\mathbf{s}) \end{pmatrix}.$$

The two processes with $S_i(t),\ i = 1,2$ are given by

$$\mathrm{d}S_i(t) = \mu_i S_i(t)\mathrm{d}t + S_i(t)\sum_{j=1}^{d} L_{ij} dW_j(t)$$

with vector $\boldsymbol{\mu}$ and matrix $\boldsymbol{\sigma}$

$$\boldsymbol{\mu}(\mathbf{s}) = \begin{pmatrix} \mu_1 s_1 \\ \mu_2 s_2 \end{pmatrix}, \qquad \boldsymbol{\sigma}(\mathbf{s}) = \begin{pmatrix} s_1 & \\ & s_2 \end{pmatrix}\mathbf{L} := \mathrm{diag}(\mathbf{s})\mathbf{L}$$

with 2×2 matrix $\mathbf{L} = \left(L_{ij}\right)$. Thus, the covariance matrix $\mathbf{Q}$ is given by

$$\mathbf{Q}(\mathbf{s}) = \boldsymbol{\sigma}(\mathbf{s})\boldsymbol{\sigma}(\mathbf{s})^\top = \mathrm{diag}(\mathbf{s})\mathbf{L}\mathbf{L}^\top diag(\mathbf{s}) = \begin{pmatrix} \sigma_1^2 s_1^2 & \sigma_{12} s_1 s_2 \\ \sigma_{12} s_1 s_2 & \sigma_2^2 s_2^2 \end{pmatrix}$$

with notation $\sigma_{ij} = (\mathbf{L}\mathbf{L}^\top)_{ij}$ and $\sigma_i^2 = \sigma_{ii} = (\mathbf{L}\mathbf{L}^\top)_{ii}$. The diagonal elements are determined by the matrix

$$\mathbf{Q}(\mathbf{s})D^2 f = \begin{pmatrix} \sigma_1^2 \sigma_1^2 & \sigma_{12} s_1 s_2 \\ \sigma_{12} s_1 s_2 & \sigma_2^2 \sigma_2^2 \end{pmatrix}\begin{pmatrix} \partial_{s_1 s_1} f & \partial_{s_1 s_2} f \\ \partial_{s_2 s_1} f & \partial_{s_2 s_2} f \end{pmatrix}.$$

The trace of the 2×2 matrix $\mathbf{Q}(\mathbf{s})D^2 f$ is given by

$$\mathrm{tr}[\mathbf{Q}(\mathbf{s})D^2 f] = \sigma_1^2 s_1^2 \partial_{s_1 s_1} f + \sigma_{12} s_1 s_2 \partial_{s_2 s_1} f + \sigma_{12} s_1 s_2 \partial_{s_1 s_2} f + \sigma_2^2 s_2^2 \partial_{s_2, s_2} f$$

and assuming that the order has no influence according to Clairaut's theorem. Then, the trace results in

$$\boldsymbol{\mu}(\mathbf{s})^\top D^1 f = \begin{pmatrix} \mu_1 s_1 & \mu_2 s_2 \end{pmatrix}\begin{pmatrix} \partial_{s_1} f \\ \partial_{s_2} f \end{pmatrix} = \mu_1 s_1 \partial_{s_1} f + \mu_2 s_2 \partial_{s_2} f.$$

Thus, the solutions can be compiled to result in the infinitesimal generator $\mathcal{A}$

$$\mathcal{A}f = \frac{1}{2}\sigma_1^2 s_1^2 \partial_{s_1 s_1} f + \frac{1}{2}\sigma_2^2 s_2^2 \partial_{s_2 s_2} f + \sigma_{12} s_1 s_2 \partial_{s_1 s_2} f + \mu_1 s_1 \partial_{s_1} f + \mu_2 s_2 \partial_{s_2} f$$

(Hilber, 2023, p. 326-329). Inserting the infinitesimal generator $\mathcal{A}$ into the differential equation and replacing μ_i with $r - q_i$ results in

$$\begin{cases} \partial_t V + \frac{1}{2}\sigma_1^2 s_1^2 \partial_{s_1 s_1} V + \frac{1}{2}\sigma_2^2 s_2^2 \partial_{s_2 s_2} V + \sigma_{12} s_1 s_2 \partial_{s_1 s_2} V \\ \qquad + (r - q_1) s_1 \partial_{s_1} V + (r - q_2) s_2 \partial_{s_2} V - rV = 0 & \text{in } G \times [0, T[\\ V(s_1, s_2, T) = g(s_1, s_2) & \text{in } G \end{cases}$$

with grid $G =]0, \infty[^2$. The equation is again simplified as in the case (3.12) by introducing a logarithmic payoff function $s_i = \ln(s_i)$ and substituting function $v(e^{s_1}, e^{s_2}, T - t)$ for $V(s_1, s_2, t)$ to write the equation with constant coefficients

$$\begin{cases} \partial_t v + a_1 \partial_{s_1 s_1} v + a_2 \partial_{s_2 s_2} v + a_3 \partial_{s_1 s_2} v + b_1 \partial_{s_1} v + b_2 \partial_{s_2} v + cv = 0 & \text{in } G \times]0, T] \\ v(s_1, s_2, T) = g(e^{s_1}, e^{s_2}) & \text{in } G \end{cases}$$

with $a_i = -\frac{1}{2}\sigma_i^2, a_3 = -\sigma_{12}, b_i = \frac{1}{2}\sigma_i^2 - r + q_i$ and $c = r$. Consequently, the index x replaces s_1 and y replaces s_2 in the new interval $G_z =]x_{min}, x_{max}[\times]y_{min}, y_{max}[$ with suitable boundary conditions (BC). The equation is given by

$$\begin{cases} \partial_t w + a_1(x, y)\partial_{xx} w + a_2(x, y)\partial_{yy} w \\ \qquad + a_3(x, y)\partial_{xy} w + b_1(x, y)\partial_x w \\ \qquad + b_2(x, y)\partial_y w + c(x, y) w = 0 & \text{in } G_z \times [0, T[\\ \qquad \mathbf{BC} & \text{in } \partial G_z \times [0, T[\\ w(x, y, 0) = g(x, y) & \text{in } G_z \end{cases} \tag{4.48}$$

with continuous functions a_i, b_i, and c given as the product of univariate functions

$$\begin{gathered} a_i(x, y) = a_i^x(x) a_i^y(y), \qquad b_i(x, y) = b_i^x(x) b_i^y(y), \\ c(x, y) = c^x(x) c^y(y). \end{gathered} \tag{4.49}$$

The two variables require a two-dimensional discretisation grid for the calculation of finite differences. Let Δx and Δy denote the discretisation increments of variables x and y. The equidistant grid of x and y is given by

$$G_{x,y} = \{(x_i, y_j) \mid 1 \le i \le N_x, 1 \le j \le N_y\} \tag{4.50}$$

with the minimum values (x_{min}, y_{min}), maximum values (x_{max}, y_{max}), and

$$x_i = x_{min} + i\Delta x, \qquad \Delta x = \frac{x_{max} - x_{min}}{N_x - 1}$$

$$y_j = y_{min} + j\Delta y, \qquad \Delta y = \frac{y_{max} - y_{min}}{N_y - 1}$$

in the grid $G_{x,y} = [x_{min}, x_{max}] \times [y_{min}, y_{max}]$. While the theoretical solution $w(x, y)$ is continuous, the approximations $w_{i,j}$ are only defined on the intersections of the grid lines $x = x_i$ and $y = y_j$ forming the node points or centred between the node points in case of the Crank-Nicolson method with the central difference quotient (Hilber, 2023, p. 332-334; Seydel, 2017, p. 135-137).

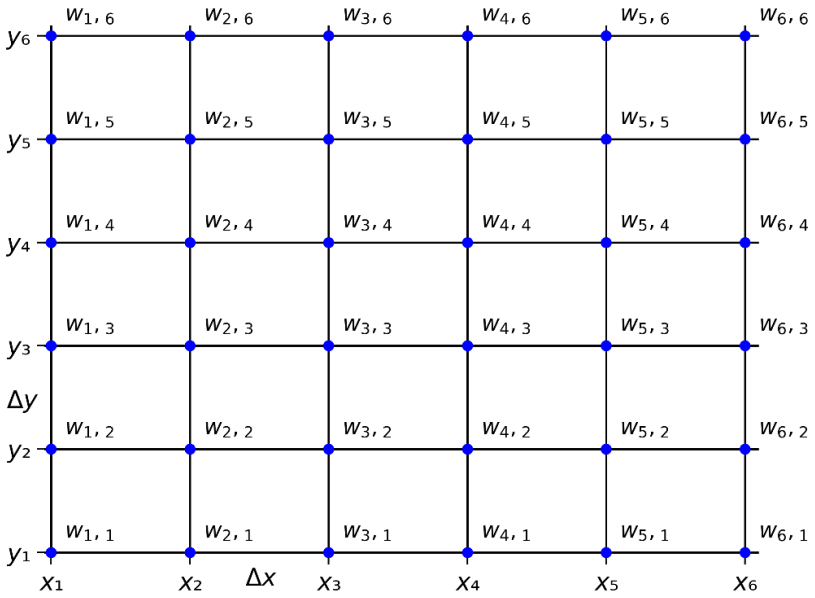

Figure 4.1: Illustration of a two-dimensional grid $G_{x,y}$ with $N_x = 6$ and $N_y = 6$, where the values $w(x_i, y_j) \approx w_{i,j}$ are approximated at every node point $w(x, y)$ in reference to Hilber (2023, p. 334).

Rewriting the first-order central difference quotient in (4.22) for the two variables and replacing the function values $w(x_i, y_j)$ by the approximated values $w_{i,j}$ with $w(x_i, y_j) \approx w_{i,j}$ leads to

$$\partial_x w_{i,j} \approx \frac{w_{i+1,j} - w_{i-1,j}}{2\Delta x} \tag{4.51}$$

$$\partial_y w_{i,j} \approx \frac{w_{i,j+1} - w_{i,j-1}}{2\Delta y} \tag{4.52}$$

and rewriting the second-order central difference quotient in (4.24) to

$$\partial_{xx} w_{i,j} \approx \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{\Delta x^2} \tag{4.53}$$

$$\partial_{xy} w_{i,j} \approx \frac{w_{i+1,j+1} - w_{i+1,j-1} - w_{i-1,j+1} + w_{i-1,j-1}}{4\Delta x \Delta y} \tag{4.54}$$

$$\partial_{yy} w_{i,j} \approx \frac{w_{i,j+1} - 2w_{i,j} + w_{i,j-1}}{\Delta y^2} \tag{4.55}$$

(Rouah, 2015, p. 239-240). To illustrate the principle, the term $a_1(x,y)\partial_{xx} w$ of equation (4.48) is approximated. An arbitrary node point (x_i, y_i) is discretised by

$$a_1(x_i, y_j)\partial_{xx} w(x_i, y_j) \approx \frac{a_1(x_i, y_j)}{\Delta x^2}\left(w_{i+1,j} - 2w_{i,j} + w_{i-1,j}\right)$$

with index $i, i = 2, \dots, N_x - 1$ and $j, j = 2, \dots, N_y - 1$. Thus, $(N_x - 2)(N_y - 2)$ grid points are estimated while the boundary points are given. The matrix $\mathbf{M}_{a_1^x}^{(2)}$ and the vector $\mathbf{w}_2$ are introduced and exemplified for the row $j = 2$ with $a_{1,i}^x := a_1^x(x_i)$ by

$$\mathbf{M}_{a_1^x}^{(2)} = \frac{1}{\Delta x^2}\begin{pmatrix} -2a_{1,2}^x & a_{1,2}^x & & & & \\ a_{1,3}^x & -2a_{1,3}^x & a_{1,3}^x & & & \\ & & \ddots & & & \\ & & a_{1,N_y-2}^x & -2a_{1,N_y-2}^x & a_{1,N_y-2}^x & \\ & & & a_{1,N_y-1}^x & -2a_{1,N_y-1}^x \end{pmatrix},$$

$$\mathbf{w}_2 = \begin{pmatrix} w_{2,2} \\ w_{3,2} \\ \vdots \\ w_{N_x-2,2} \\ w_{N_x-1,2} \end{pmatrix}.$$

Thus, the terms for row $j = 2$ are written with $a_{1,j}^y := a_1^y(y_j)$ as

$$a_{1,2}^y \mathbf{M}_{a_1^x}^{(2)} \mathbf{w}_2.$$

The same procedure for row $j = 3$ and vector $\mathbf{w}_3 = \left(w_{2,3}, \dots, w_{N_{x-1},3}\right)^\top$ yields in

$$a_{1,3}^y \mathbf{M}_{a_1^x}^{(2)} \mathbf{w}_3.$$

Continuing the procedure for the remaining rows and generating the vectors leads to

$$\begin{pmatrix} a_{1,2}^{y}\mathbf{M}_{a_1^x}^{(2)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & a_{1,3}^{y}\mathbf{M}_{a_1^x}^{(2)} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ & & \ddots & & \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & a_{1,N_y-2}^{y}\mathbf{M}_{a_1^x}^{(2)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & a_{1,N_y-1}^{y}\mathbf{M}_{a_1^x}^{(2)} \end{pmatrix}, \qquad \mathbf{w} = \begin{pmatrix} \mathbf{w}_2 \\ \mathbf{w}_3 \\ \vdots \\ \mathbf{w}_{N_y-2} \\ \mathbf{w}_{N_y-1} \end{pmatrix}$$

with zero matrices $\mathbf{0}$. Defining the $(N_y - 2) \times (N_y - 2)$ matrix

$$\mathbf{M}_{a_1^y}^{(0)} = \begin{pmatrix} a_{1,2}^{y} & 0 & 0 & 0 & 0 \\ 0 & a_{1,3}^{y} & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & a_{1,N_y-2}^{y} & 0 \\ 0 & 0 & 0 & 0 & a_{1,N_y-2}^{y} \end{pmatrix}$$

enables the multiplication of both matrices as a Kronecker product

$$\mathbf{M}_{a_1^y}^{(0)} \otimes \mathbf{M}_{a_1^x}^{(2)}$$

and the terms of the discretisation of $a_1^x(x)a_1^y(y)\partial_{xx}w$ can be written as

$$\left(\mathbf{M}_{a_1^y}^{(0)} \otimes \mathbf{M}_{a_1^x}^{(2)}\right)\mathbf{w}.$$

In addition, the Kronecker product of the discretisation of $a_3(x, y)\partial_{xy}w$ is illustrated with the node point (x_i, y_j) by

$$a_3(x_i, y_j)\partial_{xy}w(x_i, y_j) \approx \frac{a_3^x(x_i)a_3^y(y_j)}{4\Delta x\Delta y}\left(w_{i+1,j+1} - w_{i+1,j-1} - w_{i-1,j+1} + w_{i-1,j-1}\right).$$

The boundary points are known and the expressions for the interior points of row $j = 2$ are written as

$$\frac{a_{3,2}^{y}}{2\Delta y}\mathbf{M}_{a_3^x}^{(1)}\mathbf{w}_3$$

with matrix $\mathbf{M}_{a_3^x}^{(1)}$ given by

$$\mathbf{M}_{a_3^x}^{(1)} = \frac{1}{2\Delta y}\begin{pmatrix} 0 & a_{3,2}^{x} & 0 & 0 & 0 & 0 \\ -a_{3,3}^{x} & 0 & a_{3,3}^{x} & 0 & 0 & 0 \\ 0 & -a_{3,4}^{x} & 0 & a_{3,4}^{x} & 0 & 0 \\ & & & \ddots & & \\ 0 & 0 & 0 & -a_{3,N_y-2}^{x} & 0 & a_{3,N_y-2}^{x} \\ 0 & 0 & 0 & 0 & -a_{3,N_y-1}^{x} & 0 \end{pmatrix}.$$

Row $j = 3$ can be written as

$$\frac{a_{3,3}^{y}}{2\Delta y}\mathbf{M}_{a_3^x}^{(1)}(-\mathbf{w}_2 + \mathbf{w}_4)$$

and the same procedure is applied to the subsequent rows. Summarising the terms for all rows with zero matrices $\mathbf{0}$ leads to the matrix

$$\frac{1}{2\Delta y}\begin{pmatrix} \mathbf{0} & a_{3,2}^{y}\mathbf{M}_{a_3^x}^{(1)} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ -a_{3,3}^{y}\mathbf{M}_{a_3^x}^{(1)} & \mathbf{0} & a_{3,3}^{y}\mathbf{M}_{a_3^x}^{(1)} & \mathbf{0} & \mathbf{0} \\ & \ddots & & \ddots & \\ \mathbf{0} & \mathbf{0} & -a_{3,N_y-2}^{y}\mathbf{M}_{a_3^x}^{(1)} & \mathbf{0} & a_{3,N_y-2}^{y}\mathbf{M}_{a_3^x}^{(1)} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & -a_{3,N_y-1}^{y}\mathbf{M}_{a_3^x}^{(1)} & \mathbf{0} \end{pmatrix}\begin{pmatrix} \mathbf{w}_2 \\ \mathbf{w}_3 \\ \vdots \\ \mathbf{w}_{N_y-2} \\ \mathbf{w}_{N_y-1} \end{pmatrix}.$$

Consequently, rewriting results in the $(N_y - 2) \times (N_y - 2)$ matrix

$$\mathbf{M}_{a_3^y}^{(1)} = \frac{1}{2\Delta y}\begin{pmatrix} 0 & a_{3,2}^{y} & 0 & 0 & 0 \\ -a_{3,3}^{y} & 0 & a_{3,3}^{y} & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & -a_{3,N_y-2}^{y} & 0 & a_{3,N_y-2}^{y} \\ 0 & 0 & 0 & -a_{3,N_y-1}^{y} & 0 \end{pmatrix}$$

and the discretisation of $a_3(x,y)\partial_{xy}w$ results in

$$\left(\mathbf{M}_{a_3^y}^{(1)} \otimes \mathbf{M}_{a_3^x}^{(1)}\right)\mathbf{w}.$$

Applying the same procedure for the remaining terms, the discretisation of the differential equation (4.48) can be determined on the finite difference grid (4.50).

The coefficients $a_1, a_2, a_3, b_1, b_2, c$ of equation (4.48) in the form of the product structure (4.49) can be calculated by the partial derivatives of the matrices

$$\begin{aligned} a_1(x,y)\partial_{xx} &\Rightarrow \mathbf{M}_{a_1^y}^{(0)} \otimes \mathbf{M}_{a_1^x}^{(2)} \\ a_2(x,y)\partial_{yy} &\Rightarrow \mathbf{M}_{a_2^y}^{(2)} \otimes \mathbf{M}_{a_2^x}^{(0)} \\ a_3(x,y)\partial_{xy} &\Rightarrow \mathbf{M}_{a_3^y}^{(1)} \otimes \mathbf{M}_{a_3^x}^{(1)} \\ b_1(x,y)\partial_{x} &\Rightarrow \mathbf{M}_{b_1^y}^{(0)} \otimes \mathbf{M}_{b_1^x}^{(1)} \\ b_2(x,y)\partial_{y} &\Rightarrow \mathbf{M}_{b_2^y}^{(1)} \otimes \mathbf{M}_{b_2^x}^{(0)} \\ c(x,y) &\Rightarrow \mathbf{M}_{c^y}^{(0)} \otimes \mathbf{M}_{c^x}^{(0)}. \end{aligned}$$

The matrices with respect to x are $(N_x - 2) \times (N_x - 2)$ matrices, and with respect to y are $(N_y - 2) \times (N_y - 2)$ matrices. Thus, the discretisation of the infinitesimal generator $\mathcal{A}$ is given by the sum of the Kronecker products

$$\begin{aligned}\mathbf{A} := \mathbf{M}_{a_1^y}^{(0)} \otimes \mathbf{M}_{a_1^x}^{(2)} + \mathbf{M}_{a_2^y}^{(2)} \otimes \mathbf{M}_{a_2^x}^{(0)} + \mathbf{M}_{a_3^y}^{(1)} \otimes \mathbf{M}_{a_3^x}^{(1)} + \mathbf{M}_{b_1^y}^{(0)} \otimes \mathbf{M}_{b_1^x}^{(1)} \\ + \mathbf{M}_{b_2^y}^{(1)} \otimes \mathbf{M}_{b_2^x}^{(0)} + \mathbf{M}_{c^y}^{(0)} \otimes \mathbf{M}_{c^x}^{(0)}.\end{aligned} \tag{4.56}$$

The determination of differential equations for the interior points can be written analogously to (4.45) as

$$\mathbf{w}'(t) + \mathbf{A}\mathbf{w}(t) = \mathbf{f}(t), \qquad \boldsymbol{w}(1) = \mathbf{g}$$

with vector $\mathbf{f}(t)$ dependent on the specified boundary conditions and vector $\mathbf{g}$ results from the payoff function $g(x, y)$ by

$$\mathbf{g} = \begin{pmatrix} g(x_2, y_2) \\ g(x_3, y_3) \\ \vdots \\ g(x_i, y_j) \\ \vdots \\ g(x_{N_x-1}, y_{N_x-1}) \end{pmatrix}.$$

Furthermore, the boundary conditions of the grid (4.50) for the differential equation (4.48) and the above vector $\mathbf{f}$ are given by

$$\text{BC} = \begin{cases} (\mathcal{B}_{min}^y w)(x, y_{min}, t) = w_{min}^x(x, t) & \text{in} \quad]x_{min}, x_{max}[\times]0, T[\\ (\mathcal{B}_{max}^y w)(x, y_{max}, t) = w_{max}^x(x, t) & \text{in} \quad]x_{min}, x_{max}[\times]0, T[\\ (\mathcal{B}_{min}^x w)(x_{min}, y, t) = w_{min}^y(x, t) & \text{in} \quad]y_{min}, y_{max}[\times]0, T[\\ (\mathcal{B}_{max}^x w)(x_{max}, y, t) = w_{max}^y(x, t) & \text{in} \quad]y_{min}, y_{max}[\times]0, T[\end{cases}$$

with $\mathcal{B}_{min,max}^{x,y} w$ indicating a Dirichlet, Neumann or intrinsic boundary condition for each side of the two-dimensional discretisation grid (Hilber, 2023, p. 334-339).

5. Implementation

The valuation of lookback options is executed in the Heston volatility model as explained in Chapter 3.3 and discretised by the finite difference method in two dimensions. First, the basis of the valuation is provided by the decisive PDE for lookback options in the Heston model from Leung (2013, p. 146)

$$\begin{cases} \partial_t V + \frac{1}{2} v s^2 \partial_{ss} V + \frac{1}{2}\sigma^2 v \partial_{vv} V \\ \quad + \rho\sigma v s \partial_{sv} V + (r-q) s \partial_s V \\ \quad\quad + \kappa(\theta_m - v)\partial_v V - rV = 0 & \text{in } G_{x,y} \times [0, T[\\ \quad\quad\quad \partial_{s^*} V(s, s^*, v, t) = 0 & \text{in } [0, T[\\ \quad\quad\quad\quad V(s, s^*, v, T) = f(s, s^*) & \text{in } G_{x,y} \end{cases} \tag{5.1}$$

with the volatility of the variance σ, $0 \leq t \leq T$, $s > 0$, and $\omega s \leq s^*$. The variable s^* represents either S_{min} for $\omega = -1$ or S_{max} for $\omega = 1$. Moreover, the payoff function $f(s, s^*)$ is given by

$$f(s, s^*) = s g\left(\ln \frac{s^*}{s}\right)$$

for function g. The dimensions of equation (5.1) are reduced from 4 to 3 by transforming $x = \ln\left(\frac{s^*}{s}\right)$ and $U = \frac{V}{s}$. First, variable s is multiplied on the left side to yield $sU = V$. Thus, the functions are given by

$$V(s, s^*, v, t) = sU(x, v, t)$$

with the variables s^*, s being replaced by x. The individual terms of equation (5.1) are transformed by partial derivation, whereby additional derivation rules are applied for the derivatives of s because of the change in variables. The resulting derivatives are given by

$$\begin{aligned} \partial_t V &= s\partial_t U \\ \partial_s V &= U + s\partial_x U * \frac{s}{s^*} * (-s^*) * \frac{1}{s^2} = U - \partial_x U \\ \partial_{ss} V &= \partial_x U * \frac{s}{s^*} * (-s^*) * \frac{1}{s^2} - \partial_{xx} U * \frac{s}{s^*} * (-s^*) * \frac{1}{s^2} = \frac{1}{s}(\partial_{xx} U - \partial_x U) \\ \partial_v V &= s\partial_v U \\ \partial_{vv} V &= s\partial_{vv} U \\ \partial_{sv} V &= \partial_v U - \partial_{xv} U. \end{aligned}$$

The transformed equations are given by adding all derivatives to obtain

$$s\partial_t U + \frac{1}{2}vs^2 * \frac{1}{s}(\partial_{xx}U - \partial_x U) + \frac{1}{2}\sigma^2 v * s\partial_{vv}U + \rho\sigma vs * (\partial_v U - \partial_{xv}U) + (r-q)s * (U - \partial_x U) + \kappa(\theta_m - v) * s\partial_v U - r * sU = 0$$

and simplifying the equation by reducing the positive variable s and assorting all partial derivatives leads to the equation

$$\partial_t U + \frac{1}{2}v\partial_{xx}U + \frac{1}{2}\sigma^2 v\partial_{vv}U - \rho\sigma v\partial_{xv}U - \left(r - q + \frac{v}{2}\right)\partial_x U + (\kappa(\theta_m - v) + \rho\sigma v)\partial_v U - qU = 0. \tag{5.2}$$

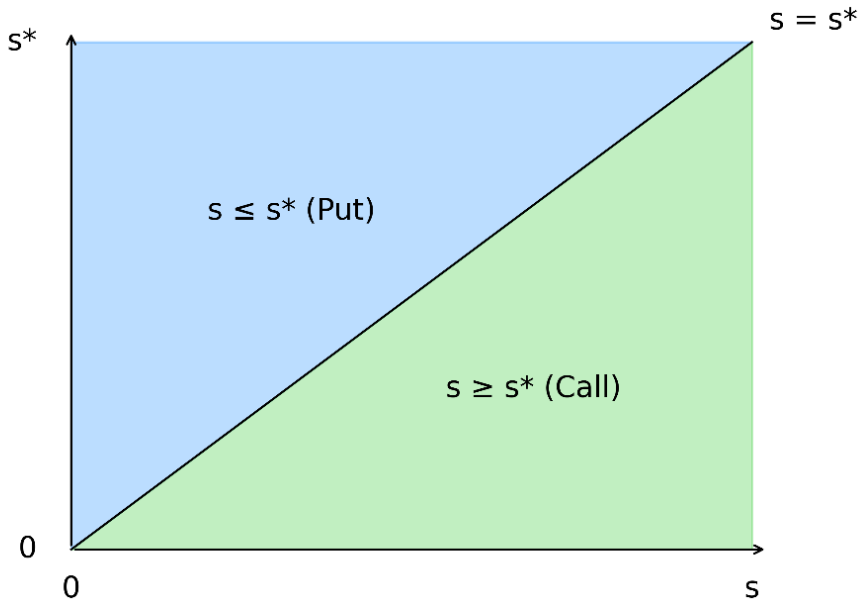

Figure 5.1: The grid of a Lookback call option with s^* representing the maximum or minimum process, in reference to Hilber (2023, p. 368).

In addition, this transformation removes a further problem. The payoff of lookback options depends on the maximum for put options and the minimum for call options. Consequently, only in the case of a put option can s^* be greater than s and only in the case of a call option can s^* be smaller than s. Thus, the variables (s, s^*) are arranged in a triangle, while the finite difference method requires a quadrilateral area. As shown above, this problem can be solved by setting s^* and s in proportion and replacing the ratio with x to result in a quadrilateral area. Hence, the transformation enables the application of the finite difference method for lookback options.

Replacing the volatility $v = y$ in equation (5.2) leads to the definitive equation

$$\begin{cases} \partial_t U + \frac{1}{2} y \partial_{xx} U + \frac{1}{2} \sigma^2 y \partial_{yy} U \\ -\rho\sigma y \partial_{xy} U - \left(r - q + \frac{y}{2}\right) \partial_x U \\ +(\kappa(\theta_m - y) + \rho\sigma y)\partial_y U - qU = 0 & \text{in } G_l \times [0, T[\\ \partial_x U(0, y, t) = 0 & \text{in } [0, T[\\ U(x, y, T) = g(x) & \text{in } G_l \end{cases} \tag{5.3}$$

Furthermore, equation (5.3) can be rewritten in the form known from (4.48) to enable the calculation with the introduced concepts and leads to

$$\partial_t U + a_1(x,y)\partial_{xx} U + a_2(x,y)\partial_{yy} U + a_3(x,y)\partial_{xy} U + b_1(x,y)\partial_x U + b_2(x,y)\partial_y U + c(x,y)U = 0$$

with its coefficients

$$\begin{aligned} &a_1(x,y) = \frac{1}{2} y, \qquad a_2(x,y) = \frac{1}{2}\sigma^2 y, \qquad a_3(x,y) = -\rho\sigma y, \\ &b_1(x,y) = -\left(r - q + \frac{y}{2}\right), \qquad b_2(x,y) = (\kappa(\theta_m - y) + \rho\sigma y), \\ &c(x,y) = -q. \end{aligned} \tag{5.4}$$

This form enables the usage of the infinitesimal generator $\mathcal{A}$ for discretisation through the sum of the Kronecker products (4.56).

The two-dimensional grid for lookback options is defined with the two variables x and y and their corresponding increments Δx and Δy. The equidistant grid is given by

$$G_l = \{(x_i, y_j) \mid 1 \le i \le N_x, 1 \le j \le N_y\} \tag{5.5}$$

with the minimum values (x_{min}, y_{min}), the maximum values (x_{max}, y_{max}), and

$$\begin{aligned} x_i &= x_{min} + i\Delta x, \qquad \Delta x = \frac{x_{max} - x_{min}}{N_x} \\ y_j &= y_{min} + j\Delta y, \qquad \Delta y = \frac{y_{max} - y_{min}}{N_y} \end{aligned}$$

in the grid $G_l = [x_{min}, x_{max}] \times [y_{min}, y_{max}]$. Moreover, the boundary conditions for lookback options are defined. The value of variable x is the natural logarithm of the ratio between the minimum or maximum of the asset price and the asset price itself.

Hence, variable x takes negative values for ratios with $S^* < S$ and positive values for $S^* > S$. Therefore, the lower and upper boundary conditions of x are defined as Neumann boundary conditions with the first derivative equal to zero. The lower boundary for y is given by the intrinsic boundary condition of the Heston model. The introduced Feller condition at the end of Chapter 3.3 reassures that the volatility is positive as long as the Feller condition is satisfied. Hence, the stochastic term is inapplicable when the variance touches zero, and the deterministic term increases the value, thus bringing the variance back into positive terrain. Otherwise, the volatility can assume zero value if the Feller condition is not satisfied (Austing, 2014, p. 80-82). Supplementary, Ekström and Tysk (2010, p. 499-503) validate that the volatility equals zero even if the Feller condition is not satisfied. The upper boundary for the volatility y is defined as Neumann boundary conditions with the first derivative equal to zero.

$$\begin{cases} \partial_x U(0,y,t) = 0 \\ \partial_x U(\infty,y,T) = 0 \\ \quad U(x,0,t) \rightarrow intrinsic\ BC \\ \partial_y U(x,\infty,t) = 0 \end{cases} \tag{5.6}$$

5.1. Application in Python

The valuation of lookback options is performed in the programming language Python, which enables the calculation of complex routines with a short processing time. The approximation of equation (5.3) first requires the creation of applicable matrices, including the boundary conditions. For this purpose, the matrixgenerator_BC.py routine of Hilber (2023, p. 183-185) is utilised to generate the matrices (4.31) with the boundary matrices dependent on the selected boundary conditions covered in Chapter 4.1.4. Secondly, the pde_2d_ah_theta.py routine of Hilber (2023, p. 341) is applied to approximate the solution PDE with the form of (4.48). The pricing formula of lookback options with the corresponding equation is deduced in the previous intercept and must be adapted for the routine. The first adaption is the change from time t to the term of maturity $\tau = T - t$ in the terms (5.4). Thus, applying the chain rule as shown in (3.10) results in the change of signs $\partial_\tau v = -\partial_t V$ and the term of maturity is redefined as t for simplicity.

The new terms with changed signs are given by

$$\begin{aligned} &a_1(x,y) = -\frac{1}{2}y, \qquad a_2(x,y) = -\frac{1}{2}\sigma^2 y, \qquad a_3(x,y) = \rho\sigma y, \\ &b_1(x,y) = r - q + \frac{y}{2}, \qquad b_2(x,y) = -(\kappa(\theta_m - y) + \rho\sigma y), \\ &c(x,y) = q. \end{aligned} \tag{5.7}$$

Furthermore, the continuous functions a_i, b_i, and c are transformed into products of univariate functions. This transformation is executed by separating the terms dependent on x and y. For technical purposes, the function is defined as the variable to the power of 0 if a variable is not present in a term. Finally, the 12 functions are defined via the lambda function in Python and assigned to the variables a, b, and c:

```
a = [lambda x: x**0, lambda y: -0.5*y,
     lambda x: x**0, lambda y: -0.5*sigma**2*y,
     lambda x: x**0, lambda y: rho*sigma*y]
b = [lambda x: x**0, lambda y: (r-q+0.5*y),
     lambda x: x**0, lambda y: -(kappa*(theta_m-y)+rho*sigma*y)]
c = [lambda x: q*x**0, lambda y: y**0]
```

Moreover, the boundary conditions, as defined in (5.6), are adapted for the routine. The list of boundary conditions is defined as

$$\mathbf{BC} = \left[n^x_{min}, n^x_{max}, n^y_{min}, n^y_{max}\right]$$

with $n \in \{0, 1, 2, 3\}$ representative for Dirichlet, Neumann first-order, Neumann second-order or intrinsic boundary condition (Hilber, 2023, p. 340). Therefore, the list with the defined boundary conditions for lookback options is given by

$$\mathbf{BC} = [1, 1, 3, 1].$$

The payoff function defined by Leung (2013, p. 146-148) must satisfy the linear homogeneous property to apply to the formula in (5.1). Floating lookback options fulfil the linear homogeneous property, while fixed lookback options do not satisfy the property. Thus, the discretisation can only be conducted for floating lookback options. Due to the reduction of the dimension to x, the new initial payoff function is given for the floating put option with $x = \ln\left(\frac{s^*}{s}\right)$ by

$$f(s, s^*) = s(e^x - 1).$$

In addition, the transformation in (5.2) from four to three variables with $U = \frac{V}{s}$ is conducted with the s being omitted. Moreover, ω is integrated to allow the discretisation

of put options with $\omega = 1$ and call options with $\omega = -1$ through the same function. The calculation of partial lookback options is enabled by inserting λ into the payoff function to receive

$$f(s, s^*) = \omega(\lambda e^x - 1).$$

The matrices, the boundary conditions, and the payoff function remain defined for different types of lookback options and are not dependent on individual values. The finite difference grid $G_{x,y}$ is defined by the list of boundaries in

$$\mathbf{G} = [x_{min}, x_{max}, y_{min}, y_{max}]$$

with the number of grid points $\mathbf{N} = [N_x, N_y]$ relating to the x- and y-axis as well as the number of time steps M. The initial x-values for the grid are $\mathbf{G} = [0, 3, 0.0, 0.8]$ for put options and $\mathbf{G} = [-3, 0, 0.0, 0.8]$ for call options due to the domain, evident in Figure 5.1. The grid values can be adjusted, and the number of grid points must be defined independently for each input.

Further inputs for solving the routine are determined through a Heston model calibration. They are the continuous risk-free interest rate r, the continuous dividend yield q, the term of maturity T, the instantaneous variance v, the volatility of the instantaneous variance σ, the correlation ρ, the mean reversion speed of the variance κ, the mean reversion level of the variance θ_m, and the initial stock price S_0. The Greek letter θ represents the parameter of the theta method and the model parameter of lookback options for the mean reversion level. These two parameters are distinguished by denominating the discretisation parameter `theta_d` and denominating the model parameter `theta_m`. The strike price K is only required for the calculation of fixed lookback call options, while $\omega \in \{-1, 1\}$ is required to define the type of lookback option and represent either S_{min} or S_{max}. The parameter λ denoted `l` can be defined for partial lookback options with $\lambda \geq 1$ for call and $0 < \lambda \leq 1$ for put options.

5.2. Interpolation and Put-Call Parity

The price of floating lookback options can be calculated after all inputs have been determined and the routine has generated the three matrices **x**, **y**, and **w**. Matrix **x** contains all grid points related to variable x, matrix **y** contains all grid points related to variable y, and matrix **w** contains all option prices at the individual grid points. The calculation of continuous values within the grid is enabled by interpolating the matrices and generating a surface representing the option values. Thus, a value can not only be defined at the node points but also in between the node points for each coordinate. The interpolation in Python is executed by the function `interpn` of the NumPy library with the method "`pchip`", which conducts cubic interpolation at every node point and preserves the shape of the arrangement. The option value would be defined at the lower edge of the grid, but the grid is limited by the Neumann boundary condition of the first-order at the lower boundary of x. Thus, the grid points required for the valuation of the price are outside the grid. This problem can be solved by setting the `bounds_error=False` and `fill_value=None` in the interpn function. Setting `bounds_error` to "False" activates `fill_value`, which extrapolates the values outside the grid for "None". Consequently, the function will estimate the node point outside the grid by employing the pchip method.

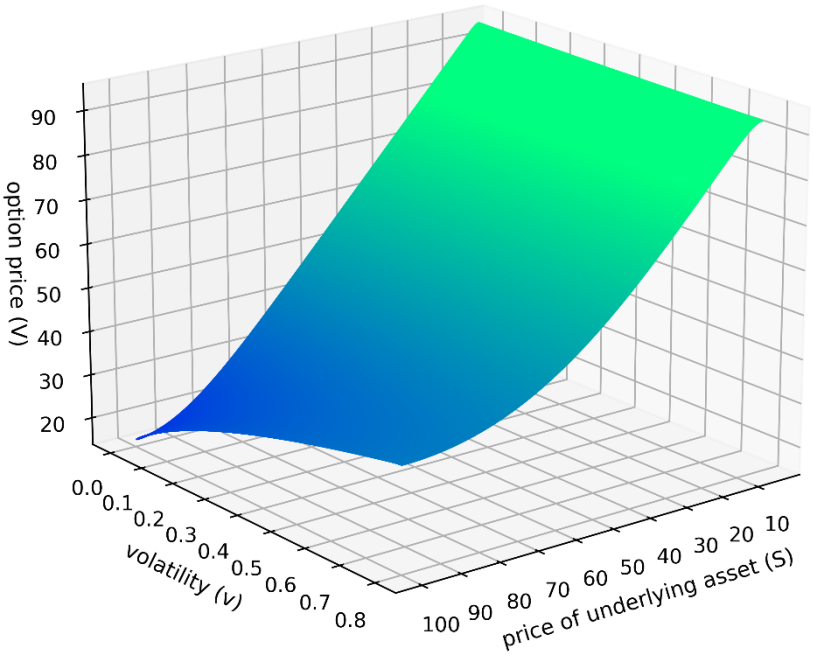

Figure 5.2: Option prices of floating strike lookback put options.

The option value is calculated at specific coordinates, which are dependent on the calibrated parameters. The x-coordinate is indirectly defined by Leung (2013, p. 148) in the put-call parity. In the parity, the floating lookback put option depends on the time t, the asset price at time t denoted S, and the maximum of S_{max} or strike K. Thus, the maximum of S_{max} or K replaces S^* in $x = \ln\left(\frac{S^*}{S}\right)$. Therefore, x is given by

$$x = \ln\left(\frac{max(S_{max}, K)}{S}\right) = \ln\left(max\left(\frac{S_{max}}{S}, \frac{K}{S}\right)\right),$$

given that $S_0 > 0$. Moreover, the maximum achieved asset price S_{max} and the current asset price S are equal to the initial asset price S_0 at the valuation date since the term has not started and there is no recorded lookback period yet. Consequently, the variables are replaced and transformed as follows.

$$x = \ln\left(\max\left(\frac{S_0}{S_0}, \frac{K}{S_0}\right)\right.$$

$$x = \ln\left(\max\left(1, \frac{K}{S_0}\right)\right.$$

$$x = \max\left(0, ln\left(\frac{K}{S_0}\right)\right)$$

The y-coordinate is defined by the initial instantaneous variance v given by the calibrated parameter. Thus, the coordinates for the determination of the price are given by

$$\left(\max\left(0, ln\left(\frac{K}{S_0}\right)\right), v\right).$$

The payoff function of the PDE for lookback options in (5.1) satisfies the linear homogeneous property only for floating lookback options. Consequently, the price of the fixed lookback option must be obtained by different means. Leung (2013, p. 148) presents in his paper a put-call parity relation for lookback options, which enables a fast and efficient calculation of fixed lookback call options. The option price is received by the price of the floating lookback put option and a so-called replenishing premium and is given by

$$C_{fix}(t, S, S_{max}, K) = P_{fl}(t, S, \max(S_{max}, K)) + Se^{-q(T-t)} - Ke^{-r(T-t)}.$$

The parity for fixed lookback put options is dependent on the price of the floating lookback call option and is defined by Wong and Kwok (2003, p. 89) as

$$C_{fl}(t, S, S_{min}) = S - e^{-(r-q)(T-t)}S_{min} + P_{fix}(t, S, S_{min})$$

and can be transposed to

$$P_{fix}(t,S,S_{min}) = C_{fl}(t,S,S_{min}) - S + e^{-(r-q)(T-t)}S_{min}$$

with the current asset price S. It should be noted that no strike occurs in the put-call parity for fixed lookback put options. Hence, implementing the interpolation and the put-call parity in Python yields:

```
01: from scipy.interpolate import interpn
02:
03: if omega == -1:
04:     C_fl = interpn((x[:,0],y[0,:]), w*s0, (np.maximum(np.log(K/s0),0), v),
05:                    method='pchip', bounds_error=False, fill_value=None);
06:     print('floating Call:', C_fl)
07:     P_fix = C_fl-s0+np.exp(-(r-q)*T)*s0; print('fixed Put:', P_fix)
08: elif omega == 1:
09:     P_fl = interpn((x[:,0],y[0,:]), w*s0, (np.maximum(np.log(K/s0),0), v),
10:                    method='pchip', bounds_error=False, fill_value=None);
11:     print('floating Put:', P_fl)
12:     C_fix = P_fl+s0*np.exp(-q*T)-K*np.exp(-r*T); print('fixed Call:', C_fix)
13: else:
14:     print('Wrong value for Omega, enter -1 for call or 1 for put options.')
```

5.3. Craig-Sneyd Method

The calculation of option prices with the finite difference method shown in Chapter 5.1 requires considerable computation power due to large-dimensioned matrices. Especially the case of tiny Δx and Δy leads to a considerable computation time and can exhaust the capacity of the computation device. To address this limitation, Craig and Sneyd (1988, p. 341-349) published an implicit method to execute iterative methods in less time with second-order accuracy and unconditional stability for two and three-dimensional problems. Furthermore, the method takes advantage of the tridiagonal matrix structure, allowing specialised algorithms to compute the result considerably faster. For example, a tridiagonal matrix $\mathbf{M}_{i,j}$ has elements $\mathbf{M}_{i,j} = non-zero$ if $|i-j| \leq 1$ and $\mathbf{M}_{i,j} = 0$ if $|i-j| > 1$.

$$\mathbf{M} = \begin{pmatrix} a_{11} & a_{12} & & & & & \\ a_{21} & a_{22} & a_{23} & & & & \\ & a_{32} & a_{33} & a_{34} & & & \\ & & & \ddots & & & \\ & & & a_{i-1,j-2} & a_{i-1,j-1} & a_{i-1,j} \\ & & & & a_{i,j-1} & a_{ij} \end{pmatrix}$$

The characteristic of tridiagonal matrices implies that every element is zero except the main diagonal and the diagonals immediately above and below the main diagonal. Hence, the algorithm can ignore all zero elements and only consider the non-zero elements, resulting in an accelerated calculation. The software Python provides efficient algorithms such as the tridiagonal solver in the SciPy library and solves tridiagonal matrices faster than ordinary matrices due to the particular structure.

Subsequently, the Craig-Sneyd method requires all matrices in the two-dimensional case to be tridiagonal. The infinitesimal generator $\mathcal{A}$ from (4.56) exhibits the matrices

$$\mathbf{A} := \mathbf{M}_{a_1^y}^{(0)} \otimes \mathbf{M}_{a_1^x}^{(2)} + \mathbf{M}_{a_2^y}^{(2)} \otimes \mathbf{M}_{a_2^x}^{(0)} + \mathbf{M}_{a_3^y}^{(1)} \otimes \mathbf{M}_{a_3^x}^{(1)} + \mathbf{M}_{b_1^y}^{(0)} \otimes \mathbf{M}_{b_1^x}^{(1)} + \mathbf{M}_{b_2^y}^{(1)} \otimes \mathbf{M}_{b_2^x}^{(0)} + \mathbf{M}_{c^y}^{(0)} \otimes \mathbf{M}_{c^x}^{(0)}$$

with the tridiagonal matrices $\mathbf{M}_{a_1^y}^{(0)} \otimes \mathbf{M}_{a_1^x}^{(2)}, \mathbf{M}_{b_1^y}^{(0)} \otimes \mathbf{M}_{b_1^x}^{(1)}$, and $\mathbf{M}_{c^y}^{(0)} \otimes \mathbf{M}_{c^x}^{(0)}$. These matrices result from the discretisation of the variable x or the residual term, while the matrices $\mathbf{M}_{a_2^y}^{(2)} \otimes \mathbf{M}_{a_2^x}^{(0)}$ and $\mathbf{M}_{b_2^y}^{(1)} \otimes \mathbf{M}_{b_2^x}^{(0)}$ resulting from variable y are not tridiagonal but can be transformed into tridiagonal matrices. Only the matrix $\mathbf{M}_{a_3^y}^{(1)} \otimes \mathbf{M}_{a_3^x}^{(1)}$ originating from the mixed derivation of variables x and y cannot be transformed into a tridiagonal matrix. The matrices are regrouped to enable the accelerated calculation and are written as a sum

$$\mathbf{A} = \mathbf{A}_1 + \mathbf{A}_2 + \mathbf{A}_0$$

with the summands

$$\mathbf{A}_1 = \mathbf{M}_{a_1^y}^{(0)} \otimes \mathbf{M}_{a_1^x}^{(2)} + \mathbf{M}_{b_1^y}^{(0)} \otimes \mathbf{M}_{b_1^x}^{(1)} + \frac{1}{2}\mathbf{M}_{c^y}^{(0)} \otimes \mathbf{M}_{c^x}^{(0)}$$

$$\mathbf{A}_2 = \mathbf{M}_{a_2^y}^{(2)} \otimes \mathbf{M}_{a_2^x}^{(0)} + \mathbf{M}_{b_2^y}^{(1)} \otimes \mathbf{M}_{b_2^x}^{(0)} + \frac{1}{2}\mathbf{M}_{c^y}^{(0)} \otimes \mathbf{M}_{c^x}^{(0)}$$

$$\mathbf{A}_0 = \mathbf{M}_{a_3^y}^{(1)} \otimes \mathbf{M}_{a_3^x}^{(1)}.$$

The Craig-Sneyd schema starts by conducting the explicit Euler's method ($\theta_d = 0$) of (4.38) on the matrices $\mathbf{A}_1$ and $\mathbf{A}_2$. The application of the explicit method in the direction of the x-coordinate for matrix $\mathbf{A}_1$ yields

$$\frac{\mathbf{w}_{j+1} - \mathbf{w}_j}{\Delta t} + \theta_d \mathbf{A}_1 \mathbf{w}_{j+1} + (1 - \theta_d)\mathbf{A}_1 \mathbf{w}_j + (\mathbf{A}_2 + \mathbf{A}_0)\mathbf{w}_j = 0$$

and rewriting results in

$$(\mathbf{I} + \Delta t\theta_d \mathbf{A}_1)\mathbf{w}_{j+1} = (\mathbf{I} - \Delta t(1-\theta_d)\mathbf{A}_1 - \Delta t\mathbf{A}_2 - \Delta t\mathbf{A}_0)\mathbf{w}_j \qquad (5.8)$$

with the now tridiagonal matrix $\mathbf{I} + \Delta t\theta_d \mathbf{A}_1$. The same procedure can be conducted in the direction of the y-coordinate with

$$(\mathbf{I} + \Delta t\theta_d \mathbf{A}_2)\mathbf{w}_{j+1} = (\mathbf{I} - \Delta t\mathbf{A}_1 - \Delta t(1-\theta_d)\mathbf{A}_2 - \Delta t\mathbf{A}_0)\mathbf{w}_j. \qquad (5.9)$$

First, the result for the explicit Euler's method is defined by $\mathbf{y}_0$ through

$$\mathbf{y}_0 = (\mathbf{I} + \Delta t\mathbf{A})\mathbf{w}_j$$

and the system of equations (5.8) and (5.9) can be combined. Solving implicitly in the direction of the x-coordinate returns

$$(\mathbf{I} + \Delta t\theta_d \mathbf{A}_1)\mathbf{y}_1 = \mathbf{y}_0 + \Delta t\theta_d \mathbf{A}_1 \mathbf{w}_j$$

with $\mathbf{y}_1$ corresponding to $\mathbf{w}_{j+1}$ of (5.8). After receiving $\mathbf{y}_1$ from above, solving in the direction of the y-coordinate yields

$$(\mathbf{I} + \Delta t\theta_d \mathbf{A}_2)\mathbf{y}_2 = \mathbf{y}_1 + \Delta t\theta_d \mathbf{A}_2 \mathbf{w}_j.$$

Moreover, instead of replacing $\mathbf{w}_{j+1}$ with $\mathbf{y}_2$ to solve the system of equations in the direction of the y-coordinate, the auxiliary vector $\mathbf{z}_0$ is defined and given by

$$\mathbf{z}_0 = \mathbf{y}_0 - \frac{1}{2}\Delta t\mathbf{A}_0(\mathbf{y}_2 - \mathbf{w}_j).$$

Repeating the steps above and inserting $\mathbf{z}_0$ into the system of equations results in

$$(\mathbf{I} + \Delta t\theta_d \mathbf{A}_1)\mathbf{z}_1 = \mathbf{z}_0 + \Delta t\theta_d \mathbf{A}_1 \mathbf{w}_j,$$
$$(\mathbf{I} + \Delta t\theta_d \mathbf{A}_2)\mathbf{z}_2 = \mathbf{z}_1 + \Delta t\theta_d \mathbf{A}_2 \mathbf{w}_j.$$

Determining $\mathbf{w}_{j+1} = \mathbf{z}_2$ and compiling all equations results in the Craig-Sneyd schema

$$\begin{cases} \mathbf{y}_0 = (\mathbf{I} + \Delta t\mathbf{A})\mathbf{w}_j & \\ (\mathbf{I} + \Delta t\theta_d \mathbf{A}_i)\mathbf{y}_i = \mathbf{y}_{i-1} + \Delta t\theta_d \mathbf{A}_i \mathbf{w}_j & i = 1, \dots, d \\ \mathbf{z}_0 = \mathbf{y}_0 - \frac{1}{2}\Delta t\mathbf{A}_0(\mathbf{y}_d - \mathbf{w}_j) & \\ (\mathbf{I} + \Delta t\theta_d \mathbf{A}_i)\mathbf{z}_i = \mathbf{z}_{i-1} + \Delta t\theta_d \mathbf{A}_i \mathbf{w}_j & i = 1, \dots, d \\ \mathbf{w}_{j+1} = \mathbf{z}_d & \end{cases}$$

with $\mathbf{w}_0 = \mathbf{g}$ and $j = 0, \dots, M-1$. After deriving the schema, the matrix $\mathbf{A}_2$ must be transformed into a tridiagonal matrix to enable the fast solving of systems of equations. Therefore, a permutation matrix is used, and the transformation is illustrated in the

following example. Two matrices $\mathbf{X} \in \mathbb{R}^{n \times n}$ and $\mathbf{Y} \in \mathbb{R}^{m \times m}$ are defined. Given the two matrices, the matrix $\mathbf{P}^{nm \times nm}$ exists with $\mathbf{P}^{\top}\mathbf{P} = \mathbf{P}\mathbf{P}^{\top} = \mathbf{I}$ and is valid for

$$\mathbf{X} \otimes \mathbf{Y} = \mathbf{P}(\mathbf{Y} \otimes \mathbf{X})\mathbf{P}^{\top}.$$

Assuming the matrix $\mathbf{X} \otimes \mathbf{Y}$ to be not tridiagonal, while the matrix $\mathbf{Y} \otimes \mathbf{X}$ is tridiagonal. Then the tridiagonal matrix can be received by solving the system $(\mathbf{X} \otimes \mathbf{Y})\mathbf{x} = \mathbf{f}$ as follows

$$\begin{aligned}
(\mathbf{X} \otimes \mathbf{Y})\mathbf{x} &= \mathbf{f} \\
\mathbf{P}(\mathbf{Y} \otimes \mathbf{X}) \underbrace{\mathbf{P}^{\top}\mathbf{x}}_{\tilde{\mathbf{x}}} &= \mathbf{f} \\
\underbrace{\mathbf{P}^{\top}\mathbf{P}}_{\mathbf{I}} (\mathbf{Y} \otimes \mathbf{X})\tilde{\mathbf{x}} &= \underbrace{\mathbf{P}^{\top}\mathbf{f}}_{\tilde{\mathbf{f}}} \\
(\mathbf{Y} \otimes \mathbf{X})\tilde{\mathbf{x}} &= \tilde{\mathbf{f}}
\end{aligned}$$

The original solution $\mathbf{x}$ results from $\tilde{\mathbf{x}} = \mathbf{P}^{\top}\mathbf{x}$ and its multiplication with $\mathbf{P}$ to receive $\mathbf{P}\tilde{\mathbf{x}} = \mathbf{x}$. Thus, the not tridiagonal system $(\mathbf{X} \otimes \mathbf{Y})\mathbf{x} = \mathbf{f}$ can be solved by the tridiagonal systems $(\mathbf{Y} \otimes \mathbf{X})\tilde{\mathbf{x}} = \tilde{\mathbf{f}}$ with the multiplications $\tilde{\mathbf{f}} = \mathbf{P}^{\top}\mathbf{f}$ and $\mathbf{P}\tilde{\mathbf{x}} = \mathbf{x}$. This transformation allows for shortened processing times, and the matrix $\mathbf{P}$ is defined for two matrices $\mathbf{X} \in \mathbb{R}^{n \times n}$ and $\mathbf{Y} \in \mathbb{R}^{m \times m}$ as

$$\mathbf{P} = \begin{pmatrix} \mathbf{I}_n \otimes \mathbf{1}_{1m}^{\top} \\ \mathbf{I}_n \otimes \mathbf{1}_{2m}^{\top} \\ \vdots \\ \mathbf{I}_n \otimes \mathbf{1}_{mm}^{\top} \end{pmatrix}$$

with the $x \times n$ identity matrix $\mathbf{I}_n$ and the column vector $\mathbf{1}_{jm}$ of length m with input 1 at the jth element and other elements with input 0. The Craig-Sneyd method is unconditionally stable for $\theta_d \geq \frac{1}{2}$ and converges quadratically in Δt. The examined methods are implemented in the get_diagonals.py and perm_matrix.py routines and enable the accelerated discretisation of option prices in the routine pde_2d_ah_cs.py (Hilber, 2023, p. 345-350).

The difference in processing time is subsequently illustrated by utilising the model parameters of set A from De Gennaro Aquino & Bernard (2019, p. 738) for fixed lookback call options. The model inputs remain unchanged while the number of grid points $\mathbf{N} = [N_x, N_y]$ and the number of time steps M changes.

	Theta method t_{CPU}		Craig-Sneyd t_{CPU}	
N	$M = 40$	$M = 100$	$M = 40$	$M = 100$
2’500	0.40591	0.90757	0.07579	0.11668
10’000	2.36468	5.65787	0.17054	0.31715
250’000	152.27192	375.81132	4.31346	9.07174
1’000’000	1’042.99178	2’580.68735	17.77248	37.78199

*Table 5.1: The processing time t_{CPU} in seconds for the discretisation with **N** grid points for the theta and Craig-Sneyd method by the Python function time.time().*

The table illustrates the significant reduction of processing time for discretising lookback options in the Craig-Sneyd scheme. Furthermore, a technical error for large matrices in the theta method can occur when the discretisation is too extensive for the device to process. This limitation is considerably shifted outwards with the Craig-Sneyd method and enables the stable calculation of larger matrices with second-order accuracy.

5.4. Graphical User Interface

A graphical user interface (GUI) was created with the tkinter library in Python to enable a user-friendly operation of the model. This allows the user to efficiently enter the required inputs without searching for the individual variables in the code. The execution of the GUI_lookback_option.py code opens an additional window, as illustrated below.

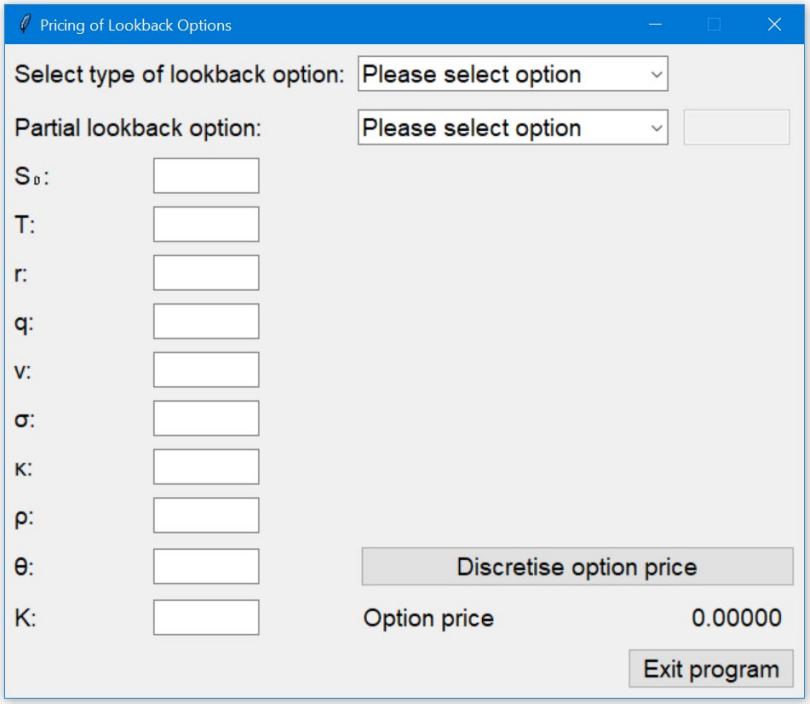

Figure 5.3: Graphical user interface with default settings.

First, the type of lookback option can be selected from the dropdown menu with the four options of floating and fixed, call and put lookback options. Secondly, the user can select whether the inputs are calibrated for a partial lookback option in a dropdown menu. An input box on the right side of the dropdown menu is activated if "Yes" is selected. Otherwise, the box stays deactivated, and the default value $\lambda = 1$ is automatically defined. Furthermore, all other required inputs can be entered in the corresponding input field to the right. An error message automatically occurs when no option in the dropdown menu is chosen, or a value is not entered. Subsequently, the "Discretise option price" button can be clicked, and the routine is executed to evaluate the option price. Moreover, the button is deactivated to prevent multiple executions, and the user is informed about the ongoing process by a notification in the window. The discretisation is conducted in the grid $\mathbf{G} = [0, 3, 0.0, 0.8]$ for floating lookback put and fixed lookback call options and grid $\mathbf{G} = [-3, 0, 0.0, 0.8]$ for floating lookback call and fixed lookback put options with

$\mathbf{N} = [1'000, 1'000]$ grid points and $M = 100$ time steps, which results in an accurate valuation. After the discretisation is executed, the option price is indicated in the bottom right corner of the window, rounded to five decimal places, and the button is enabled for further calculations. In addition, the programme may crash, or the kernel shuts down when the user clicks on the window while the option price is discretised. Hence, the library threading was implemented to prevent the programme from being aborted. Therefore, the window can invariably be clicked or closed during the discretisation process without causing the kernel to crash.

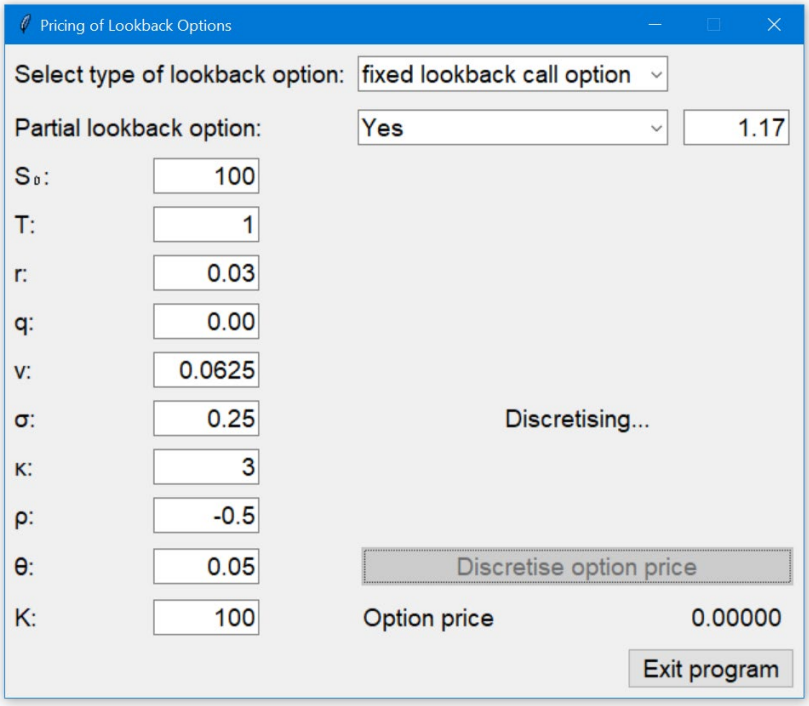

Figure 5.4: Graphical user interface with entered inputs during discretisation.

6. Numerical Analysis and Discussion of Results

The developed model for the evaluation of lookback options is subsequently verified by discretising the prices for lookback options with calibrated parameters defined by De Gennaro Aquino & Bernard (2019, p. 738). The Craig-Sneyd scheme is applied for accelerated processing, and the calibrated parameters with strike $K = \{90, 100, 110\}$ for fixed lookback call options are subsequently given.

	r	q	v	κ	θ	σ	ρ	S_0	T
Set A	0.03	0.00	0.0625	3	0.05	0.25	-0.5	100	1
Set B	0.03	0.00	0.0700	5	0.04	0.50	-0.5	100	1
Set C	0.03	0.00	0.0500	4	0.03	0.40	-0.5	100	1

Table 6.1: Calibrated parameters for fixed lookback call options from Table 6 of De Gennaro Aquino & Bernard (2019, p. 738).

The discretisation parameter θ_d is determined as 0.5 to ensure a stable process with unconditional stability. In addition, the prices for fixed lookback call options are calculated using the put-call parity. Thus, the model must discretise a floating lookback put option, and the parameter ω is set as 1. Furthermore, the paper assumes ordinary lookback options. Consequently, the parameter λ is set to 1 to omit the partial lookback consideration. Moreover, the number of grid points is determined as $\mathbf{N} = [1'000, 1'000]$ with the number of time steps equal to $M = 100$, which ensures sufficient accuracy.

The original paper of De Gennaro Aquino & Bernard (2019, p. 735-738) included values for the different sets with varying strikes obtained by utilising a Monte Carlo simulation. These values were intended as reference values to verify the developed finite difference model. However, the first values obtained in the discretisation with the given parameters resulted in significantly divergent values. Thus, further research into comparative articles and studies was conducted, but no other papers with specific reference values for lookback options in the Heston model were found. Moreover, a very recent correction of De Gennaro Aquino & Bernard (2022, p. 447-448) about their paper published in 2019 was found. In this correction, the authors admit that they had made a mistake in the joint density function of the logarithmic asset price and its maximum and minimum values. The flaw leads to an erroneously averaged variance process, and the provided prices for lookback options are only non-convergent approximations instead of exact prices.

Nonetheless, the calibrated parameters for lookback options are not affected by the error and can therefore be utilised for the determination of option prices. Consequently, this thesis presents corrected prices for fixed lookback call options with the parameter given in Table 6.3.

The accuracy of the developed finite difference model is subsequently verified using a Monte Carlo model, including the "full truncation scheme" of Rouah (2013, p. 177-181). While the variance $V(t)$ in the Heston model (3.13) follows a CIR process, discrete simulations generate negative values for $V(t)$, even if the Feller condition is satisfied. This problem occurs because the Feller condition requires continuous time processes to hold while the Monte Carlo simulation uses discrete time steps. A constraint on the volatility at zero called the full truncation scheme is introduced to avoid negative values and to address this problem. The Monte Carlo model simulates option prices for the calibrated parameters in Table 6.1. with the number of paths set as $N = 200'000$ and the number of time steps set as $M = 20'000$ in the period $[0, T]$. The Monte Carlo prices underestimate the maximum stock price based on the construction and should therefore exhibit slightly lower option prices than the finite difference method. The comparison of option prices for both models is depicted in Table 6.2.

The computed option values exhibit a divergence between -0.01562 and -0.06151, with the Monte Carlo simulation yielding lower values because of the lower valuated maximum. Simulations with a higher number of paths and additional time steps yield more precise results, minimise the deviation, and should confirm the accuracy of the discretised option price for a subsequent decimal place. A study by Zeng, Guo, and Zhu (2017, p. 190-193) compared the fundamental divergence between the finite difference method and the Monte Carlo method and stated the divergence in the third decimal place. The clear disadvantage of the Monte Carlo simulation is the processing time of over 11 hours for one of the simulations below, compared to about 35 seconds for finite differences. Furthermore, discretisations with smaller Δx and more time steps only lead to changes in the third decimal place and thereafter. It should also be noted that the deviation for strikes 90 and 100 are nearly equal, while the option price with a strike of 110 has a distinctly lower deviation. Consequently, the divergence of both methods increases for options without intrinsic value. Thus, the developed finite difference model features high accuracy for the valuation of lookback option prices and possesses superior efficiency.

Set A				
Strike	FDM	Monte Carlo	Deviation	
90	29.52512	29.48611	-0.03901	-0.13213%
100	19.82067	19.78165	-0.03902	-0.19684%
110	11.69203	11.67641	-0.01562	-0.13356%
Set B				
Strike	FDM	Monte Carlo	Deviation	
90	27.27170	27.21019	-0.06151	-0.22554%
100	17.56724	17.50574	-0.06150	-0.35010%
110	9.58052	9.54938	-0.03114	-0.32499%
Set C				
Strike	FDM	Monte Carlo	Deviation	
90	25.01696	24.96064	-0.05632	-0.22514%
100	15.31251	15.25618	-0.05633	-0.36786%
110	7.51159	7.47705	-0.03454	-0.45985%

Table 6.2: Comparison of option prices for the finite difference method with a Monte Carlo simulation.

The proposition of corrected values for De Gennaro Aquino & Bernard (2019, p. 738) is displayed in the column “reference values” for different strikes in Table 6.3. The reference values are rounded to three decimal places in the paper, for which reason the FDM values are equally rounded. The significant deviation of the reference values from the discretised values originates from the averaged variance process, as written in the correction of De Gennaro Aquino & Bernard (2022, p. 448). The discrepancy is most significant for sets B and C and can be explained by elevated values for the volatility of the instantaneous variance σ and the mean reversion speed of the variance κ in addition to reduced values for the mean reversion level θ. Furthermore, the reference values exhibit higher values in all calculations with a deviation between +0.521% and +9.284%. Thus, the average variance process overestimates the prices for fixed lookback call options.

Set A				
Strike	FDM	Reference value	Deviation	
90	29.525	29.679	0.154	0.521%
100	19.821	19.975	0.154	0.779%
110	11.692	11.879	0.187	1.599%
Set B				
Strike	FDM	Reference value	Deviation	
90	27.272	28.350	1.078	3.954%
100	17.567	18.677	1.110	6.317%
110	9.581	10.470	0.889	9.284%
Set C				
Strike	FDM	Reference value	Deviation	
90	25.017	25.387	0.370	1.479%
100	15.313	15.742	0.429	2.805%
110	7.512	7.737	0.225	3.001%

Table 6.3: Proposed values for the calibrated parameters of De Gennaro Aquino & Bernard (2019, p. 738).

7. Conclusion

The primary objective of this thesis was to develop an accurate evaluation model for floating and fixed lookback options in Python. The model evaluates option prices in the Heston model with the discretisation conducted in a two-dimensional finite difference grid. Furthermore, the discretisation is accelerated by applying the Craig-Sneyd scheme, and the option prices are determined at the specified coordinates. The prices for fixed lookback options are obtained by the conversion of floating lookback option values through a put-call parity. Based on the numerical analysis and the comparison to the Monte Carlo simulation, the developed finite difference model delivers highly accurate option prices with superior efficiency and curtate processing times. In addition, a graphical user interface was developed to enable higher usability and accessibility. Moreover, this thesis contributes to research in the field of quantitative finance by proposing the first amended option prices for calibrated parameters of current literature and providing them for all sets and strikes.

This thesis clearly illustrates the practical implementation of the finite difference method and enhanced discretisation processes but also raises the question of whether there is an alternative to evaluate the option price of lookback options with advanced mathematical methods more accurately and efficiently. The exerted pricing formula is based on continuous monitoring, while only discrete monitoring is plausible for real financial products. Nevertheless, a risk-averse stance should be adopted when considering the risk management perspective for issuers, and continuous monitoring should instead be applied for the mitigation of seldom events, influences of human biases or prejudices. In addition, the prices of fixed lookback options are obtained by a put-call parity, raising the question of accuracy. The verification of fixed lookback call option prices has shown exact values, but further research is required to assess the accuracy of the lookback option put-call parity conclusively.

Moreover, the Heston model excels compared to the Black-Scholes model but exhibits some imprecision for short maturities. Hence, further extended models, such as the double Heston model with a two-factor structure for volatility, could be applied for increased accuracy. The domain of lookback options led to the limitation on the lower boundary of the asset price, and extrapolation was required to obtain the option value. Although the accuracy was augmented by progressive extrapolation methods and minute time steps, some marginal variation remains. Alternatively, other approximation methods could be

used but exhibit distinct disadvantages, from prolonged processing times to the dependence on subjective assumptions. Further research could analyse the deviation of efficient processes, such as the finite difference method, from more exact but computationally intensive methods, and a correction for the discretised prices could be developed to enhance the accuracy to several decimal places.

In summary, the discretisation employing finite differences and the Craig-Sneyd scheme for pricing lookback options not only elucidate the numerical valuation of complex financial instruments but also contribute to the field of quantitative finance by proposing corrected values for existing literature. Furthermore, the aggregation of multiple option pricing concepts is applied through practical implementation, consolidating the theoretical concepts in a domain characterised by a limited volume of research. Ultimately, this thesis serves as a robust foundation for future research focused on refining or extending the evaluation of lookback options in the Heston model.

Bibliography

Austing, P. (2014). *Smile Pricing Explained*. Palgrave Macmillan UK. https://doi.org/10.1057/9781137335722

Albrecher, H., Binder, A., & Mayer, P. (2009). *Einführung in die Finanzmathematik*. Birkhäuser. https://doi.org/10.1007/978-3-7643-8784-6

Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *The Journal of Political Economy*, *81*(3), 637–654. https://doi.org/10.1086/260062

Bodie, Z., Kane, A., & Marcus, A. J. (2019). *Essentials of Investments* (11th ed.). McGraw-Hill Education.

Craig, I. J. D., & Sneyd, A. D. (1988). An alternating-direction implicit scheme for parabolic equations with mixed derivatives. *Computers & Mathematics with Applications*, *16*(4), 341–350. https://doi.org/10.1016/0898-1221(88)90150-2

De Gennaro Aquino, L., & Bernard, C. (2019). Semi-analytical prices for lookback and barrier options under the Heston model. *Decisions in Economics and Finance*, *42*(2), 715–741. https://doi.org/10.1007/s10203-019-00254-x

De Gennaro Aquino, L., & Bernard, C. (2022). Correction to: Semi-analytical prices for lookback and barrier options under the Heston model. *Decisions in Economics and Finance*, *45*(1), 447–449. https://doi.org/10.1007/s10203-021-00360-9

Ekström, E., & Tysk, J. (2010). The Black–Scholes equation in stochastic volatility models. *Journal of Mathematical Analysis and Applications*, *368*(2), 498–507. https://doi.org/10.1016/j.jmaa.2010.04.014

EUREX (2023). *Market statistics (online) – DAX Options*. https://www.eurex.com/ex-en/data/statistics/market-statistics-online/100!onlineStats?productGroupId=13394&productId=70044&viewType=3&cp=Call&month=6&year=2023&busDate=20230301

Hilber, N. (2023). *Bewertung von Finanzderivaten mit Python*. Springer Gabler Wiesbaden. https://doi.org/10.1007/978-3-658-39210-9

Leung, K. S. (2013). An analytic pricing formula for lookback options under stochastic volatility. *Applied Mathematics Letters*, *26*(1), 145–149. https://doi.org/10.1016/j.aml.2012.07.008

Refinitiv (2023). *Swiss Market Index (.SSMI) Price History, 01.01.2013-31.12.2022*. https://go.refinitiv.com/?u=Y3B1cmw6Ly9hcHBzLmNwLi9BcHBzL1ByaWNlSGlzdG9yeS8%2Fcz0uU1NNSSZzdD1SSUM%3D&title=.SSMI%20PH&key=VbUvk0ZFZ%2B%2FpH7UzagCmUoBP3RZnj7kGMQpIS3RVyYM%3D

Rouah, F. D. (2013). *The Heston Model and Its Extensions in Matlab and C#*. John Wiley & Sons.

Rouah, F. D. (2015). *The Heston Model and Its Extensions in VBA*. John Wiley & Sons.

Seydel, R. (2017). *Einführung in die numerische Berechnung von Finanzderivaten* (2nd ed.). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-50299-0

Wong, H. Y., & Kwok, Y. K. (2003). Sub-Replication and Replenishing Premium: Efficient Pricing of Multi-State Lookbacks. *Review of Derivatives Research*, *6*(2), 83–106. https://doi.org/10.1023/A:1027377228682

Zeng, X. C., Guo, I., & Zhu, S. P. (2017). Pricing european options on regime-switching assets: A comparison study of Monte Carlo and finite-difference approaches. *The ANZIAM Journal*, *59*(2), 183-199. https://doi.org/10.1017/S1446181117000335

Appendix

The appendix provides the relevant Python codes applied in the thesis, the underlying routines from Hilber (2023) as well as the graphic user interface and the codes for extensive graphs.

1. Generation of matrices comprising boundary conditions

matrixgenerator_BC.py (Hilber, 2023, p. 183-185)

```
import numpy as np
from scipy.sparse import spdiags

def matrixgenerator_BC(liste,BC,xl,xr,N,*args):
    '''Determines NxN matrices of the form M_y^(k) and M_y^(k,bc), k = 0,1,2.
    Example. [Mkj means the jth of the matrices M_y^(k)]

    Mat = matrixgenerator(liste,BC,xl,xr,N)

    with the list

    liste = [["M2",lambda x: x**2],["M2",lambda x: x],["M1",lambda x: x],
             ["M0",lambda x: 1]]

    gives the finite difference matrices M21 and M22 to x^2*u''(x) and
    x*u''(x), the matrix M11 to x*u'(x) as well as the matrix M01 to u(x) over
    the interval G = [xl,xr]. In addition, the corresponding matrices
    Mkjbc are outputs for the boundary conditions.

    matrixgenerator_BC(liste,BC,xl,xr,N,p) returns the same matrices for a
    lattice extension phi(x). Here is

    p = [phi_x(xl),phi_x(xr),phi_xx(xl),phi_xx(xr)]

    where phi_x(z) and phi_xx(z) are the first and second derivatives of the
    lattice stretch function phi(x) evaluated at the point x = z.

    matrixgenerator_BC(liste,BC,xl,xr,N,1,1,0,0) is the same as
    matrixgenerator_BC(liste,BC,xl,xr,N).

    The number of matrices in the output vector can be arbitrary but must be
    the same as the number of cells in the list.'''

    h = (xr-xl)/(N+1); x = np.linspace(xl-h,xr+h,N+4); nl,nr = BC;
    hd = lambda x: 0.5*x**2-1.5*x+1; hn = lambda x:-x**2+2*x;
    hs = lambda x: 0.5*(x**2-x)
    if len(args)>0:
        phipl,phipr,phippl,phippr = args;
```

```
    else:
        phipl = 1; phipr = 1; phippl = 0; phippr = 0; # no lattice stretching

    kp = phippl/phipl; km = phippr/phipr;
    ap = (10+4*h*kp)/(4+3*h*kp); am = (10-4*h*km)/(4-3*h*km);
    bp = (-8-h*kp)/(4+3*h*kp); bm = (-8+h*km)/(4-3*h*km);
    cp = 2/(4+3*h*kp); cm = 2/(4-3*h*km);

    U = [None]*len(liste)*2; count = 0

    # Creating matrices Mjk
    for j in range(len(liste)):
        count =count+1; v = liste[j]; y = v[1]
        if v[0]=="M2":
            U1 = 1/(h**2)*spdiags([y(x[2:N+5]),-2*y(x[1:N+3]),y(x[0:N+2])],
                    [-1,0,1],N+2,N+2).tolil()
            if nl==3:
                U1[0,0:4] = y(xl)/h**2*np.array([2,-5,4,-1])
            else:
                U1 = U1[1:,:]; U1 = U1[:,1:]
                U1[0,0:3] = y(x[2])/h**2*(hd(nl)*np.array([-2,1,0])+
                  hn(nl)*np.array([-2/3,2/3,0])+hs(nl)*np.array([ap-2,1+bp,cp]))
            if nr==3:
                U1[-1,-4:] = y(xr)/h**2*np.array([-1,4,-5,2])
            else:
                U1 = U1[:-1,:]; U1 = U1[:,:-1]
                U1[-1,-3:] = y(x[N+1])/h**2*(hd(nr)*np.array([0,1,-2])+
                  hn(nr)*np.array([0,2/3,-2/3])+hs(nr)*np.array([cm,1+bm,am-2]))

            U[j] = U1.todia()
        elif v[0]=="M1":
            U1 = 1/(2*h)*spdiags([-y(x[2:N+5]),np.zeros(N+2),y(x[0:N+2])],
                    [-1,0,1],N+2,N+2).tolil()
            if nl==3:
                U1[0,0:3] = y(xl)/(2*h)*np.array([-3,4,-1])
            else:
                U1 = U1[1:,:]; U1 = U1[:,1:]
                U1[0,0:3] = y(x[2])/(2*h)*(hd(nl)*np.array([0,1,0])+
                  hn(nl)*np.array([-4/3,4/3,0])+hs(nl)*np.array([-ap,1-bp,-cp]))
            if nr==3:
                U1[-1,-3:] = y(xr)/(2*h)*np.array([1,-4,3])
            else:
                U1 = U1[:-1,:]; U1 = U1[:,:-1]
                U1[-1,-3:] = y(x[N+1])/(2*h)*(hd(nr)*np.array([0,-1,0])+
                  hn(nr)*np.array([0,-4/3,4/3])+hs(nr)*np.array([cm,bm-1,am]))

            U[j] = U1.todia()

        else:
```

```
            U1 = spdiags(y(x[1:N+3]),[0],N+2,N+2).tolil();
            if nl<3: U1 = U1[1:,:]; U1 = U1[:,1:]
            if nr<3: U1 = U1[:-1,:]; U1 = U1[:,:-1]

            U[j] = U1.todia()

    # Creating boundary matrices Mkjbc
    for j in range(len(liste)):
        v = liste[j]; y = v[1]
        if v[0]=="M2":
            U1 = spdiags(np.zeros(N+2),0,N+2,N+2).tolil()
            if nl<3:
                U1 = U1[1:,:]; U1 = U1[:,1:]
                U1[0,0] = y(xl+h)*(hd(nl)/h**2+hn(nl)*(-2)/(3*h)*phipl\
                  +hs(nl)*phipl**2*cp)
            if nr<3:
                U1 = U1[:-1,:]; U1 = U1[:,:-1]
                U1[-1,-1] = y(xr-h)*(hd(nr)/h**2+hn(nr)*2/(3*h)*phipr\
                  +hs(nr)*phipr**2*cm)

            U[j+count] = U1.todia()

        elif v[0]=="M1":
            U1 = spdiags(np.zeros(N+2),0,N+2,N+2).tolil()
            if nl<3:
                U1 = U1[1:,:]; U1 = U1[:,1:]
                U1[0,0] = y(xl+h)*(-hd(nl)/(2*h)+hn(nl)/3*phipl\
                  +hs(nl)*(-h)/2*phipl**2*cp)
            if nr<3:
                U1 = U1[:-1,:]; U1 = U1[:,:-1]
                U1[-1,-1] = y(xr-h)*(hd(nr)/(2*h)+hn(nr)/3*phipr\
                  +hs(nr)*h/2*phipr**2*cm)

            U[j+count] = U1.todia()

        else:
            d = np.zeros(N); U1 = spdiags(d,0,N,N); U[j+count] = U1

    return U
```

2. Routine FDM

pde_2d_ah_theta.py (Hilber, 2023, p. 341)

```
import numpy as np
from scipy import sparse
from scipy.sparse.linalg import spsolve
from matrixgenerator_BC import matrixgenerator_BC
```

```
5
6  def pde_2d_ah_theta(a,b,c,T,g,G,BC,N,M,R,theta):
7      '''[x,y,w] = pde_2d_ah_theta() approximates the solution w(x,y,t) of the
8      partial differential equation
9
10     w_t + a1w_xx + a2w_yy + a3w_xy + b1w_x + b2w_y + cw = 0  in G x ]0,T]
11                                                       BC = 0
12                                                  w(.,0) = g
13
14     in the domain G = ]xl,xr[ x ]yl,yr[ for homogeneous boundary conditions.
15     The functions a1, a2, a3, b1, b2 and c can depend on x and y,
16     but must be the product of univariate functions, for example
17     a1(x,y) = a1x(x)a1y(y).'''
18
19     # Define functions
20     a1x,a1y,a2x,a2y,a3x,a3y = a; b1x,b1y,b2x,b2y = b; cx,cy = c;
21
22     # Define constants
23     xl,xr,yl,yr = G; nxl,nxr,nyl,nyr = BC;
24     hx = (xr-xl)/(N[0]+1); hy = (yr-yl)/(N[1]+1); k = T/M;
25
26     Matx = matrixgenerator_BC([["M2",a1x],["M1",a3x],["M1",b1x],["M0",a2x],
27                                ["M0",b2x],["M0",cx]],[nxl,nxr],xl,xr,N[0])
28     Maty = matrixgenerator_BC([["M2",a2y],["M1",a3y],["M1",b2y],["M0",a1y],
29                                ["M0",b1y],["M0",cy]],[nyl,nyr],yl,yr,N[1])
30
31     A = (sparse.kron(Maty[3],Matx[0])+sparse.kron(Maty[0],Matx[3])+
32          sparse.kron(Maty[1],Matx[1])+sparse.kron(Maty[4],Matx[2])+
33          sparse.kron(Maty[2],Matx[4])+sparse.kron(Maty[5],Matx[5]))
34
35     I = sparse.eye((N[0]+(nxr==3)+(nxl==3))*(N[1]+(nyr==3)+(nyl==3)))
36     B = I + k*theta*A; C = I - (1-theta)*k*A
37
38     # Define start vector w0 (exercise function)
39     x = np.linspace(xl+(1-(nxl==3))*hx,xr-(1-(nxr==3))*hx,\
40                     N[0]+(nxl==3)+(nxr==3))
41     y = np.linspace(yl+(1-(nyl==3))*hy,yr-(1-(nyr==3))*hy,\
42                     N[1]+(nyl==3)+(nyr==3))
43     x,y = np.meshgrid(x,y,indexing='ij'); w = g(x,y); w = w.flatten('F')
44
45     # Rannacher method
46     for j in range(R): w = spsolve(I+k/2*A,w)
47     for j in range(int(R/2),M): w = spsolve(B,C*w)
48
49     w = np.reshape(w,(N[0]+(nxl==3)+(nxr==3),N[1]+(nyl==3)+(nyr==3)),order='F')
50
51     return x,y,w
```

3. Routine lookback options

lookback_option_theta.py

```
import numpy as np
from pde_2d_ah_theta import pde_2d_ah_theta
from scipy.interpolate import interpn

# Input of calibrated parameter for lookback option
r = 0.03; q = 0.00; T = 1; s0 = 100; v = 0.0625; sigma = 0.25; kappa = 3;
rho = -0.5; theta_m = 0.05; theta_d = 0.5; omega = 1; K = 100; l = 1

# Definition of univariate functions and payoff function
a = [lambda x: x**0, lambda y: -0.5*y,
     lambda x: x**0, lambda y: -0.5*sigma**2*y,
     lambda x: x**0, lambda y: rho*sigma*y]
b = [lambda x: x**0, lambda y: (r-q+0.5*y),
     lambda x: x**0, lambda y: -(kappa*(theta_m-y)+rho*sigma*y)]
c = [lambda x: q*x**0, lambda y: y**0]
g = lambda x,y: omega*(l*np.exp(x)-1)*y**0
'''floating lookback call option - omega = -1
floating lookback put option - omega = 1'''

# Definition of grid, boundary conditions, node points, and time steps
G = [0,3,0.0,0.8]; BC = [1,1,3,1]; N = [250,250]; M = 100; R = 2

# Discretisation and generation of matrices
x,y,w = pde_2d_ah_theta(a,b,c,T,g,G,BC,N,M,R,theta_d)

# Calculation of floating and fixed option prices via put-call parity
if omega == -1:
    C_fl = interpn((x[:,0],y[0,:]), w*s0, (np.maximum(np.log(K/s0),0), v),
                   method='pchip', bounds_error=False, fill_value=None);
    print('floating Call:', C_fl)
    P_fix = C_fl-s0+np.exp(-(r-q)*T)*s0; print('fixed Put:', P_fix)
elif omega == 1:
    P_fl = interpn((x[:,0],y[0,:]), w*s0, (np.maximum(np.log(K/s0),0), v),
                   method='pchip', bounds_error=False, fill_value=None);
    print('floating Put:', P_fl)
    C_fix = P_fl+s0*np.exp(-q*T)-K*np.exp(-r*T); print('fixed Call:', C_fix)
else:
    print('Wrong value for Omega, enter -1 for call or 1 for put options.')
```

4. Creation of diagonal matrices

get_diagonals.py (Hilber, 2023, p. 655)

```
import numpy as np

def get_diagonals(M,*args):
    mp1 = np.hstack((0,M.diagonal(1))); m0 = M.diagonal(0);
    mm1 = np.hstack((M.diagonal(-1),0)); D = np.vstack((mp1,m0,mm1))

    if len(args)>0:
        nl = args[0]; nr = args[1]
        if nl>1: # upper
            for j in range(1,nl):
                #display(np.sum(np.abs(M.diagonal(j+1))))
                m = np.hstack((np.zeros(j+1),M.diagonal(j+1)));
                D = np.vstack((m,D))

        if nr>1: # lower
            for j in range(1,nr):
                #display(np.sum(np.abs(M.diagonal(-j-1))))
                m = np.hstack((M.diagonal(-j-1),np.zeros(j+1)));
                D = np.vstack((D,m))

    return D
```

5. Permutation matrix

perm_matrix.py (Hilber, 2023, p. 350)

```
import numpy as np; from scipy import sparse; from scipy.sparse import eye;
from scipy.sparse import vstack

def perm_matrix(n,m):
    '''Returns the (nm x nm) permutation matrix P'''
    Im = eye(m,m); x = np.zeros(n); x[0] = 1.0; P = sparse.kron(Im,x)

    for j in range(1,n):
        x = np.zeros(n); x[j] = 1.0; P = vstack([P,sparse.kron(Im,x)])
    return P
```

6. Routine FDM – Craig-Sneyd

pde_2d_ah_cs.py (Hilber, 2023, p. 350-351)

```
import numpy as np
from scipy import sparse
from scipy.linalg import solve_banded
from matrixgenerator_BC import matrixgenerator_BC
from get_diagonals import get_diagonals
from perm_matrix import perm_matrix

def pde_2d_ah_cs(a,b,c,T,g,G,BC,N,M,theta):
    '''Approximates the solution w(x,y,t) of the partial differential equation

    w_t+a1w_xx+a2w_yy+a3w_xy+b1w_x+b2w_y+(c1+c2)w = 0 in G x ]0 , T]
                                                BC = 0
                                            w(.,0) = g

    in the domain G = ]xl,xr[x]yl,yr[ for homogeneous boundary conditions
    via the Craig-Sneyd method .'''

    # Define functions
    a1x,a1y,a2x,a2y,a3x,a3y = a; b1x,b1y,b2x,b2y = b; c1x,c1y,c2x,c2y = c ;

    # Define constants
    xl,xr,yl,yr = G; nxl, nxr, nyl,nyr = BC ;
    hx = (xr-xl)/(N[0]+1); hy = (yr-yl)/(N[1]+1); k = T/M;
    beta = lambda n:1+n-(n>0); diagsx = (beta(nxr),beta(nxl))
    diagsy = (beta(nyr),beta(nyl))

    Matx = matrixgenerator_BC([["M",a1x],["M1",a3x],["M1",b1x],["M0",a2x],
                               ["M0",b2x],["M0",c1x],["M0",c2x]],[nxl,nxr],
                              xl,xr,N[0])

    Maty = matrixgenerator_BC([["M2",a2y],["M1",a3y],["M1",b2y],["M0",a1y],
                               ["M0",b1y],["M0",c1y],["M0",c2y]],[nyl,nyr],
                              yl,yr,N[1])

    A1 = (sparse.kron(Maty[3],Matx[0])+sparse.kron(Maty[4],Matx[2])+
          0.5*sparse.kron(Maty[5],Matx[5])+0.5*sparse.kron(Maty[6],Matx[6]))
    A2 = (sparse.kron(Maty[0],Matx[3])+sparse.kron(Maty[2],Matx[4])+
          0.5*sparse.kron(Maty[5],Matx[5])+0.5*sparse.kron(Maty[6],Matx[6]))
    A2t = (sparse.kron(Matx[3],Maty[0])+sparse.kron(Matx[4],Maty[2])+
           0.5*sparse.kron(Matx[5],Maty[5])+0.5*sparse.kron(Matx[6],Maty[6]))
    A0 = sparse.kron(Maty[1],Matx[1]); A = A1+A2+A0;

    I = sparse.eye((N[0]+(nxr==3)+(nxl==3))*(N[1]+(nyr==3)+(nyl==3)))
    B = I-k*A; C1 = k*theta*A1; C2 = k*theta*A2; C0 = 0.5*k*A0;
    B1 = get_diagonals(I+theta*k*A1,nxl,nxr);
```

```
    B2t = get_diagonals(I+theta*k*A2t,nyl,nyr)

    P = perm_matrix(N[1]+(nyr==3)+(nyl==3),N[0]+(nxr==3)+(nxl==3)); PT = P.T;

    # Define start vector w0 (exercise function)
    x = np.linspace(xl+(1-(nxl==3))*hx,xr-(1-(nxr==3))*hx,\
                        N[0]+(nxl==3)+(nxr==3))
    y = np.linspace(yl+(1-(nyl==3))*hy,yr-(1-(nyr==3))*hy,\
                        N[1]+(nyl==3)+(nyr==3))
    x,y = np.meshgrid(x,y,indexing ='ij'); w = g(x,y); w = w.flatten('F')

    # Craig-Sneyd method
    for j in range (M):
        aux1 = C1*w; aux2 = C2*w;
        y0 = B*w ; y1 = solve_banded(diagsx,B1,y0+aux1);
        y2 = solve_banded(diagsy,B2t,PT*(y1+aux2)); y2 = P*y2;
        z0 = y0-C0*(y2-w); z1 = solve_banded(diagsx,B1,z0+aux1);
        z2 = solve_banded(diagsy,B2t,PT*(z1+aux2)); w = P*z2;

    w = np.reshape(w,(N[0]+(nxl==3)+(nxr==3),N[1]+(nyl==3)+(nyr==3)),order ='F')
    return x,y,w
```

Remark: The univariate functions [c2x, c2y] of c are for the valuation of derivatives for interest rates and are not used in this thesis and are neutralised.

7. Routine lookback option – Craig-Sneyd

lookback_option_CS.py

```
import numpy as np
from pde_2d_ah_cs import pde_2d_ah_cs
from scipy.interpolate import interpn

# Input of calibrated parameter for lookback option
r = 0.03; q = 0.00; T = 1; s0 = 100; v = 0.0625; sigma = 0.25; kappa = 3;
rho = -0.5; theta_m = 0.05; theta_d = 0.5; omega = 1; K = 100; l = 1

# Definition of univariate functions and payoff function
a = [lambda x: x**0, lambda y: -0.5*y,
     lambda x: x**0, lambda y: -0.5*sigma**2*y,
     lambda x: x**0, lambda y: rho*sigma*y]
b = [lambda x: x**0, lambda y: (r-q+0.5*y),
     lambda x: x**0, lambda y: -(kappa*(theta_m-y)+rho*sigma*y)]
c = [lambda x: q*x**0, lambda y: y**0,
     lambda x: 0*x, lambda y: 0*y]
g = lambda x,y: omega*(l*np.exp(x)-1)*y**0
''' floating lookback call option - omega = -1
floating lookback put option - omega = 1'''
```

```
20
21 # Definition of grid, boundary conditions, node points, and time steps
22 G = [0,3,0.0,0.8]; BC = [1,1,3,1]; N = [1000,1000]; M = 100
23
24 # Discretisation and generation of matrices
25 x,y,w = pde_2d_ah_cs(a,b,c,T,g,G,BC,N,M,theta_d)
26
27 # Calculation of floating and fixed option prices via put-call parity
28 if omega == -1:
29     C_fl = interpn((x[:,0],y[0,:]), w*s0, (np.maximum(np.log(K/s0),0), v),
30                    method='pchip', bounds_error=False, fill_value=None);
31     print('floating Call:', C_fl)
32     P_fix = C_fl-s0+np.exp(-(r-q)*T)*s0; print('fixed Put:', P_fix)
33 elif omega == 1:
34     P_fl = interpn((x[:,0],y[0,:]), w*s0, (np.maximum(np.log(K/s0),0), v),
35                    method='pchip', bounds_error=False, fill_value=None);
36     print('floating Put:', P_fl)
37     C_fix = P_fl+s0*np.exp(-q*T)-K*np.exp(-r*T); print('fixed Call:', C_fix)
38 else:
39     print('Wrong value for Omega, enter -1 for call and 1 for put options.')
```

8. Graphical user interface

GUI_lookback_option.py

```
 1 import tkinter as tk
 2 from tkinter import ttk
 3 from tkinter import messagebox
 4 import numpy as np
 5 from pde_2d_ah_cs import pde_2d_ah_cs
 6 from scipy.interpolate import interpn
 7 import threading
 8
 9 root = tk.Tk() # Open loop
10 root.title('Pricing of Lookback Options') # Title of window
11 root.geometry('625x512') # Size of frame
12 root.resizable(width=False, height=False) # Frame fixed
13
14 # Set font for buttons
15 style = ttk.Style()
16 style.configure('TButton', font=('Arial', 14))
17
18 # Define basic operating mode
19 operating = 'No'
20
21 # Partial option dropdown menu
22 def p_drop(event):
23     selected_option = pdrop.get()
```

```
    if selected_option == 'Yes':
        entry_box.config(state='normal') # Activate the entry box
    else:
        entry_box.config(state='disabled') # Deactivate the entry box

# Enable simultaneous processes
def perform_b():
    thread = threading.Thread(target=perform_calculation)
    thread.start()

# Display 'Discretising...' and deactivate button while processing
def perform_calculation():
    global operating
    if operating == 'No':
        operating = 'Yes'
    else:
        operating = 'No'

    processing.grid(row=7, column=4, columnspan=2, padx=5, pady=5)
    calc_button.config(state='disabled')
    root.update()

    if operating == 'Yes':
        variables()

    processing.grid_remove()
    calc_button.config(state='normal')

# Assign inputs, discretise option value, and generate an error message
def variables():
    try:
        # Assign type of lookback option to process
        option_type = drop.get()
        o_entry = options[option_type]
        if o_entry == 1:
            o_entry = -1
        elif o_entry == 2:
            o_entry = 1
        elif o_entry == 3:
            o_entry = 1
        elif o_entry == 4:
            o_entry = -1

        # Define variables
        omega = float(o_entry)
        s0 = float(s0_entry.get())
        T = float(T_entry.get())
        r = float(r_entry.get())
        q = float(q_entry.get())
```

```
        v = float(v_entry.get())
        sigma = float(s_entry.get())
        kappa = float(k_entry.get())
        rho = float(rho_entry.get())
        theta_m = float(th_entry.get())
        K = float(K_entry.get())

        selected_option = pdrop.get()
        if selected_option == 'Yes':
            l = float(entry_box.get())
        else:
            l = 1.0

        # Definition of univariate functions and payoff function
        a = [lambda x: x**0, lambda y: -0.5*y,
             lambda x: x**0, lambda y: -0.5*sigma**2*y,
             lambda x: x**0, lambda y: rho*sigma*y]
        b = [lambda x: x**0, lambda y: (r-q+0.5*y),
             lambda x: x**0, lambda y: -(kappa*(theta_m-y)+rho*sigma*y)]
        c = [lambda x: q*x**0, lambda y: y**0,
             lambda x: 0*x, lambda y: 0*y]
        g = lambda x,y: omega*(l*np.exp(x)-1)*y**0

        # Definition of grid, boundary conditions, node points, and time steps
        if omega == 1:
            G = [0,3,0.0,0.8]; BC = [1,1,3,1]; N = [1000,1000]; M = 100
        elif omega == -1:
            G = [-3,0,0.0,0.8]; BC = [1,1,3,1]; N = [1000,1000]; M = 100

        # Discretisation and generation of matrices
        x,y,w = pde_2d_ah_cs(a,b,c,T,g,G,BC,N,M,0.5)

        # Calculate the specified option value
        if options[option_type] == 1:
            C_fl = interpn((x[:,0],y[0,:]),w*s0, (np.maximum(np.log(K/s0),0),v),
                           method='pchip', bounds_error=False, fill_value=None)
            C_fl_c = C_fl.item()
            V_value.set('{:.5f}'.format(C_fl_c))
        elif options[option_type] == 2:
            P_fl = interpn((x[:,0],y[0,:]),w*s0, (np.maximum(np.log(K/s0),0),v),
                           method='pchip', bounds_error=False, fill_value=None)
            P_fl_c = P_fl.item()
            V_value.set('{:.5f}'.format(P_fl_c))
        elif options[option_type] == 3:
            P_fl = interpn((x[:,0],y[0,:]),w*s0, (np.maximum(np.log(K/s0),0),v),
                           method='pchip', bounds_error=False, fill_value=None)
            C_fix = P_fl+s0*np.exp(-q*T)-K*np.exp(-r*T)
            C_fix_c = C_fix.item()
            V_value.set('{:.5f}'.format(C_fix_c))
```

```
122         elif options[option_type] == 4:
123             C_fl = interpn((x[:,0],y[0,:]),w*s0, (np.maximum(np.log(K/s0),0),v),
124                            method='pchip', bounds_error=False, fill_value=None)
125             P_fix = C_fl-s0+np.exp(-(r-q)*T)*s0
126             P_fix_c = P_fix.item()
127             V_value.set('{:.5f}'.format(P_fix_c))
128 
129     # Generate error for missing inputs
130     except ValueError:
131         messagebox.showerror('Error', 'Please insert values correctly.')
132     except TypeError:
133         messagebox.showerror('Error', 'Please insert values correctly.')
134 
135 # Define the initial value for the option price
136 V_value = tk.StringVar(value='0.00000')
137 
138 # Define variables
139 
140 # Omega (call/put)
141 o_text = ttk.Label(root, text='Select type of lookback option:',
142                    font=('Arial', 14))
143 o_text.grid(row=0, column=0, sticky='w', columnspan=3, padx=5, pady=10)
144 options = {'Please select option': None,
145            'floating lookback call option': 1,
146            'floating lookback put option':2,
147            'fixed lookback call option':3,
148            'fixed lookback put option':4}
149 drop = ttk.Combobox(root, values=list(options.keys()), font=('Arial', 14))
150 drop.current(0)
151 drop.grid(row=0, column=4)
152 
153 # Partial lookback option
154 l_text = ttk.Label(root, text='Partial lookback option:', font=('Arial', 14))
155 l_text.grid(row=1, column=0, sticky='w', columnspan=3, padx=5, pady=5)
156 pdrop = ttk.Combobox(root, values=['Please select option', 'Yes', 'No'],
157                      state='readonly', font=('Arial', 14))
158 pdrop.current(0)
159 pdrop.grid(row=1, column=4)
160 root.bind('<<ComboboxSelected>>', p_drop)
161 entry_box = ttk.Entry(root, width=7, font=('Arial', 14), justify='right',
162                       state='disabled')
163 entry_box.grid(row=1, column=5, padx=5, pady=5)
164 
165 # Initial stock price
166 s0_text = ttk.Label(root, text='S\u2080:', font=('Arial', 14))
167 s0_text.grid(row=2, column=0, sticky='w', padx=5, pady=5)
168 s0_entry = ttk.Entry(root, width=7, font=('Arial', 14), justify='right')
169 s0_entry.grid(row=2, column=1, padx=5, pady=5)
170 
```

```
171 # Time
172 T_text = ttk.Label(root, text='T:', font=('Arial', 14))
173 T_text.grid(row=3, column=0, sticky='w', padx=5, pady=5)
174 T_entry = ttk.Entry(root, width=7, font=('Arial', 14), justify='right')
175 T_entry.grid(row=3, column=1, padx=5, pady=5)
176
177 # Continuous interest rate
178 r_text = ttk.Label(root, text='r:', font=('Arial', 14))
179 r_text.grid(row=4, column=0, sticky='w', padx=5, pady=5)
180 r_entry = ttk.Entry(root, width=7, font=('Arial', 14), justify='right')
181 r_entry.grid(row=4, column=1, padx=5, pady=5)
182
183 # Continuous dividend yield
184 q_text = ttk.Label(root, text='q:', font=('Arial', 14))
185 q_text.grid(row=5, column=0, sticky='w', padx=5, pady=5)
186 q_entry = ttk.Entry(root, width=7, font=('Arial', 14), justify='right')
187 q_entry.grid(row=5, column=1, padx=5, pady=5)
188
189 # Variance
190 v_text = ttk.Label(root, text='v:', font=('Arial', 14))
191 v_text.grid(row=6, column=0, sticky='w', padx=5, pady=5)
192 v_entry = ttk.Entry(root, width=7, font=('Arial', 14), justify='right')
193 v_entry.grid(row=6, column=1, padx=5, pady=5)
194
195 # Sigma
196 s_text = ttk.Label(root, text='\u03c3:', font=('Arial', 14))
197 s_text.grid(row=7, column=0, sticky='w', padx=5, pady=5)
198 s_entry = ttk.Entry(root, width=7, font=('Arial', 14), justify='right')
199 s_entry.grid(row=7, column=1, padx=5, pady=5)
200
201 # Kappa
202 k_text = ttk.Label(root, text='\u03BA:', font=('Arial', 14))
203 k_text.grid(row=8, column=0, sticky='w', padx=5, pady=5)
204 k_entry = ttk.Entry(root, width=7, font=('Arial', 14), justify='right')
205 k_entry.grid(row=8, column=1, padx=5, pady=5)
206
207 # Rho
208 rho_text = ttk.Label(root, text='\u03C1:', font=('Arial', 14))
209 rho_text.grid(row=9, column=0, sticky='w', padx=5, pady=5)
210 rho_entry = ttk.Entry(root, width=7, font=('Arial', 14), justify='right')
211 rho_entry.grid(row=9, column=1, padx=5, pady=5)
212
213 # Theta of model
214 th_text = ttk.Label(root, text='\u03b8:', font=('Arial', 14))
215 th_text.grid(row=10, column=0, sticky='w', padx=5, pady=5)
216 th_entry = ttk.Entry(root, width=7, font=('Arial', 14), justify='right')
217 th_entry.grid(row=10, column=1, padx=5, pady=5)
218
219 # Strike
```

```
K_text = ttk.Label(root, text='K:', font=('Arial', 14))
K_text.grid(row=11, column=0, sticky='w', padx=5, pady=5)
K_entry = ttk.Entry(root, width=7, font=('Arial', 14), justify='right')
K_entry.grid(row=11, column=1, padx=5, pady=5)

# Create buttons and display option value
calc_button = ttk.Button(root, text='Discretise option price',
                         command=perform_b, width=30, state='normal')
calc_button.grid(row=10, column=4, columnspan=2, padx=5, pady=5)

V_text = ttk.Label(root, text='Option price', font=('Arial', 14))
V_text.grid(row=11, column=4, sticky='w', padx=5, pady=5)

display_V = ttk.Label(root, textvariable=V_value, font=('Arial', 14))
display_V.grid(row=11, column=5, padx=5, pady=5)

processing = ttk.Label(root, text = 'Discretising...', font=('Arial', 14))

quit_button = ttk.Button(root, text='Exit program', command=root.destroy)
quit_button.grid(row=12, column=4, columnspan=2, padx=5, pady=5, sticky='e')

# Close loop
root.mainloop()
```

9. SMI distributions

SMI_distribution.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st

# Import data from the Excel file
data = pd.read_excel(r'../Python/SMI_PnL.xlsx')
R = data['Log_r']

# Maximum likelihood estimation
dist_n = st.norm; args_n = dist_n.fit(R); dist_t = st.t; args_t = dist_t.fit(R)

# Define plot size and limits for x-axis
plt.figure(figsize=(8,6))
plt.xlim(-0.065,0.065)

# Create a histogram
n, bins, patches = plt.hist(R,bins=60,density=True,histtype='bar',color='w',
                            edgecolor='grey')
x = np.arange(bins[0],bins[-1],0.001)
```

```
21
22 # Visualise distributions
23 plt.plot(x,dist_n.pdf(x,*args_n),color='blue', alpha=0.75)
24 plt.plot(x,dist_t.pdf(x,*args_t),color='red', alpha=0.75)
25
26 # Labelling of axes
27 plt.xlabel('$x$', fontsize=14); plt.ylabel('$f(x)$',rotation=0,labelpad=15,
28                                             fontsize=14)
29 plt.xticks(fontsize=11); plt.yticks(fontsize=11)
30
31 # Labelling of distributions
32 plt.text(0.014,18,r'$\varphi_{\mu,\sigma}(x)$',fontsize=14);
33 plt.text(0.007,42,r'$f_{\mu,\sigma,\nu}(x)$',fontsize=14);
34
35 # Save plot
36 plt.savefig('SMI_distributions.png')
```

10. Volatility smile DAX

vola_smile_DAX.py

```
01 import numpy as np
02 import matplotlib.pyplot as plt
03 from impl_vola import impl_vola
04 from scipy.interpolate import UnivariateSpline
05
06 # Input data
07 VM = [3878.2,3492.6,3110.8,2733.9,2363.7,2002.3,1653.3,1321.3,1011.9,732.2,
08       492.1,301.9,167.2,83.6,39.5,17.9,8.0,3.0,1.2,0.5]
09 K = [11600,12000,12400,12800,13200,13600,14000,14400,14800,15200,15600,16000,
10      16400,16800,17200,17600,18000,18500,19000,19500]
11 s = 15305.02; T = 112/360; r = 0.02783;
12
13 # Compute implied volatility
14 sigma = np.zeros(len(VM))
15 for j in range(len(VM)):
16     sigma[j] = impl_vola(VM=VM[j],s=s,T=T,K=K[j],r=r,q=0,omega=1,init=0.3);
17
18 # Create and plot line
19 Kvec = np.linspace(0.99*min(K),1.01*max(K),200)
20 sigma_spline = UnivariateSpline(K,sigma,k=3,s=0)
21 plt.figure(figsize=(8,6))
22 plt.plot(K,sigma,'.',markersize=10,color = 'blue', alpha=0.75)
23 plt.plot(Kvec,sigma_spline(Kvec),color = 'blue', alpha=0.75 ,linewidth=1)
24
25 # Define labelling and size of labelling
26 plt.xticks(fontsize=11)
27 plt.yticks(fontsize=11)
```

```
28 plt.xlabel('$K$', fontsize=14, labelpad=5)
29 plt.ylabel(r'$\sigma^{\rm i}$', fontsize=14, rotation=0, labelpad=15);
30
31 # Save plot
32 plt.savefig('vola_smile_DAX.png')
```

impl_vola.py (Hilber, 2023, p. 18)

```
01: import numpy as np
02: import scipy.stats as ss
03:
04: def impl_vola(VM,s,K,T,r,q,omega,init):
05:     '''Calculates the implied volatility sigma of a call (omega = 1)
06:        or put (omega = -1) option with market price VM, exercise price K and
07:        maturity T using Newton's method starting in init.
08:        The underlying is s, r is the continuous interest rate,
09:        q the continuous yield.'''
10:
11:     # Tolerance tol and initial value for sigma
12:     tol = 10**-10; sigma0 = init/2; sigma1 = init;
13:
14:     # Iterate until tolerance is undercut
15:     while abs(sigma0-sigma1)>tol:
16:         sigma0 = sigma1
17:
18:         # option price of Black-Scholes
19:         d1 = (np.log(s/K)+(r-q+sigma0**2/2)*T)/(sigma0*np.sqrt(T))
20:         d2 = d1-sigma0*np.sqrt(T)
21:         V = omega*(np.exp(-q*T)*s*ss.norm.cdf(omega*d1) \
22:                    -K*np.exp(-r*T)*ss.norm.cdf(omega*d2))
23:
24:         # Vega of Black-Scholes
25:         dV = np.exp(-q*T)*s*np.sqrt(T)*ss.norm.pdf(d1)
26:         # One Newton-step
27:         sigma1 = sigma0 - (V-VM)/dV; #display(sigma1)
28:
29:     return sigma1
```

11. 2-dimensional FDM grid

grid_finite_differences.py

```
01 import matplotlib.pyplot as plt
02
03 # Create figure and axis objects
04 fig, ax = plt.subplots()
05
06 # Set x and y limits
07 ax.set_xlim([-0.05, 5.1])
08 ax.set_ylim([-0.05, 5.1])
09
10 # Set ticks and labels for x- and y-axis
11 ax.set_xticks(range(6))
12 ax.set_xticklabels(['$x\u2081$', '$x\u2082$', '$x\u2083$',
13                     '$x\u2084$', '$x\u2085$', '$x\u2086$'],
14                     fontsize=12)
15 ax.set_yticks(range(6))
16 ax.set_yticklabels(['$y\u2081$', '$y\u2082$', '$y\u2083$',
17                     '$y\u2084$', '$y\u2085$', '$y\u2086$'],
18                     fontsize=12)
19
20 # Create gridlines
21 ax.grid(True, which='both', color='black', linestyle='-',
22         linewidth=1)
23
24 # Create labels at every node point
25 for i in range(6):
26     for j in range(6):
27         ax.text(i+0.1, j+0.1, f'$w_{i+1},_{j+1}$', ha='left',
28                 va='bottom', fontsize=11)
29
30 # Add blue points to nodes
31 for i in range(6):
32     for j in range(6):
33         ax.scatter(i, j, s=20, c='blue', zorder=2)
34
35 # Remove the frame
36 for spine in ax.spines.values():
37     spine.set_visible(False)
38
39 # Add 'delta x'
40 ax.text(1.4,-0.32, '$\u0394x$', fontsize=12)
41 ax.text(-0.32,1.4, '$\u0394y$', fontsize=12);
42
43 # Save plot
44 plt.savefig('FDM_grid.png')
```

12. Domain of lookback options

domain_lookback.py

```python
import matplotlib.pyplot as plt
import numpy as np

# Create a range of x-values from 0 to 10
x = np.arange(0, 11)

# Set up a plot and remove frame lines
fig, ax = plt.subplots()
ax.axis('off')

# Plot the 45-degree line
ax.plot(x, x, color='black', linewidth=1)

# Fill the area above the line with blue
ax.fill_between(x, x, 10, where=x>=0, color='dodgerblue', alpha=0.3,
                interpolate=True)

# Fill the area below the line with green
ax.fill_between(x, x, 0, where=x<=10, color='limegreen', alpha=0.3,
                interpolate=True)

# Add arrows
ax.annotate('s', xy=(10, 0), xytext=(10, -0.5), ha='center', va='top',
            fontsize=14)
ax.annotate('s*', xy=(0, 10), xytext=(-0.5, 10), ha='right', va='center',
            fontsize=14)
ax.annotate('', xy=(10.25, 0), xytext=(0, 0),
            arrowprops=dict(arrowstyle='->'))
ax.annotate('', xy=(0, 10.25), xytext=(0, 0),
            arrowprops=dict(arrowstyle='->'))

# Add 0 for x-axis
ax.text(0, -0.5, '0', fontsize=14, ha='center', va='top')

# Add 0 for y-axis
ax.text(-0.5, 0, '0', fontsize=14, ha='right', va='center')

# Add labels for the coloured areas and the 45-degree line
ax.text(2, 6.5, 's \u2264 s* (Put)', fontsize=14)
ax.text(5.5, 3, 's \u2265 s* (Call)', fontsize=14)
ax.text(10.25, 10.25, 's = s*', fontsize=14);

# Save plot
plt.savefig('domain_lookback.png')
```