Numerical Methods for Pricing American Options with Continuous Dividend Yields Assets.

MATH4062

MSc Dissertation in

Financial and Computational Mathematics

2022/23

School of Mathematical Sciences

University of Nottingham

Alvin Jonel De la Cruz Guerrero

Supervisor: Dr. Anna Kalogirou

I have read and understood the School and University guidelines on plagiarism. I confirm that this work is my own, apart from the acknowledged references.

Abstract

In this work, we present numerical methods for pricing American options with continuous dividend yield asset. We focus on their implementation, convergence analysis, and performance. Our focus on two important formulation of the pricing problem: The free boundary and the linear complementary formulations. In the free boundary formulation, we consider the Black-Scholes PDE with one moving boundary condition. We consider two transformation-based approaches, proposed by Nielsen et al. and Company et al. to fix the moving boundary. Explicit and implicit schemes are developed for the Nielsen transformation, along with an explicit scheme for the Company transformation. In the linear complementary formulation, we consider the Black-Scholes PDE as variational inequalities that transform to linear complementary problem once finite difference schemes are applied. The PSOR is presented as an approach for solving the complementary problem arisen. Finally, several numerical experiments are conducted to evaluate convergence of the proposed schemes and discuss their relative performance.

Contents

1 Introduction

1.1 Overview

Options are equity-based derivatives that are primarily used to mitigate risk. The options market is significantly larger compared to other derivatives. In fact, options were the most traded derivatives in 2019, with a volume of 18.55 billion contracts when combining index and individual equities contracts [8]. An enormous market like the options market demands reliable pricing mechanisms to minimize arbitrage opportunities. Naturally, pricing American options is a broad research area because there is no closed-form solution to the PDE resulting from the Black-Scholes model. Merton [11] was the first to consider the Black-Scholes model proposed by Black and Scholes [1] to price European and American options. Although Merton derived a nice formula to price European options, he stated that, in general, a closed-form solution was not attainable for American options. In 1977, Schwartz [13] and Brennan [2] proposed using finite difference schemes to solve the pricing problem for American options. The work of Merton and Schwartz served as a foundation for the free boundary problem for American options pricing. The motivation behind the free boundary problem formulation is that for American options, there exists an optimal exercise price that marks the boundary between the region where exercising the option is profitable and the region where it is not. Moreover, this optimal exercise price changes with time, making it impossible for the holder to determine when to exercise. Based on the work of Landau [10], Wu et al. [16] formulated the front-fixing method as an approach to solve the free boundary problem for options, in which the Landau transformation is used to transform the optimal exercise price or the moving boundary into a fixed boundary. Multiple transformations have been proposed by Huang et al. [9], Nielsen et al. [12], and Company et al. [3]. Around the same period, Dewynne et al. [6] took a different approach to solve the pricing problem. The idea was to reformulate the free boundary problem as a problem with variational inequalities that, when finite difference is applied, transforms into a linear complementary problem [4].

1.2 Aim

The goal of this work is to implement the numerical methods proposed by Nielsen et al. [12] and Company et al. [3] to solve the free boundary formulation of the pricing problem for American options [6]. Moreover, Company et al. [3] and Nielsen et al. [12] proposed schemes for pricing American put options with underlying assets that have non-paying dividends. However, we aim to derive analogous schemes for pricing call contracts as well, considering underlying assets that have a continuous dividend yield, such as the S&P500. Additionally, we want to conduct a convergence analysis of each of the methods. Furthermore, we consider the linear complementary formulation proposed by Dewynne [6] [15]. Finally, we perform a convergence analysis of each of the methods implemented.

1.3 Main Achievements

In this work, we were able to derive the corresponding PDE problem resulting from applying the transformations proposed by Nielsen et al. [12] and Company et al. [3] for call and put options with underlying assets featuring a continuous dividend yield. Moreover, we implemented explicit and implicit schemes for Nielsen et al.'s method [12], an explicit scheme for Company et al.'s method [3], and the theta method for the linear complementary problem proposed by [15] using Python. Finally, we derived the order of convergence for each of the implemented methods.

1.4 Outline

The outline of this paper is as follows. In Section 2, we explore the Black-Scholes model for American options for assets that pay dividends, resulting in the free boundary formulation of the pricing problem. Furthermore, we delve into the front-fixing method as a strategy for fixing the moving boundary by applying a change of variable. We consider the changes of variables proposed by Company et al. [3] and Nielsen et al. [12]. In Section 3, we explore explicit and implicit schemes to solve the partial differential equations resulting from applying the front-fixing method and the Nielsen transformation to the free boundary problem obtained

in Section 1. We conclude Section 3 with numerical experiments and convergence analysis for the numerical schemes presented based on the work of Nielsen and Company (see Appendix A). In Section 5, we explore a reformulation of the pricing problem as a variational inequality and the linear complementary system of equations resulting from it. Additionally, we delve into the theta method, which is a more general numerical method that yields explicit, implicit, and Crank-Nicholson schemes. Finally, in the same section, we discuss the results of solving the linear complementary problem reformulation of pricing American options using the theta method.

2 Free boundary problem

2.1 Overview

A common problem in finance is pricing financial derivatives, often referred to simply as derivatives. In essence, derivatives are contracts set between parties whose value over time derives from the price of their underlying assets. A notorious family of derivatives in financial markets is "options." Options are contracts set between two parties in which the holder has the right to sell or buy—an action commonly referred to as exercising—an underlying asset at a pre-established price—also known as the strike price—in the future. Options are referred to as "call options" or "put options" if the exercise position is to buy or to sell, respectively. Similarly, options are classified depending on their exercise style. In that regard, the simplest options are European options. European options give the right to exercise on the expiration date of the contract. Another well-known type of option is the American option. American options work similarly to European options, with the difference that they can be exercised at any point in time between the beginning and the expiration date of the contract. Obviously, American and European options are almost identical, differing only in the times at which the holder can exercise them. Therefore, we will start by describing the pricing problem from the European options perspective and then extend it to the case of American options.
Let us define the payoff function of European option as

$$\textbf{Call:} \qquad H_{\mathsf{Eur}}(S) = \max(S - K, 0) \tag{2.1a}$$

$$\textbf{Put:} \qquad H_{\mathsf{Eur}}(S) = \max(K - S, 0) \tag{2.1b}$$

where $S \in [0, \infty]$ is the asset price at the maturity date and K is strike price. Note that the strike price remains constant during the lifespan of the option. We can extend the European payoff to American options by introducing the time axis to equation above.

$$\textbf{Call:} \qquad H(S, t) = \max(S - K, 0) \tag{2.2a}$$

$$\textbf{Put:} \qquad H(S, t) = \max(K - S, 0) \tag{2.2b}$$

The interval $t \in [0, T]$ is the time elapsed since the beginning of the contract, measure in years. Clearly, $t = 0$ and $t = T$ mark starting date and the expiration date of the option. While the payoff of European options is defined only at $t = T$, American options' payoff is defined for all $(S, t) \in [0, \infty] \times [0, T]$.

Options provide greater flexibility to holders by eliminating their exposure to negative payoffs. Therefore, the writer of the option charges premiums to the holders for the right of entering the contract. The premium is often referred to as the price or value of the option, and the problem of determining this value is called option pricing. Let V_t represent the value of the option at time t. For instance, V_0 represents the value or premium. When pricing options, it is crucial to find the fair premium V_0; otherwise, the writer or holder of the option could devise a scheme in which the option will always be profitable to them. In other words, options pricing must adhere to the principle of no-arbitrage. Therefore, we assume that the writer of the option uses the premium to construct a portfolio consisting of ϕ_0 units of the a risky asset S_t and invests ψ_0 units of cash in a risk-free asset B_t, such as a US Treasury bills, certificates of deposit, or a bank account. Then, the writer rebalances the portfolio (ϕ_0, ψ_0) to hedge against any potential claims from the holder at any future time $0 < t \leq T$. Consequently, at any time t, the writer holds a portfolio (ϕ_t, ψ_t) with a value

$$\Pi_t = \phi_t S_t + \psi_t B_t$$

Moreover, the portfolio is self-financing. In other words, the changes in portfolio depend only on the changes in S_t and B_t, and the rebalancing of portfolio (ϕ_t, ψ_t)

$$d\Pi_t = \phi_t dS_t + \psi_t dB_t$$

$$S_t d\phi_t + B_t d\psi_t = 0$$

Finally, the portfolio value matches the option value

$$\Pi_t = V_t$$

at any time $0 \leq t \leq T$. Using the self-financing portfolio hedging strategy, Black et al.[1] presented a mathematical model for the dynamics of the price of European and American options. While the model makes several assumptions about the market[11], we enumerate just a few of them. Firstly, the asset price S_t is distributed log-normally

$$S_t = S_0 \exp\left\{ \int_0^t \left(r - \frac{1}{2}\sigma\right)ds + \sqrt{t}Z \right\} \tag{2.3}$$

where the risk-free interest rate r and the price volatility σ remain constant time during the life of the option. Secondly, the bank account $B(t)$ is a deterministic function

$$dB = rB(t)dt$$

Finally, the underlying asset does not pay dividends. By applying the Black-Scholes model to price European options, Merton [11] obtained the famous Black-Scholes PDE

$$\begin{cases} \dfrac{\partial V}{\partial t} + \dfrac{1}{2}\sigma^2 S^2 \dfrac{\partial^2 V}{\partial S^2} + rS\dfrac{\partial V}{\partial S} - rV = 0 & \text{for } t \in [0,T) \text{ and } S \in [0,\infty) \\ V(S,T) = H(S,T) & \text{for } S \in [0,\infty) \end{cases}$$

where $V(S,t)$ is a deterministic function. We previously mentioned that Black-Scholes model assumes that the underlying asset does not pay dividends. In most cases, assets such as stocks pay out dividends just a few times at year. In this case, dividends are to be modelled discretely. However, there are certain assets that pay out a proportion of the current price during and interval of time. For instance, indexes such as the SPX. Thus, in such cases, it is useful to model dividends as a continuous yield. Dewynne et al.[15] shows a slight adjusted asset price model that includes continuous yield dividends

$$S_t = S_0 \exp\left\{ \int_0^t \left(r - \delta(t) - \frac{1}{2}\sigma\right)ds + \sqrt{t}Z \right\} \tag{2.4}$$

Note that when the asset does not pay out dividends $\delta = 0$, the asset price model will be exactly as in (2.3). Similar to the constant interest rate and volatility assumptions, continuous dividend yield δ is assumed to remain constant. As a consequence of the asset price mode

(2.4), the Black-Scholes PDE changes to

$$\begin{cases} \dfrac{\partial V}{\partial t} + \mathcal{L}_{\mathsf{BS}}(V) = 0 & \text{for } t \in [0, T) \text{ and } S \in [0, \infty) \\ V(S, T) = H(S, T) & \text{for } S \in [0, \infty) \end{cases} \tag{2.5}$$

where $\mathcal{L}_{\mathsf{BS}}(\cdot)$ is the linear parabolic operator applied to the function $V \in \mathcal{C}^2$

$$\mathcal{L}_{\mathsf{BS}}(V) := \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r - \delta) S \frac{\partial V}{\partial S} - rV \tag{2.6}$$

Similarly, to the asset price model with dividends, the Black-Scholes PDE with dividends fall back to the original Black-Scholes PDE if the asset does not pay out dividends $\delta = 0$. Likewise, by considering the Black-Scholes model to price American options, Merton [11] derived some important facts about the value function $V(S, t)$. Firstly, that the $V(S, t)$ is bounded from below by the payoff function

$$V(S, t) \geq H(S, t) \qquad \text{for } t \in [0, T] \tag{2.7}$$

Moreover, that the domain of $V(S, t)$ can be separated into the exercise region in

$$\mathcal{S} := \{(S, t) : V(S, t) = H(S, t)\} \tag{2.8}$$

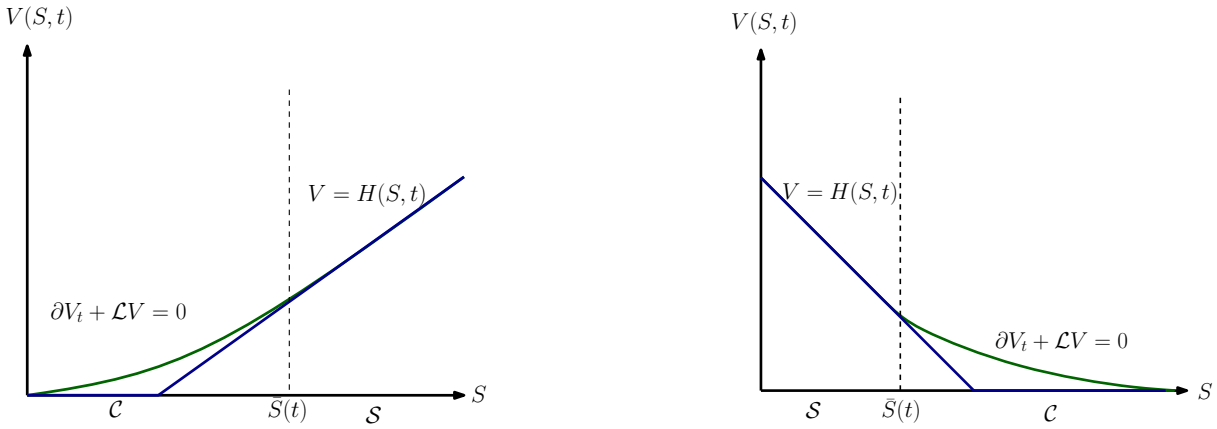
in which it is profitable for the holder to exercise the option, the continuation region

$$\mathcal{C} := \{(S, t) : V(S, t) > H(S, t)\} \tag{2.9}$$

in which it is preferable to continue holding the option because exercising is not profitable, the optimal exercise boundary that separates the continuation region and exercise region

$$\partial\mathcal{C} := \{(S, t) : S = \bar{S}(t)\} \tag{2.10}$$

where $\bar{S}(t)$ is the optimal exercise price.



(a) Call option (b) Put option

Figure 2.1: Value $V(S,t)$ of American option value curve.

and, lastly, that the price dynamics of American options is governed by the same Black-Scholes PDE as European options in the continuation region. Therefore, Pricing American options reduces to solve (2.1) with a boundary condition at the optimal exercise price $\bar{S}(t)$.

$$\begin{cases} \dfrac{\partial V}{\partial t} + \dfrac{1}{2}\sigma^2 S^2 \dfrac{\partial^2 V}{\partial S^2} + (r-\delta)S\dfrac{\partial V}{\partial S} - rV = 0 & \text{for } (S,t) \in \mathcal{C} \\ V(S,t) = H(S,t) & \text{for } (S,t) \in \partial\mathcal{C} \end{cases} \tag{2.11}$$

The problem above is also known as the free boundary formulation of the pricing problem because it requires to solve a PDE with a moving boundary. Clearly, the optimal exercise price $\bar{S}(t)$ is within the exercise region (2.8). Moreover, the boundary condition opposite the moving boundary are

$$\textbf{Call:} \quad V(0,t) = 0, \quad V(\bar{S}(t),t) = \bar{S}(t) - K \tag{2.12a}$$

$$\textbf{Put:} \quad V(\bar{S},t) = K - \bar{S}(t), \quad \lim_{\infty} V(S,t) = 0 \tag{2.12b}$$

Additionally, it can be observed in figure (2.1) that $V(S,t)$ touches the payoff $H(S,t)$ tangentially at the optimal exercise price $\bar{S}(t)$

$$\textbf{Call:} \quad \frac{\partial V}{\partial S}(\bar{S}(t),t) = 1 \tag{2.13a}$$

$$\textbf{Put:} \quad \frac{\partial V}{\partial S}(\bar{S}(t),t) = -1 \tag{2.13b}$$

This extra condition is called the contact point or smooth pasting condition and later on will serve handy in approximating $\bar{S}(t)$. Finally, the terminal condition of the PDE will be given at $t = T$. At the expiration date of the contract, the holder will either exercise or not the option. Therefore, the value of the option will be equal to the payoff function. Obviously, in that case, the optimal exercise price $S(T)$ will be equal to the strike price K. Hence,

$$V(S,T) = H(S,T), \qquad \bar{S}(T) = K \tag{2.14}$$

By grouping (2.11), (2.13), and (2.14) in one equation, a system for the free boundary problem is obtained.

Call:
$$\begin{cases} \dfrac{\partial V}{\partial t} + \dfrac{1}{2}\sigma^2 S^2 \dfrac{\partial^2 V}{\partial S^2} + (r-\delta)S\dfrac{\partial V}{\partial S} - rV = 0 \quad \text{for } S \in (0, \bar{S}(t)) \text{ and } t \in [0,T) \\ V(S,T) = S - K \\ \bar{S}(T) = K \\ V(0,t) = 0 \\ \dfrac{\partial V}{\partial S}(\bar{S}(t),t) = 1 \end{cases} \tag{2.15a}$$

Put:
$$\begin{cases} \dfrac{\partial V}{\partial t} + \dfrac{1}{2}\sigma^2 S^2 \dfrac{\partial^2 V}{\partial S^2} + (r-\delta)S\dfrac{\partial V}{\partial S} - rV = 0 \quad \text{for } S \in (\bar{S}(t), \infty), \text{ and } t \in [0,T) \\ V(S,T) = K - S \\ \bar{S}(T) = K \\ \lim_{S\to\infty} V(S,t) = 0 \\ \dfrac{\partial V}{\partial S}(\bar{S}(t),t) = -1 \end{cases} \tag{2.15b}$$

2.2 Front-Fixing method

In the previous section, we presented the pricing of American options problem. By applying the Black-Scholes model, we derived the Black-Scholes PDE that describes the price dynamics in the continuation region $\mathcal{C}$ of call and put options. Moreover, we presented the moving

boundary condition $\bar{S}(t)$ for this PDE. The moving boundary condition $\bar{S}(t)$ makes the Black-Scholes PDE more involved since we also need to determine this boundary as time changes. This type of problems are known as free boundary problems. The front fixing method is a strategy in which a transformation is used to map the domain from the original problem to a new domain where moving boundary remains fixed as time changes. In this section, we explore two transformation based on the work of Nielsen et al. [12], and the work of Company and et al. [3].

2.2.1 Nielsen transformation

The Nielsen transformation suggests a really simple transformation in which the asset price S is divided by the optimal exercise price $\bar{S}$

$$x = \frac{S}{\bar{S}(t)} \tag{2.16}$$

Clearly, the moving boundary in the original problem will be fixed when $S = \bar{S}(t)$ at $x = 1$. Now, we define $v(x,t)$ as the value function of the option but under the front fixing domain given by x

$$v(x,t) := V(S,t) \tag{2.17}$$

Moreover, we want to understand how this transformation affects the Black-Scholes PDE, the boundary, terminal and contact point conditions given in equation in (2.15).

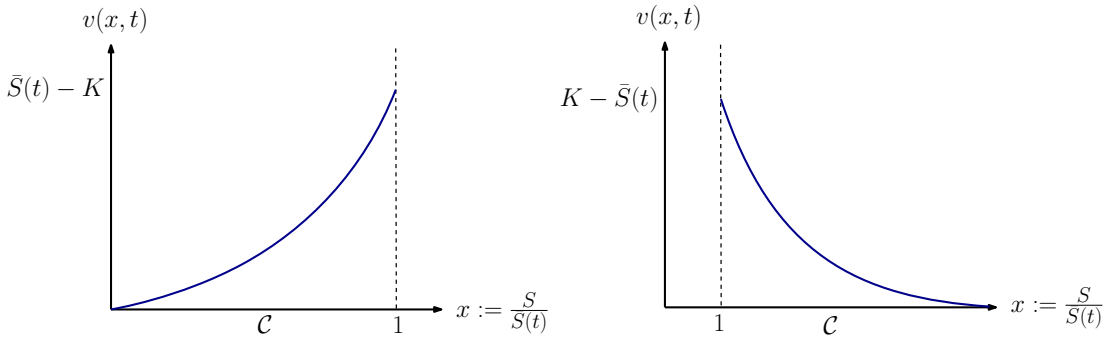
Firstly, we start with the Black-Scholes PDE which is defined at the interval $S \in (0, \bar{S}(t))$ for call options or the open interval $S \in (\bar{S}(t), \infty)$ for put options. Under the front fixing domain, the transformed PDE will be defined in the interval $x \in (0,1)$ for call and $x \in (1,\infty)$ for put. Moreover, we apply the chain rule to rewrite the Black-Scholes PDE in terms of v, so that,

the new PDE is given as

$$\textbf{Call:} \quad \frac{\partial v}{\partial t} + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 v}{\partial x^2} + \left[(r-\delta) - \frac{\bar{S}'(t)}{\bar{S}(t)}\right] x \frac{\partial v}{\partial x} - rv = 0 \quad \text{for } x \in [0,1) \text{ and } t \in [0,T) \tag{2.18a}$$

$$\textbf{Put:} \quad \frac{\partial v}{\partial t} + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 v}{\partial x^2} + \left[(r-\delta) - \frac{\bar{S}'(t)}{\bar{S}(t)}\right] x \frac{\partial v}{\partial x} - rv = 0 \quad \text{for } x > 1 \text{ and } t \in (0,T] \tag{2.18b}$$

Similarly, we express the boundary conditions in the front fixing domain. We already stated that the Nielsen transformation fixes the moving boundary $\bar{S}$ at $x = 1$. Additionally, x goes to infinity as S goes to infinity, and $x = 0$ for $S = 0$. Therefore, the boundary condition opposite to the optimal exercise price $\bar{S}(t)$ will remain as in the original problem. Hence, as it can be observed in figure (2.2), the call option has left boundary condition $v(0,t) = 0$ at $x = 0$ and right boundary condition $v(1,t) = \bar{S}(t) - K$ at $x = 1$. Alternatively, the put option has left boundary condition $v(1,t) = K - \bar{S}(t)$ at $x = 1$ and right boundary condition $v(x,t) = 0$ at a sufficiently large x.



(a) Call option (b) Put option

Figure 2.2: Value $v(x,t) := V(S,t)$ in the front fixing domain defined by Nielsen transformation.

Likewise, we express the contact point condition in terms of $v(x,t)$. Recall that at the contact point, the slope $V(S,t)$ with respect to S is the same as the slope of the linear segment in the payoff function. This can be seen clearly in figure (2.1). Hence, by the chain rule, the

contact point condition of $v(x,t)$ is given by

$$\textbf{Call:} \qquad \frac{\partial v}{\partial x}(1,t) = \bar{S}(t) \tag{2.19a}$$

$$\textbf{Put:} \qquad \frac{\partial v}{\partial x}(1,t) = -\bar{S}(t) \tag{2.19b}$$

Finally, recall that the terminal condition of $\bar{S}(t)$ is given by (2.14). Moreover, $x >= 1$ for call options, and $x <= 1$ for put options. Hence, by simple substitution, we can rewrite the terminal conditions of $v(x,t)$ as

$$\textbf{Call:} \qquad v(x,T) = \max(x\bar{S}(T) - K) = K\max(x-1,0) = 0 \tag{2.20a}$$

$$\textbf{Put:} \qquad v(x,T) = \max(K - x\bar{S}(T)) = K\max(1-x,0) = 0 \tag{2.20b}$$

In summary, by groping equations (2.18), (2.20), and (2.19), we obtain the system

Call:
$$\begin{cases} \dfrac{\partial v}{\partial t} + \dfrac{1}{2}\sigma^2 x^2 \dfrac{\partial^2 v}{\partial x^2} + \left[(r-\delta) - \dfrac{\bar{S}'(t)}{\bar{S}(t)}\right] x \dfrac{\partial v}{\partial x} - rv = 0 & \text{for } x \in (0,1) \text{ and } t \in [0,T) \\ v(x,T) = 0 & \text{for } x \in [0,1] \\ \bar{S}(T) = K & \\ v(0,t) = 0 & \text{for } t \in [0,T) \\ v(1,t) = \bar{S}(t) - K & \text{for } t \in [0,T) \\ \dfrac{\partial v}{\partial x}(1,t) = \bar{S}(t) & \text{for } t \in [0,T) \end{cases} \tag{2.21a}$$

Put:
$$\begin{cases} \dfrac{\partial v}{\partial t} + \dfrac{1}{2}\sigma^2 x^2 \dfrac{\partial^2 v}{\partial x^2} + \left[(r-\delta) - \dfrac{\bar{S}'(t)}{\bar{S}(t)}\right] x \dfrac{\partial v}{\partial x} - rv = 0 & \text{for } x > 1 \text{ and } t \in [0,T) \\ v(x,T) = 0 & \text{for } x \geq 1 \\ \bar{S}(T) = K & \\ v(1,t) = K - \bar{S}(t) & \text{for } t \in [0,T) \\ \lim_{x\to\infty} v(x,t) = 0 & \text{for } t \in [0,T) \\ \dfrac{\partial v}{\partial x}(1,t) = -\bar{S}(t) & \text{for } t \in [0,T) \end{cases} \tag{2.21b}$$

2.2.2 Company transformation

The Company transformation proposes set of change of variable for the asset price S, the time t, the value function $V(S,t)$ and the moving boundary

$$x := \log \frac{S}{\bar{S}_f(t)}, \quad \tau := T - t, \quad v(x,\tau) := \frac{V(S,t)}{K}, \quad \bar{S}_f(\tau) := \frac{\bar{S}(t)}{K} \tag{2.22}$$

Let us break down the transformations. Firstly, the transformation proposed are written forward in time. Therefore, $\tau = 0$ refers to the expiration date of the options $t = T$. Secondly, both the value function and the optimal exercise price is scaled by the strike price. Finally, the new moving boundary is fixed at $S = \bar{S}_f(t)$ or $x = 0$.

Similarly, as we did for the Nielsen method, we rewrite the Black-Scholes PDE in terms of $v(x,\tau)$. Note that as x goes to infinity S goes to infinity. Conversely, as x goes to negative infinity S goes to zero. Moreover, $S = \bar{S}(t)$ at $x = 0$. Using the previous information, we deduce that the Black-Scholes PDE is defined in the intervals $x \in (-\infty, 0)$ for call options and $x \in (0, \infty)$ for put options. Therefore, we have

$$\textbf{Call:} \quad \frac{\partial v}{\partial \tau} - \frac{1}{2}\sigma^2 \frac{\partial^2 v}{\partial x^2} - \left((r-\delta) + \frac{\sigma^2}{2} - \frac{\bar{S}'(\tau)}{\bar{S}(\tau)}\right)\frac{\partial v}{\partial x} + rv = 0 \quad \text{for } x < 0 \text{ and } \tau \in (0, T] \tag{2.23a}$$

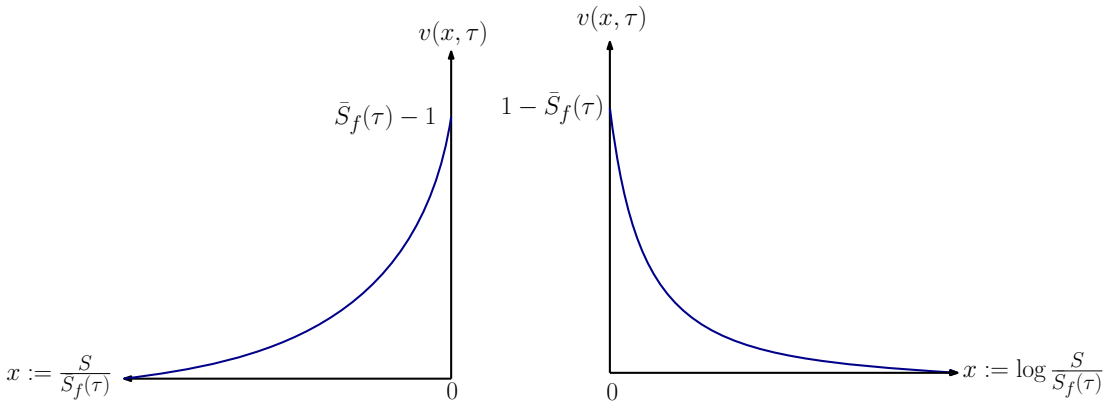
$$\textbf{Put:} \quad \frac{\partial v}{\partial \tau} - \frac{1}{2}\sigma^2 \frac{\partial^2 v}{\partial x^2} - \left((r-\delta) - \frac{\sigma^2}{2} - \frac{\bar{S}'(\tau)}{\bar{S}(\tau)}\right)\frac{\partial v}{\partial x} + rv = 0 \quad \text{for } x > 0 \text{ and } \tau \in (0, T] \tag{2.23b}$$

Note that the term on the new PDE that correspond to the terms in the linear parabolic operator $\mathcal{L}V$ defined in (2.6) are negative because the Black-Scholes PDE was inverted in time.

Again, the boundary conditions for the call option in the original domain are $V(0,t)$ at $S = 0$ and $V(\bar{S},t) = \bar{S} - K$ at $S = \bar{S}(t)$, and when transforming those boundary conditions to the front fixing domain, they become $v(x,\tau) = 0$ for a sufficiently negative x and $v(0,\tau) := \bar{S}_f(\tau) - 1 = V(\bar{S},t)/K$ at $x = 0$. Similarly, the boundary conditions for the put option in the original domain are $V(\bar{S}(t),t) = K - \bar{S}$ at $S = \bar{S}(t)$ and $V(S,t) = 0$ for a sufficiently large S, and under the front fixing domain, they become $v(0,\tau) = 1 - \bar{S}_f(\tau) = V(\bar{S},t)/K$ at $x = 0$ and $v(x,\tau) = 0$ for a sufficiently large x. Similarly, to as we did for the Nielsen transformation, we also rewrite the contact point condition

$$\textbf{Call:} \quad \frac{\partial v}{\partial x}(0,\tau) = \bar{S}_f(\tau) \tag{2.24a}$$

$$\textbf{Put:} \quad \frac{\partial v}{\partial x}(0,\tau) = -\bar{S}_f(\tau) \tag{2.24b}$$



(a) Call option (b) Put option

Figure 2.3: Value $v(x,t) := V(S,t)/K$ in the front fixing domain defined by Company transformation.

Since the transformed PDE is forward in time, we have to come up with initial conditions for $\bar{S}_f(\tau)$ and $v(x,\tau)$. For $\bar{S}_f(\tau)$, the initial condition is given by

$$\bar{S}_f(0) = \frac{\bar{S}(T)}{K} = 1 \tag{2.25}$$

Moreover, for call options, the initial condition is given by $v(x,0) = V(S,T)/K = \max\left(\bar{S}_f(0)e^x - 1, 0\right) = \max\left(e^x - 1, 0\right) = 0$ since x is always negative. Similarly, for put options, the initial condition is given by $v(x,0) = V(S,T)/K = \max\left(1 - \bar{S}_f(0)e^x, 0\right) = \max\left(1 - e^x, 0\right) = 0$ since x is always positive. Hence,

$$\textbf{Call:} \qquad v(x,0) = 0 \tag{2.26a}$$

$$\textbf{Put:} \qquad v(x,0) = 0 \tag{2.26b}$$

Finally, grouping the equations together, we have the system

Call:

$$\begin{cases} \dfrac{\partial v}{\partial \tau} - \dfrac{1}{2}\sigma^2 \dfrac{\partial^2 v}{\partial x^2} - \left((r-\delta) + \dfrac{\sigma^2}{2} - \dfrac{\bar{S}'(\tau)}{\bar{S}(\tau)} \right) \dfrac{\partial v}{\partial x} + rv = 0 & \text{for } x < 0 \text{ and } \tau \in (0, T] \\ v(x, 0) = 0 & \text{for } x < 0 \\ \bar{S}_f(0) = 1 & \\ \lim_{x \to -\infty} v(x, \tau) = 0 & \text{for } \tau \in (0, T] \\ \dfrac{\partial v}{\partial x}(0, \tau) = \bar{S}_f(\tau) & \text{for } \tau \in (0, T] \end{cases} \tag{2.27a}$$

Put:

$$\begin{cases} \dfrac{\partial v}{\partial \tau} - \dfrac{1}{2}\sigma^2 \dfrac{\partial^2 v}{\partial x^2} - \left((r-\delta) - \dfrac{\sigma^2}{2} - \dfrac{\bar{S}'(\tau)}{\bar{S}(\tau)} \right) \dfrac{\partial v}{\partial x} + rv = 0 & \text{for } x > 0 \text{ and } \tau \in (0, T] \\ v(x, 0) = 0 & \text{for } x > 0 \\ \bar{S}_f(0) = 1 & \\ \lim_{x \to \infty} v(x, \tau) = 0 & \text{for } \tau \in (0, T] \\ \dfrac{\partial v}{\partial x}(0, \tau) = -\bar{S}_f(\tau) & \text{for } \tau \in (0, T] \end{cases} \tag{2.27b}$$

3 Finite difference schemes

3.1 Overview

In this section, we present explicit and implicit central finite difference schemes for solving the PDE problem in (2.21). Previously, we considered the pricing problem of American options which requires solving the free boundary problem defined in (2.15). Then, we presented the front fixing method as a strategy to fix the moving boundary using a change of variable. Moreover, we derived the PDE problem for call and put options that resulted from applying the Nielsen transformation suggested by [12], and the Company transformation suggested by [3], resulting in the systems (2.21) and (2.27), respectively. In the following part, we present numerical methods for solving (2.21). But before we jump into that, we define what it means to compute a numerical solution to a PDE problem.

Recall that the solution $v(x,t)$ of (2.21) is defined in the continuous region

$$\textbf{Call:} \qquad \mathcal{T} : [0,T], \qquad \mathcal{X} : [0,1], \qquad \mathcal{F} : \mathcal{X} \times \mathcal{T} \tag{3.1a}$$

$$\textbf{Put:} \qquad \mathcal{T} : [0,T], \qquad \mathcal{X} : [1,\infty), \qquad \mathcal{F} : \mathcal{X} \times \mathcal{T}, \tag{3.1b}$$

Now, we want to discretize $\mathcal{F}$ using the grid $\mathcal{G}$ with $N+1$ and $M+1$ nodes

$$\mathcal{G} := \{(x_i, t_n) : (i,n) \in \{0,\dots,M+1\} \times \{0,\dots,N+1\}\} \tag{3.2}$$

where

$$x_i := x_{\text{min}} + i\Delta x \qquad \text{for } i = 0,\dots,M+1 \tag{3.3}$$

$$t_n := t_{\text{min}} + i\Delta t \qquad \text{for } i = 0,\dots,N+1 \tag{3.4}$$

$$\Delta x := \frac{x_{\text{max}} - x_{\text{min}}}{M+1} \tag{3.5}$$

$$\Delta t := \frac{t_{\text{max}} - t_{\text{min}}}{N+1} \tag{3.6}$$

Each contiguous node will be separated by Δx on the spatial axis and Δt on the temporal axis. As Δx and Δt decreases, the number of nodes in the grid will increase. Therefore, we

refer to Δx and Δt as the resolution of the grid.

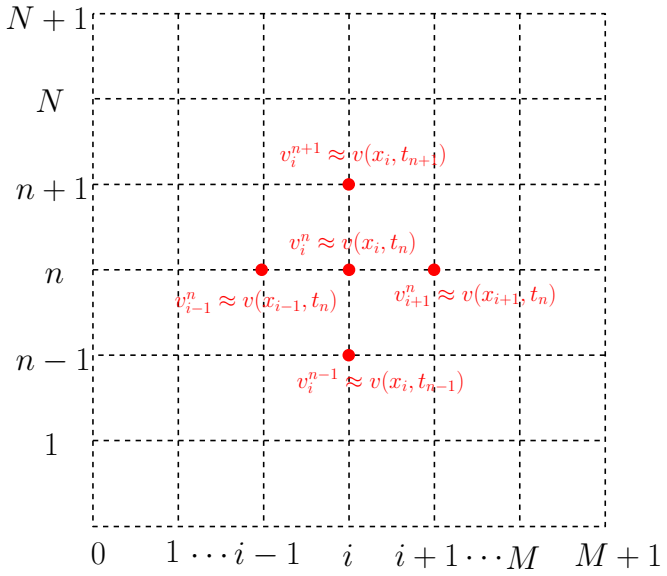


Figure 3.1: The grid $\mathcal{G}$ and the approximation $v_i^n \approx v(x_i, t_n)$ in each node.

From (3.1), it is clear that $t_{\text{min}} = 0$ and $t_{\text{max}} = T$. Moreover, for call options, $x_{\text{min}} = 0$ and $x_{\text{max}} = 1$. Likewise, for put options, $x_{\text{min}} = 1$ and $x_{\text{max}} = x_\infty$ where x_∞ is arbitrary large value. Now that we defined our grid, our goal is to approximate the value function $v(x, t)$ and the optimal exercise price $\bar{S}(t)$ at each node of the grid $\mathcal{G}$

$$v_i^n \approx v(x_i, t_n), \quad \bar{S}^n \approx \bar{S}(t_n)$$

Moreover, we want that the error of the approximation converges to zero value. Specifically, we want that the approximation error at each node

$$e_i^n := v_i^n - v(x_i, t_n) \tag{3.7}$$

goes to zero as Δx and Δt decrease. (3.7) is the local truncation error, and it measures the approximation error at time t_n. It is important to state if a single node has inferior order than the rest of the nodes, it might degrade the order of the truncation error in overall.

Finally, we need ways to approximate derivatives. Here is where finite differences schemes come into play. The idea of finite differences is trivial which is approximating derivatives as the difference of contiguous nodes in the grid. Let us say we are at point x, then the forward

differences approximate the derivative as

$$\frac{f(x+h)-f(x)}{h} = \frac{df}{dx} + O(h)$$

Conversely, the backward difference approximate the derivative as

$$\frac{f(x)-f(x-h)}{h} = \frac{df}{dx} + O(h)$$

As you can observe forward and backward difference approximation yield a local truncation error of $O(h)$. Moreover, the central finite difference approximate the first order derivative as

$$\frac{f(x+h)-f(x-h)}{2h} = \frac{df}{dx} + O(h^2)$$

and for second order derivatives as

$$\frac{f(x+h)-2f(x)+f(x-h)}{h^2} = \frac{d^2f}{dx^2} + O(h^2)$$

Note that both approximation offers a better order of convergence that forward and backward difference, but you are required to come up with strategies for approximating the derivative at the boundary of your grid where $x+h$ or $x-h$ is not defined.

3.2 Explicit scheme

Generally, explicit schemes use forward finite difference to approximate the temporal partial derivative and central finite difference to approximate the spatial derivative at time t_{n+1} and position x_i. However, since the problem (2.21) is written backward in time, we use backward finite difference at t_{n+1}, and a central finite difference at x_i.

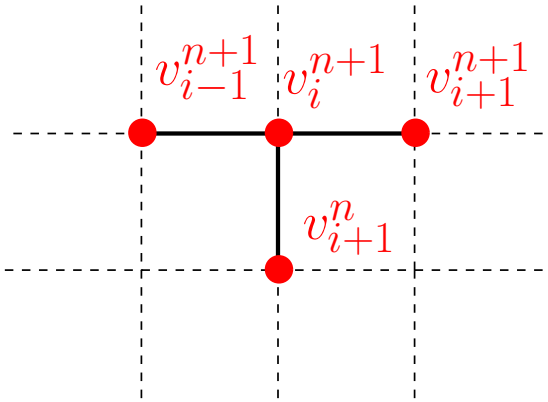


Figure 3.2: Stencil diagram of the explicit scheme.

The central finite difference for the first order and second order spatial partial derivative is given by

$$\frac{v_{i+1}^{n+1} - v_{i-1}^{n+1}}{2\Delta x} = \frac{\partial v}{\partial x} + O(\Delta x^2) \qquad \text{for } i = 1, \ldots, M \tag{3.8}$$

$$\frac{v_{i+1}^{n+1} - 2v_i^{n+1} + v_{i-1}^{n+1}}{\Delta x^2} = \frac{\partial^2 v}{\partial x^2} + O(\Delta x^2) \qquad \text{for } i = 1, \ldots, M \tag{3.9}$$

As it can be observed in figure (3.2), the first and second order central finite difference approximations at node (x_i, t_{n+1}) require to compute the difference at the nodes (x_{i-1}, t_{n+1}) and (x_{i+1}, t_{n+1}). Hence, we can only approximate the spatial partial derivative at the internal region of the grid $\mathcal{G}$ given by the nodes (x_i, t_n) for $i = 1, \ldots, M$. Also note, that the central finite difference has second order convergence in space. In other words, as we decrease Δx by one decimal place, the approximation error will decrease by two decimal places.

Analogously, the backward difference approximation at t_{n+1} for $v(x,t)$ and the optimal exercise price $\bar{S}(t)$ is given by

$$\frac{v_i^{n+1} - v_i^n}{\Delta t} = \frac{\partial v}{\partial t} + O(\Delta t) \qquad \text{for } n = N, \ldots, 0 \tag{3.10}$$

$$\frac{\bar{S}^{n+1} - \bar{S}^n}{\Delta t} = \bar{S}'(t) + O(\Delta t) \qquad \text{for } n = N, \ldots, 0 \tag{3.11}$$

Contrary to the central finite difference, the backward finite difference approximations have first order convergence in time. While it would be desirable to have second order convergence for the temporal partial derivative approximation, it is not possible use central finite difference

because we would be required to have two boundary conditions in the time axis. By combining the finite difference approximations (3.8), (3.9), (3.10), and (3.10),the approximation of the PDE in (2.21) is given by

$$\frac{v_i^{n+1} - v_i^n}{\Delta t} + \frac{1}{2}\sigma^2 x_i^2 \frac{v_{i-1}^{n+1} - 2v_i^{n+1} + v_{i+1}^{n+1}}{(\Delta x)^2} + x_i\left((r-\delta) - \frac{1}{\bar{S}^{n+1}}\frac{\bar{S}^{n+1} - \bar{S}^n}{\Delta t}\right)\frac{v_{i+1}^{n+1} - v_{i-1}^{n+1}}{2\Delta x} - r v_i^{n+1} = 0$$

for $i = 1, \ldots, M$ and $n = N, \ldots, 0$. To simplify the expression above, we introduce the terms

$$\begin{aligned}
\lambda &:= \frac{\Delta t}{(\Delta x)^2} \\
A_i &:= \frac{\lambda}{2}\sigma^2 x_i^2 - \frac{\lambda}{2}\left((r-\delta) - \frac{1}{\Delta t}\right)x_i \Delta x && \text{for } i = 1, \ldots, M \\
B_i &:= 1 - \lambda\sigma^2 x_i^2 - r\Delta t && \text{for } i = 1, \ldots, M \\
C_i &:= \frac{\lambda}{2}\sigma^2 x_i^2 + \frac{\lambda}{2}\left((r-\delta) - \frac{1}{\Delta t}\right)x_i \Delta x && \text{for } i = 1, \ldots, M \\
D_i^{n+1} &:= \frac{x_i}{2\Delta x}\frac{v_{i+1}^{n+1} - v_{i-1}^{n+1}}{\bar{S}^{n+1}} && \text{for } i = 1, \ldots, M
\end{aligned}$$

Then, we rearrange the finite difference approximation of the PDE as

$$v_i^n - D_i^{n+1}\bar{S}^n = A_i v_{i-1}^{n+1} + B_i v_i^{n+1} + C_i v_{i+1}^{n+1} \tag{3.12}$$

for $i = 1, \ldots, M$ and $t = N, \ldots, 0$. Moreover, the PDE problem in (2.21) have well-defined spatial boundary conditions. For call options, the boundary conditions are located at $x = 0$ and $x = 1$. Similarly, for put options, the boundary conditions are located at $x = 1$ and at a sufficient large x. However, since the $\mathcal{G}$ is defined in terms of x_{min} and x_{max}, regardless of the option type, the boundary conditions will be always at x_0 and x_{M+1}.

$$\textbf{Call:} \qquad v_0^n = 0, \qquad v_{M+1}^n = \bar{S}^n - K \tag{3.13a}$$

$$\textbf{Put:} \qquad v_0^n = K - \bar{S}^n, \qquad v_{M+1}^n = 0 \tag{3.13b}$$

Likewise, the terminal conditions are located at t_{N+1} $i = 0, \ldots, M+1$

$$v_i^{N+1} = 0, \qquad \bar{S}^{N+1} = K \tag{3.14a}$$

Moreover, for the problem (2.21), we have contact point condition (2.19). The contact point condition gives the slope at $x = 1$. When the option is a call option, $x = 1$ correspond to x_{M+1} in the grid $\mathcal{G}$. Reciprocally, for a put option, $x = 1$ correspond to x_0. Therefore, by using backward difference at x_{M+1} and forward difference at x_0, the contact point approximation for call and put options, respectively.

$$\begin{aligned} \textbf{Call:} \qquad & \frac{v_{M+1}^n - v_M^n}{\Delta x} = \frac{\partial v}{\partial x}(1,t) + O(\Delta x) \\ \textbf{Put:} \qquad & \frac{v_1^n - v_0^n}{\Delta x} = \frac{\partial v}{\partial x}(1,t) + O(\Delta x) \end{aligned}$$

Using the contact point condition, we obtain an explicit expression for v_M^n

$$\textbf{Call:} \qquad v_M^n = v_{M+1}^n - \Delta x \bar{S}^n = (1 - \Delta x)\bar{S}^n - K \qquad \text{for } n = N, \ldots, 0 \tag{3.15a}$$

$$\textbf{Put:} \qquad v_1^n = v_0^n - \Delta x \bar{S}^n = K - (1 + \Delta x)\bar{S}^n \qquad \text{for } n = N, \ldots, 0 \tag{3.15b}$$

Note that the approximation for v_M^n has first order convergence in space which could degrade the global convergence of the explicit method to first order in space even if we are using central finite difference to approximate the spatial partial derivatives of $v(x,t)$. Similarly, we can obtain explicit expression for $\bar{S}^n$ by computing (3.12) at x_M and at x_1 for call and put options, respectively. Then, rearranging the resulting expression in terms of $\bar{S}^n$

$$\textbf{Call:} \qquad \bar{S}^n = \frac{K + A_M v_{M-1}^{n+1} + B_M v_M^{n+1} + C_M v_{M+1}^{n+1}}{(1 - \Delta x) - D_M^{n+1}} \tag{3.16a}$$

$$\textbf{Put:} \qquad \bar{S}^n = \frac{K - (A_1 v_0^{n+1} + B_1 v_1^{n+1} + C_1 v_2^{n+1})}{D_1^{n+1} + (1 + \Delta x)} \tag{3.16b}$$

for $n = N, \ldots, 0$. Thus, combining (3.12), (3.13), (3.14), (3.15), and (3.16), the explicit

scheme of PDE problem (2.21) is given by

Call:

$$\begin{cases} v_i^n - D_i^{n+1}\bar{S}^n = A_i v_{i-1}^{n+1} + B_i v_i^{n+1} + C_i v_{i+1}^{n+1} & \text{for } i = 1, \dots, M-1 \text{ and } n = N, \dots, 0 \\ v_i^{N+1} = 0 & \text{for } i = 0, \dots, M+1 \\ \bar{S}^{N+1} = K & \\ v_0^n = 0 & \text{for } n = N, \dots, 0 \\ v_M^n = (1 - \Delta x)\bar{S}^n - K & \text{for } n = N, \dots, 0 \\ v_{M+1}^n = \bar{S}^n - K & \text{for } n = N, \dots, 0 \end{cases} \tag{3.17a}$$

Put:

$$\begin{cases} v_i^n - D_i^{n+1}\bar{S}^n = A_i v_{i-1}^{n+1} + B_i v_i^{n+1} + C_i v_{i+1}^{n+1} & \text{for } i = 2, \dots, M \text{ and } n = N, \dots, 0 \\ v_i^{N+1} = 0 & \text{for } i = 0, \dots, M+1 \\ \bar{S}^{N+1} = K & \\ v_0^n = K - \bar{S}^n & \text{for } n = N, \dots, 0 \\ v_1^n = K - (1 + \Delta x)\bar{S}^n & \text{for } n = N, \dots, 0 \\ v_M^n = 0 & \text{for } n = N, \dots 0 \end{cases} \tag{3.17b}$$

Finally, we formulate an algorithm for solving the system (3.17)

Algorithm 3.1 Explicit method for call options

Ensure: $\lambda \leq 0.5$

for $i = 0, \ldots, M+1$ **do**

$v_i^{N+1} = 0$

$\bar{S}^{N+1} = K$

for $i = 1, \ldots, M$ **do**

$$A_i = \frac{\lambda}{2}\sigma^2 x_i^2 - \frac{\lambda}{2}\left((r-\delta) - \frac{1}{\Delta t}\right)x_i \Delta x$$

$$B_i = 1 - \lambda\sigma^2 x_i^2 - r\Delta t$$

$$C_i = \frac{\lambda}{2}\sigma^2 x_i^2 + \frac{\lambda}{2}\left((r-\delta) - \frac{1}{\Delta t}\right)x_i \Delta x$$

for $n = N, \ldots, 0$ **do**

for $i = 1, \ldots, M$ **do**

$$D_i^{n+1} = \frac{x_i}{2\Delta x}\frac{v_{i+1}^{n+1} - v_{i-1}^{n+1}}{\bar{S}^{n+1}}$$

$$\bar{S}^n = \frac{K + A_M v_{M-1}^{n+1} + B_M v_M^{n+1} + C_M v_{M+1}^{n+1}}{(1-\Delta x) - D_M^{n+1}}$$

$v_0^n = 0$

$v_M^n = (1-\Delta x)\bar{S}^n - K$

$v_{M+1}^n = \bar{S}^n - K$

for $i = 1, \ldots, M-1$ **do**

$$v_i^n = A_i v_{i-1}^{n+1} + B_i v_i^{n+1} + C_i v_{i+1}^{n+1} + D_i^{n+1}\bar{S}^n$$

Algorithm 3.2 Explicit method for put options

for $i = 0, \ldots, M+1$ **do**

$v_i^{N+1} = 0$

$\bar{S}^{N+1} = K$

for $i = 1, \ldots, M$ **do**

$A_i = \frac{\lambda}{2}\sigma^2 x_i^2 - \frac{\lambda}{2}\left((r-\delta) - \frac{1}{\Delta t}\right)x_i \Delta x$

$B_i = 1 - \lambda\sigma^2 x_i^2 - r\Delta t$

$C_i = \frac{\lambda}{2}\sigma^2 x_i^2 + \frac{\lambda}{2}\left((r-\delta) - \frac{1}{\Delta t}\right)x_i \Delta x$

for $n = N, \ldots, 0$ **do**

for $i = 1, \ldots, M$ **do**

$D_i^{n+1} = \frac{x_i}{2\Delta x}\frac{v_{i+1}^{n+1} - v_{i-1}^{n+1}}{\bar{S}^{n+1}}$

$\bar{S}^n = \frac{K - (A_1 v_0^{n+1} + B_1 v_1^{n+1} + C_1 v_2^{n+1})}{D_1^{n+1} + (1 + \Delta x)}$

$v_0^n = K - \bar{S}^n$

$v_1^n = K - (1 + \Delta x)\bar{S}^n$

$v_{M+1}^n = 0$

for $i = 2, \ldots, M$ **do**

$v_i^n = A_i v_{i-1}^{n+1} + B_i v_i^{n+1} + C_i v_{i+1}^{n+1} + D_i^{n+1}\bar{S}^n$

3.3 Implicit scheme

Analogously to the previous section, implicit methods approximate the temporal partial derivative using backward difference and the spatial partial derivative using a central difference at time t_n and position x_i. Since the PDE in (2.21) is written backward in time, we use a forward difference instead.

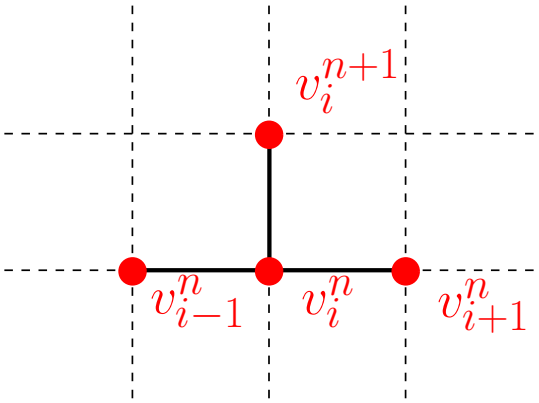


Figure 3.3: Stencil diagram of the implicit scheme.

Therefore, the central difference for the first and second order spatial partial derivative at time t_n is

$$\frac{v_{i+1}^n - v_{i-1}^n}{2\Delta x} = \frac{\partial v}{\partial x} + O(\Delta x^2) \tag{3.18}$$

$$\frac{v_{i+1}^n - 2v_i^n + v_{i-1}^n}{\Delta x^2} = \frac{\partial^2 v}{\partial x^2} + O(\Delta x^2) \tag{3.19}$$

for $i = 1, \ldots, M$. Likewise, the forward difference of $v(x,t)$ and $\bar{S}(t)$ at position x_i is

$$\frac{v_i^{n+1} - v_i^n}{\Delta t} = \frac{\partial v}{\partial t} + O(\Delta t) \tag{3.20}$$

$$\frac{\bar{S}^{n+1} - \bar{S}^n}{\Delta t} = \bar{S}'(t) + O(\Delta t) \tag{3.21}$$

for $n = N, \ldots, 0$. Hence, combining (3.18), (3.19), (3.20) and (3.21),we obtain the implicit

approximation of the PDE (2.21) as

$$\frac{v_i^{n+1} - v_i^n}{\Delta t} + \frac{1}{2}\sigma^2 x_i^2 \frac{v_{i-1}^n - 2v_i^n + v_{i+1}^n}{(\Delta x)^2} + x_i\left((r-\delta) - \frac{1}{\bar{S}^n}\frac{\bar{S}^{n+1} - \bar{S}^n}{\Delta t}\right)\frac{v_{i+1}^n - v_{i-1}^n}{2\Delta x} - rv_i^n = 0$$

for $i = 1, \ldots, M$ and $n = N, \ldots, 0$. Similar to the explicit method, the approximation error is second order in space and first order in time. Again, to make the implicit approximation more manageable, we introduce the following terms

$$\alpha_i^n := -\frac{\lambda}{2}\sigma^2 x_i^2 + \frac{\lambda \Delta x}{2} x_i\left(r - \delta + \frac{\bar{S}^{n+1} - \bar{S}^n}{\Delta t \bar{S}^n}\right) \tag{3.22}$$

$$\beta_i^n := 1 + \lambda\sigma^2 x_i^2 + r\Delta t \tag{3.23}$$

$$\gamma_i^n := -\frac{\lambda}{2}\sigma^2 x_i^2 + \frac{\lambda \Delta x}{2} x_i\left(r - \delta + \frac{\bar{S}^{n+1} - \bar{S}^n}{\Delta t \bar{S}^n}\right) \tag{3.24}$$

and rearrange the PDE as

$$\alpha_i^n v_{i-1}^n + \beta_i^n v_i^n + \gamma_i^n v_{i+1}^n = v_i^{n+1} \tag{3.25}$$

The boundary and terminal conditions are given by (3.13) and (3.14). Likewise, the approximation of v_M^n or v_1^n for put and call, respectively, is given by (3.15). Similar to the explicit method, the approximation v_M^n and v_0^n given by the contact point condition is first order in space. Hence, the global approximation error of implicit scheme might be degraded to first order in space. Contrary to the explicit method, there is not an explicit expression for $\bar{S^n}$. Now, we formulate the system of equations of the problem (2.21) using (3.13), (3.14), (3.15)

and (3.25)

$$\textbf{Call:}\quad \begin{cases} \alpha_i^n v_{i-1}^n + \beta_i^n v_i^n + \gamma_i^n v_{i+1}^n = v_i^{n+1} & \text{for } i = 1, \dots, M-1 \text{ and } n = N, \dots, 0 \\ v_i^{N+1} = 0 & \text{for } i = 0, \dots, M+1 \\ \bar{S}^{N+1} = K & \\ v_0^n = 0 & \text{for } n = N, \dots 0 \\ v_M^n = (1 - \Delta x)\bar{S}^n - K & \text{for } n = N, \dots 0 \\ v_{M+1}^n = \bar{S}^n - K & \text{for } n = N, \dots 0 \end{cases} \tag{3.26a}$$

$$\textbf{Put:}\quad \begin{cases} \alpha_i^n v_{i-1}^n + \beta_i^n v_i^n + \gamma_i^n v_{i+1}^n = v_i^{n+1} & \text{for } i = 2, \dots, M \text{ and } n = N, \dots, 0 \\ v_i^{N+1} = 0 & \text{for } i = 0, \dots, M+1 \\ \bar{S}^{N+1} = K & \\ v_0^n = K - \bar{S}^n & \text{for } n = N, \dots 0 \\ v_1^n = K - (1 + \Delta x)\bar{S}^n & \text{for } n = N, \dots 0 \\ v_M^n = 0 & \text{for } n = N, \dots 0 \end{cases} \tag{3.26b}$$

Since there is not an explicit formula for v_i^n and $\bar{S}^n$, we will have to solve a non-linear system of equation. Let's define the vector $\mathbf{v}^n \in \mathbb{R}^{M-1}$

$$\textbf{Call:}\qquad \mathbf{v}^n := \begin{bmatrix} v_1^n, & v_2^n, & \cdots, & v_{M-1}^n \end{bmatrix}^\mathsf{T} \tag{3.27a}$$

$$\textbf{Put:}\qquad \mathbf{v}^n := \begin{bmatrix} v_2^n, & v_3^n, & \cdots, & v_M^n \end{bmatrix}^\mathsf{T} \tag{3.27b}$$

the matrix $\Lambda^n \in \mathbb{R}^{M-1,M-2}$

$$\textbf{Call:} \qquad \Lambda^n = \begin{bmatrix} \beta_1^n & \gamma_1^n & & & & \\ \alpha_2^n & \beta_2^n & \gamma_2^n & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \alpha_{M-2}^n & \beta_{M-2}^n & \gamma_{M-2}^n \\ & & & & \alpha_{M-1}^n & \beta_{M-1}^n \\ & & & & & \alpha_M^n \end{bmatrix} \tag{3.28a}$$

$$\textbf{Put:} \qquad \Lambda^n := \begin{bmatrix} \gamma_1^n & & & & & \\ \beta_2^n & \gamma_2^n & & & & \\ \alpha_3^n & \beta_3^n & \gamma_3^n & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \alpha_{M-1}^n & \beta_{M-1}^n & \gamma_{M-1}^n \\ & & & & \alpha_M^n & \beta_M^n \end{bmatrix} \tag{3.28b}$$

and the vector $\mathbf{f}^n \in \mathbb{R}^{M-1}$

$$\textbf{Call:} \qquad \mathbf{f}^n := \begin{bmatrix} v_1^{n+1} \\ \vdots \\ v_{M-1}^{n+1} - \gamma_{M-1}^n[(1-\Delta x)\bar{S}^n - K] \\ v_M^{n+1} - \gamma_M^n(\bar{S}^n - K) - \beta_M^n[(1-\Delta x)\bar{S}^n - K] \end{bmatrix} \tag{3.29a}$$

$$\textbf{Put:} \qquad \mathbf{f}^n := \begin{bmatrix} v_1^{n+1} - \alpha_1^n(K - \bar{S}^n) - \beta_1^n[K - (1+\Delta x)\bar{S}^n] \\ v_2^{n+1} - \beta_2^n[K - (1+\Delta x)\bar{S}^n] \\ v_3^{n+1} \\ \vdots \\ v_{M-1}^{n+1} \end{bmatrix} \tag{3.29b}$$

Thus, the non-linear system of equations that we need to solve is

$$F(\mathbf{v}^n, \bar{S}^n) = \Lambda^n \mathbf{v}^n - \mathbf{f}^n = 0 \tag{3.30}$$

By computing the Jacobian of the system, we con solve the non-linear system using the newton's method

$$\mathbf{y}_{k+1} = \mathbf{y}_k - J^{-1}(\mathbf{y}_k) F(\mathbf{y}_k) \tag{3.31}$$

where y_k is some approximation of the solution

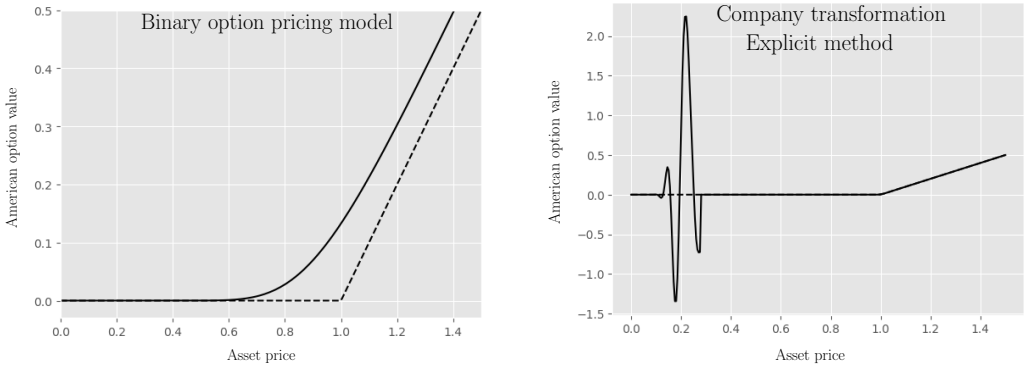
$$\mathbf{y} = \left[\mathbf{v}^n | \bar{S}^n \right]^{\mathsf{T}} \tag{3.32}$$

3.4 Numerical results

Generally, Datasets for American options are hard to get and often require paying substantial amount of money. Therefore, to validate our implementation, we mainly relied on the data available in Company, et al. [3], Nielsen, et al. [12], Seydel [14], and Wilmott, et al. [15]. Moreover, we used the approximations produced by the binomial model introduced by Cox et al. [5] as benchmark for assessing the consistency of our method. We chose the binomial model because it uses a completely different approach to price options than the one considered in our work, is widely used in the industry, and is simple to implement. First, we want to assess the correct functionality of our implementation. Naturally, the first test that come into main is to price call and put options without dividend. Therefore, we define the set of parameters taken from [12]

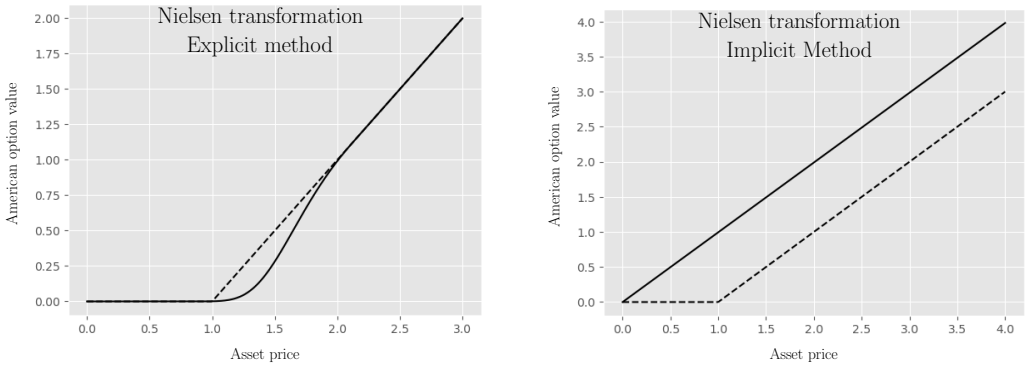
$$K = 1, \quad T = 1, \quad r = 0.2, \quad \sigma = 0.02, \quad \delta = 0 \tag{3.33}$$

Additionally, We use Nielsen [12] optimal exercise boundary $\bar{S}(t) \approx 0.86$ for parameters as a reference (3.33). Also, we approximated $\bar{S}(t)$ using the binomial option pricing model also as a reference. Finally, we priced call and put options using our implementation of the explicit (3.17) and implicit (3.26) method for the Nielsen transformation, and the explicit method for



(a) Nodes $= 2^{500}$.

(b) $\Delta x = 1\text{e}{-3}$ and $\Delta t = 0.5 \times 1\text{e}{-6}$



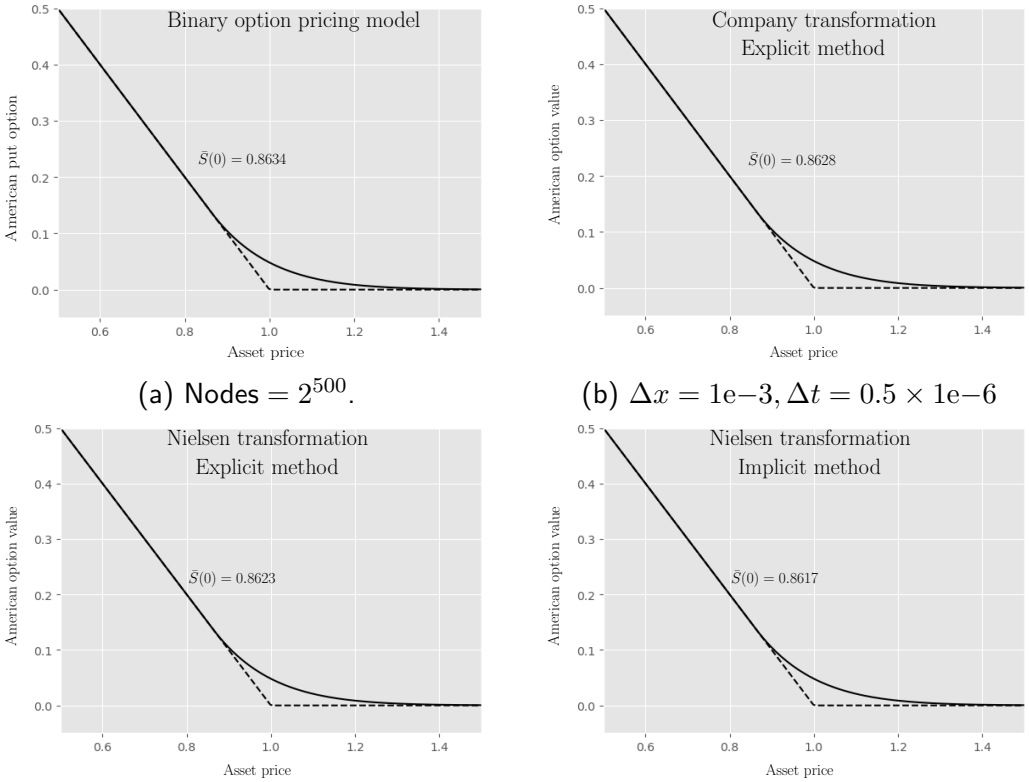
(c) $\Delta x = 1\text{e}{-3}, \Delta t = 0.5 \times 1\text{e}{-6}$

(d) $\Delta x = 1\text{e}{-3}, \Delta t = 0.5 \times 1\text{e}{-6}$

Figure 3.4: American call option value $V(S, 0)$ curve.

the Company transformation (A.1) (See appendix). As you can observe in figure (3.4b), (3.4c), and (3.4d), our implementation for the Nielsen and Company transformation seems to produce the value as per (2.1a). However, by taking a look to figure (3.4a), you can observe that the binomial option pricing model is also not capturing the geometrical properties defined in figure (2.1a). This behavior can explain by saying that American call options without dividends cannot be modelled using (2.15a). In fact, Merton [11] showed that, pricing American options without dividends is equivalent to solve (2.5). Therefore, we can conclude that we applied the wrong methods to price call options without dividends.

Now, let us consider the case for put options without dividends for the same given parameters. Figures (3.5) show the $V(S, 0)$ curve obtained by the Nielsen and Company explicit method. In each plot, we have listed the optimal exercise boundary $\bar{S}(0)$. Also, we have listed the correspondent value curve produced using the binary option pricing model using 2^{500} nodes. As you see, Nielsen and Company approximated the optimal exercise boundary within 2 decimal places. Moreover, in table (3.1), you can find the RMSE error produced by the explicit method



(a) Nodes $= 2^{500}$.

(b) $\Delta x = 1\text{e}{-3}, \Delta t = 0.5 \times 1\text{e}{-6}$

(c) $\Delta x = 1\text{e}{-3}, \Delta t = 0.5 \times 1\text{e}{-6}$

(d) $\Delta x = \Delta t = 1\text{e}{-3}$.

Figure 3.5: American put option value $V(S, 0)$ curve.

for the Company transformation. Other tables can be found in appendix (C). Contrary to the case for call options, you can observe in figures (3.5) that the approximations of the value curve $V(S, t)$ captures the geometry of American put options described by figure (2.1b). Specifically, within the continuation region $S > \bar{S}(0)$, the value is larger than the payoff function and as S gets larger, the value goes to zero. In addition, the value curve is exactly the payoff. Also notice that the RSME of the explicit method for the Company transformation fairly low which indicate us that our implementation for the method works for American put options.

Asset price	**BOPM**	0.125	0.0625	0.03125	0.015625	0.0078125
0.8	0.200000	0.200000	0.200000	0.200000	0.200000	0.200000
1.0	0.048167	0.049286	0.049274	0.048465	0.048256	0.048174
1.2	0.008666	0.010736	0.009108	0.008829	0.008686	0.008667
1.4	0.001285	0.001950	0.001501	0.001349	0.001295	0.001287
1.6	0.000167	0.000354	0.000224	0.000185	0.000172	0.000168
1.8	0.000020	0.000076	0.000035	0.000024	0.000021	0.000020
2.0	0.000002	0.000017	0.000006	0.000003	0.000003	0.000002
	RSME	0.000092	0.000045	0.000013	0.000003	0.00000

Table 3.1: RSME error produced by the explicit scheme for Company transformation for $\Delta t = 1/8, 1/16, \ldots, 1/128$ and $\Delta t = \Delta x^2/2$.

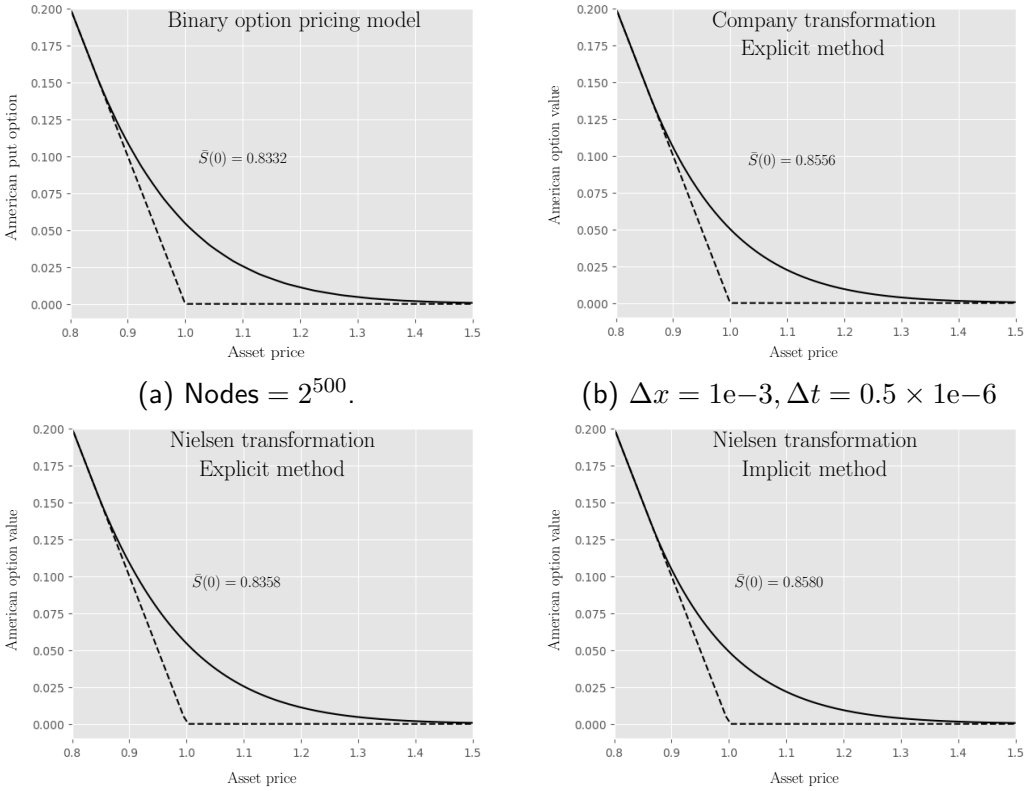
Additionally, we proceeded to the case for call and put options with dividends. We used the same set of parameters (3.33) but with the slight modification so that the underlying asset pays dividends

$$K = 1, \quad T = 1, \quad r = 0.2, \quad \sigma = 0.2, \quad \delta = 0.03 \tag{3.34}$$

In table (3.2), we present the approximation error produced by the explicit method for the Nielsen transformation, and in figure (3.6), it is shown the value curve obtained for each of the methods with parameters (3.34). See appendix (C) for the same experiments for call options.

Asset price	**BOPM**	0.125	0.0625	0.03125	0.015625	0.0078125
0.8	0.200000	0.200000	0.200000	0.200000	0.200000	0.200000
1.0	0.054464	0.053355	0.054588	0.054501	0.054465	0.054469
1.2	0.011255	0.011585	0.011457	0.011255	0.011179	0.011161
1.4	0.001855	0.002219	0.002006	0.001889	0.001857	0.001847
1.6	0.000262	0.000399	0.000315	0.000277	0.000268	0.000265
	RSME	0.00041	0.000010	0.000002	0.000002	0.000003

Table 3.2: RSME error produced by the explicit scheme for Nielsen transformation to approximate put option (3.34) given $\Delta t = 1/8, 1/16, \ldots, 1/128$ and $\Delta t = \Delta x^2/2$.



(a) Nodes $= 2^{500}$.

(b) $\Delta x = 1\text{e}{-3}, \Delta t = 0.5 \times 1\text{e}{-6}$

(c) $\Delta x = 1\text{e}{-3}, \Delta t = 0.5 \times 1\text{e}{-6}$

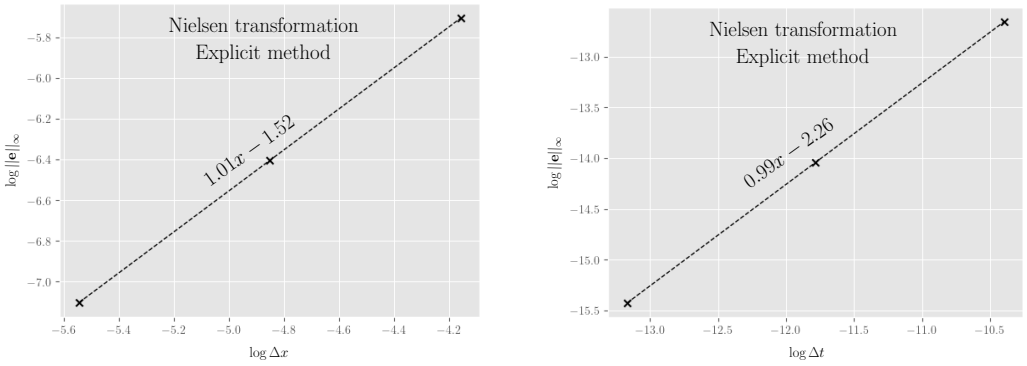
(d) $\Delta x = \Delta t = 1\text{e}{-3}$.

Figure 3.6: American put option value $V(S, 0)$ curve.

Finally, we conducted a convergence analysis to determine the order of convergence of each the methods. For simplicity, the numerical experiment was conducted for put options and with parameters (3.34). However, we expect that numerical experiment will yield similar results for call options and other set of parameters. Moreover, for analyzing the convergence, we are taking the relative error between contiguous grid in sizes. Specifically, to determine the order of convergence in space, we define the grids with resolution $\Delta x_i = h/2^i$ for $i = 0, \ldots, 3$ while Δt is fixed. Conversely, to determine the temporal order of convergence, we vary Δt and fix Δx in similar way. Therefore, the error between consecutive grids is defined as

$$\mathbf{e}_k = ||\mathbf{v}_{k+1} - \mathbf{v}_k||_\infty \quad \text{for } k = 0, \ldots, 3 \tag{3.35}$$

where $\mathbf{v}_k \in \mathbb{R}^{M+1}$ is the vector with the approximation produced by our method and $||\cdot||_\infty$ is the infinity norm. In figure (3.7), we show log-log plots of the error produced by the explicit and implicit methods for Nielsen transformation. As you can see, both explicit and implicit methods have first order convergence in space. Recall that although we are using central finite difference,

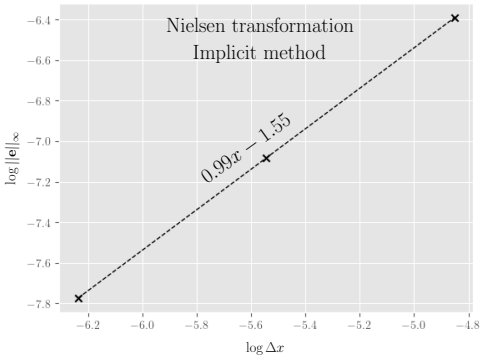


(a) $\Delta x = 2^{-7}, \ldots, 2^{-10}$

$\Delta t = 2^{-21}$

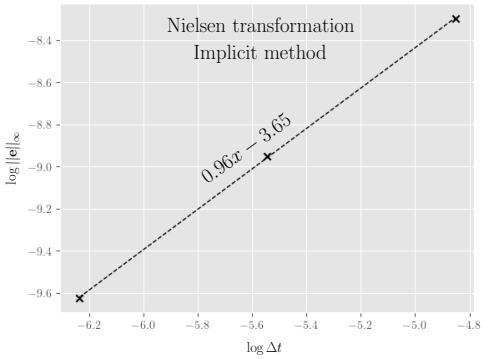
(b) $\Delta t = 2^{-15}, 2^{-17}, \ldots, 2^{-21}$

$\Delta x = 2^{-7}$



(c) $\Delta x = 2^{-6}, \ldots, 2^{-8}$

$\Delta t = 2^{-2}$

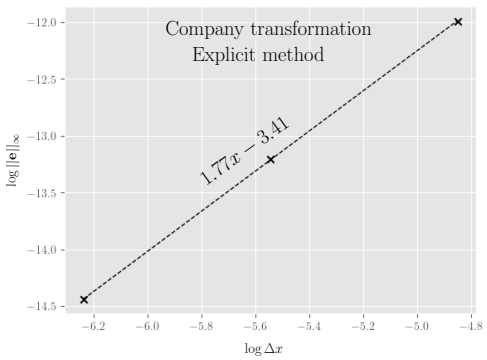


(d) $\Delta t = 2^{-7}, 2^{-8}, \ldots, 2^{-10}$

$\Delta x = 2^{-5}$

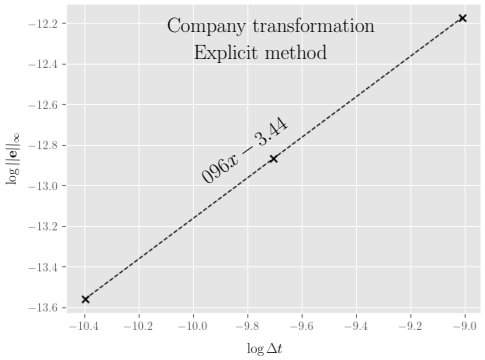
Figure 3.7: Convergence analysis for the explicit and implicit method for the Nielsen transformation.

the Nielsen schemes use forward (or backward for calls) difference to approximate the contact point condition (3.15) which is first order in space, hence, degrading the overall convergence of the method. In the other hand, figure (3.8) shows the convergence order for the explicit method for the Company transformation. Contrary to Nielsen's schemes, the explicit method for Company transformation is second order in space. This is because Company uses a central finite difference scheme for approximating the contact point condition (A.1). Moreover, all methods are first order convergence in space as suggested by the forward/backward difference approximation used for the temporal axis.



(a) $\Delta x = 2^{-7}, \ldots, 2^{-10}$

$\Delta t = 2^{-21}$



(b) $\Delta t = 2^{-15}, 2^{-17}, \ldots, 2^{-21}$

$\Delta x = 2^{-7}$

Figure 3.8: Convergence analysis for the explicit method for the Company transformation.

4 Linear complementary problem

4.1 Overview

A linear complementary problem (LCP) is an optimization problem in which the goal is to solve a system of linear equations subject to a set of complementary conditions. More formally, given the matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ and the vector $\mathbf{b} \in \mathbb{R}^d$, find the vectors $\mathbf{v} \in \mathbb{R}^d$ and $\mathbf{w} \in \mathbb{R}^d$ such as

$$\mathbf{A}\mathbf{v} - \mathbf{w} = \mathbf{b}$$

subject to the complementary conditions

$$\begin{aligned} v_i &\geq 0 \qquad \text{for } i = 1, \ldots, d \\ w_i &\geq 0 \qquad \text{for } i = 1, \ldots, d \\ \mathbf{v}^T \mathbf{w} &= 0 \end{aligned}$$

Note that the complementary conditions require that either $v_i = 0$ or $w_i = 0$. As you will see later, the pricing problem for American options can be reformulated as a linear complementary problem. Let us reconsider what the Black-Scholes model tells about American options. First, that the value $V(S,t)$ is bounded from below by the payoff function

$$V(S,t) - H(S,t) \geq 0 \qquad \text{for all } (S,t)$$

Moreover, that the value function can be divided in two complementary regions: the exercise region (2.8) where

$$V(S,t) - H(S,t) > 0 \qquad \text{for } (S,t) \in \mathcal{C} \tag{4.1}$$

the continuation region (2.9) where

$$V(S,t) - H(S,t) = 0 \qquad \text{for } (S,t) \in \mathcal{S} \tag{4.2}$$

Finally, value function $V(S,t)$ is the solution to the Black-Scholes PDE within the continuation region

$$\frac{\partial V}{\partial t} + \mathcal{L}_{\mathsf{BS}}(V) = 0 \qquad \text{for } (S,t) \in \mathcal{C}$$

where $\mathcal{L}_{\mathsf{BS}}(\cdot)$ is the linear parabolic operator (2.6) applied to the function $V(S,t)$. Now, let us consider what happens to the Black-Scholes PDE in the exercise region. From equation (4.2), we know that $V(S,t) = H(S,t)$. Hence by plugin $H(S,t)$ into the Black-Scholes PDE, we obtain the Black-Scholes PDE inequality in the exercise region

$$\frac{\partial V}{\partial t} + \mathcal{L}_{\mathsf{BS}} \leq 0 \qquad \text{for } (S,t) \in \mathcal{S} \tag{4.3}$$

By grouping (4.1), (4.2), (4.3), we form the system of variational inequalities

$$\begin{cases} \left[\frac{\partial V}{\partial \tau} - \mathcal{L}_{\mathsf{BS}}(V)\right] \cdot [V(S,T-\tau) - H(S,T-\tau)] = 0 & \text{for all } (S,t) \\ V(S,T-\tau) - H(S,T-\tau) \geq 0 & \text{for all } (S,t) \\ \frac{\partial V}{\partial \tau} - \mathcal{L}_{\mathsf{BS}}(V) \geq 0 & \text{for all } (S,t) \\ V(S,T) = H(S,T) \end{cases} \tag{4.4}$$

The benefit of the variational inequalities is that there is no explicit dependence on the optimal exercise price defined in (2.10). Also, note that we rewrote the Black-Scholes PDE forward in time by introducing the transformation $\tau = T - t$ deliberately so that the variational inequalities' formulation looks similar in structure to LCP problem. But before elaborating more about the relationship between the variational inequalities and LCP problems, we introduce the transformations

$$S = Ke^{x}, \quad t = T - \frac{2\tau}{\sigma^2}, \quad h(x,\tau) := e^{(\alpha x + \beta\tau)}\frac{H(S,t)}{K}, \quad v(x,\tau) := V(S,t) =: Ke^{-(\alpha x+\beta\tau)}y(x,\tau) \tag{4.5}$$

where

$$q := \frac{2r}{\sigma^2}, \quad q_\delta := \frac{2(r-\delta)}{\sigma^2}, \quad \alpha := \frac{1}{2}(q_\delta - 1) \quad \beta := \frac{1}{4}(q_\delta - 1)^2 + q \tag{4.6}$$

so that the Black-Scholes PDE (2.5) transforms to the heat diffusion equation

$$\frac{\partial y}{\partial \tau} - \frac{\partial^2 y}{\partial x^2} = 0 \tag{4.7}$$

where $y \in C^{2,1}(\mathcal{F})$ for

$$\mathcal{F} : \mathbb{R} \times [0, \sigma^2 T/2] \tag{4.8}$$

Although we might solve (4.4) without transforming the problem, Dewynne et al. [6] and Seydel [14] suggest that transformation to the heat diffusion equation (4.7) introduces desirable numerical properties when applying finite difference schemes. By applying transformation (4.5), the system (4.4) becomes

$$\begin{cases} \left[\dfrac{\partial y}{\partial \tau} - \dfrac{\partial^2 y}{\partial x^2}\right] \cdot [y(x,\tau) - h(x,\tau)] = 0 & \text{for all } (x,\tau) \\ y(x,\tau) - h(x,\tau) \geq 0 & \text{for all } (x,\tau) \\ \dfrac{\partial y}{\partial \tau} - \dfrac{\partial^2 y}{\partial x^2} \geq 0 & \text{for all } (x,\tau) \\ y(x,0) = h(x,0) \end{cases} \tag{4.9}$$

Now that we have expressed the system of variational inequalities using the heat discussion equation, we proceed to apply the finite difference scheme framework presented in section 3. Specifically, we want to approximate $y(x,\tau)$ at each of the nodes within the grid

$$\mathcal{G} := \{(x_i, \tau_n) : (i,n) \in \{0, \ldots, M+1\} \times \{0, \ldots, N+1\}\} \tag{4.10}$$

where

$$x_i := x_{\text{min}} + i\Delta x \qquad \text{for } i = 0, \ldots, M+1 \tag{4.11}$$

$$\tau_n := t_{\text{min}} + i\Delta\tau \qquad \text{for } i = 0, \ldots, N+1 \tag{4.12}$$

$$\Delta x := \frac{x_{\text{max}} - x_{\text{min}}}{M+1} \tag{4.13}$$

$$\Delta t := \frac{t_{\text{max}} - t_{\text{min}}}{N+1} \tag{4.14}$$

where x_{min} is set sufficiently small value, x_{max} to a sufficiently large value, τ_{min} to 0 and τ_{max} to $\frac{\sigma^2}{2}T$. Recall that we presented explicit and implicit schemes for approximating the PDE (2.18) in section 3. Analogously, we present explicit and implicit schemes for the heat diffusion equation

$$\frac{y_i^{n+1} - y_i^n}{\Delta\tau} = \frac{y_{i-1}^n - 2y_i^n + y_{i+1}^n}{(\Delta x)^2} \tag{4.15}$$

$$\frac{y_i^{n+1} - y_i^n}{\Delta\tau} = \frac{y_{i-1}^{n+1} - 2y_i^{n+1} + y_{i+1}^{n+1}}{(\Delta x)^2} \tag{4.16}$$

for $i = 1, \ldots, M$ and $n = 0 \ldots, N$. Note that contrary to (2.18), the heat equation is written forward in time. Therefore, the explicit scheme correspond to equation (4.15), and the implicit scheme to equation (4.16). In addition, we introduce the Crank-Nicholson scheme[7][14] for equation (4.7) as

$$\frac{y_i^{n+1} - y_i^n}{\Delta\tau} = \frac{1}{2}\left(\frac{y_{i-1}^n - 2y_i^n + y_{i+1}^n}{(\Delta x)^2} + \frac{y_{i-1}^{n+1} - 2y_i^{n+1} + y_{i+1}^{n+1}}{(\Delta x)^2}\right) \tag{4.17}$$

for $i = 1, \ldots, M$ and $n = 0 \ldots, N$. Epperson [7] and Seydel [14] showed that the Crank-Nicholson scheme is $O(\Delta x^2 + \Delta t^2)$.

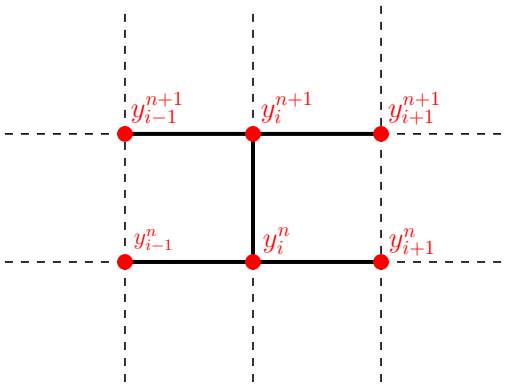


Figure 4.1: Stencil diagram for the Crank-Nicholson scheme.

We can generalize (4.15), (4.16), and (4.17) using a parametrized scheme called the theta method

$$\frac{y_i^{n+1} - y_i^n}{\Delta\tau} = (1-\theta)\frac{y_{i-1}^n - 2y_i^n + y_{i+1}^n}{(\Delta x)^2} + \theta\frac{y_{i-1}^{n+1} - 2y_i^{n+1} + y_{i+1}^{n+1}}{(\Delta x)^2} \tag{4.18}$$

for $i = 1, \ldots, M$, and $n = 0, \ldots, N$. The parameter $\theta \in [0, 1]$ controls what scheme to use. The scheme defaults to the explicit method when $\theta = 0$, to the implicit method when $\theta = 1$, to the Crank-Nicholson when $\theta = 1/2$. Next, we rearrange equation (4.18) as

$$y_i^{n+1} - \lambda\theta(y_{i-1}^{n+1} - 2y_i^{n+1} + y_{i+1}^{n+1}) = y_i^n + (1-\theta)\lambda(y_{i-1}^n - 2y_i^n + y_{i+1}^n) \tag{4.19}$$

for $i = 1, \ldots, M$, $n = 0, \ldots, N$, and $\lambda = \Delta t/\Delta x^2$. Moreover, let us define vectors $\hat{\mathbf{b}}^n \in \mathbb{R}^M$, $\mathbf{y}^{n+1} \in \mathbb{R}^M$ and $\mathbf{h}^n \in \mathbb{R}^M$ as

$$\hat{\mathbf{b}}^n := \left[\hat{b}_1^n, \ldots, \hat{b}_M^n\right]^\mathsf{T} \tag{4.20}$$

$$\mathbf{y}^{n+1} := \left[y_1^{n+1}, \ldots, y_M^{n+1}\right]^\mathsf{T} \tag{4.21}$$

$$\mathbf{h}^{n+1} := \left[h(x_1, \tau_{n+1}), \ldots, h(x_M, \tau_{n+1})\right]^\mathsf{T} \tag{4.22}$$

where

$$\hat{b}_i^n := y_i^n + (1-\theta)\lambda(y_{i-1}^n - 2y_i^n + y_{i+1}^n) \tag{4.23}$$

for $i = 1, \ldots, M$, and $n = 0, \ldots, N$. Finally, we define the matrix $\mathbf{A} \in \mathbb{R}^{M \times M}$ as

$$A := \begin{bmatrix} 1 + 2\lambda\theta & -\lambda\theta & & 0 \\ -\lambda\theta & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ 0 & & \ddots & \ddots \end{bmatrix} \tag{4.24}$$

Hence, using vectors $\hat{\mathbf{b}}^n \in \mathbb{R}^M$, $\mathbf{y}^{n+1} \in \mathbb{R}^M$ and $\mathbf{h}^n \in \mathbb{R}^M$, and the matrix $\mathbf{A} \in \mathbb{R}^{M \times M}$, we could rewrite the system of variational inequalities (4.9)using the finite difference scheme proposed as

$$\begin{cases} (\mathbf{A}\mathbf{y}^{n+1} - \hat{\mathbf{b}}^n)^\mathsf{T}(\mathbf{y}^{n+1} - \mathbf{h}^{n+1}) = 0 \\ \mathbf{A}\mathbf{y}^{n+1} - \hat{\mathbf{b}}^n \geq 0 \\ \mathbf{y}^{n+1} - \mathbf{h}^{n+1} \geq 0 \\ \mathbf{y}^0 = \mathbf{h}^0 \end{cases} \tag{4.25}$$

for $n = 0, \ldots, N$. Moreover, (4.25) can be formulated as the iterative method

Algorithm 4.1 Iterative method for variational inequalities.

$\mathbf{y}^0 = \mathbf{h}^0$

for $n = 0, \ldots, N$ **do**

Find $\mathbf{y}^{n+1}$ such as

$(\mathbf{A}\mathbf{y}^{n+1} - \hat{\mathbf{b}}^n)^\mathsf{T}(\mathbf{y}^{n+1} - \mathbf{h}^{n+1}) = 0$

$\mathbf{A}\mathbf{y}^{n+1} - \hat{\mathbf{b}}^n \geq 0$

$\mathbf{y}^{n+1} - \mathbf{h}^{n+1} \geq 0$

Finally, by defining $\mathbf{v} := \mathbf{y}^{n+1} - \mathbf{h}^{n+1}$, $\mathbf{w} := \mathbf{A}\mathbf{y}^{n+1} - \hat{\mathbf{b}}^n$, $\mathbf{b} := \hat{\mathbf{b}}^n - \mathbf{A}\mathbf{h}^{n+1}$. The iterative method becomes

Algorithm 4.2 Iterative method for LCP.

for $n = 0, \ldots, N$ **do**

Given $\mathbf{A}\mathbf{v} - \mathbf{w} = \mathbf{b}$, find $\mathbf{v}$ and $\mathbf{w}$ such as

$\mathbf{v} \geq 0$

$\mathbf{w} \geq 0$

$\mathbf{v}^{\mathsf{T}}\mathbf{w} \geq 0$

Note, that solving the variational inequalities is equivalent to solve an LCP problem for $n = 0, \ldots, N$.

4.2 PSOR method

The LCP reformulation for the pricing problem requires to solve a system of linear equations subject to complementary conditions. Therefore, we consider the problem of solving the linear equation $\mathbf{A}\mathbf{x} = \mathbf{b}$ for $\mathbf{A} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^n$. Suppose that $\mathbf{M} = \mathbf{A} + \mathbf{N}$ is a non-singular matrix. Then, the linear equation can be rewritten as

$$\mathbf{M}\mathbf{x} = \mathbf{N}\mathbf{x} + \mathbf{b} \tag{4.26}$$

or as

$$\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{M}^{-1}\mathbf{b} \tag{4.27}$$

where $\mathbf{B} := \mathbf{M}^{-1}\mathbf{N} = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A}$. Clearly, equation (4.27) constitute fixed-point problem[7] which leads to the iterative method

$$\mathbf{x}^k = \mathbf{B}\mathbf{x}^{k-1} + \mathbf{M}^{-1}\mathbf{b} \tag{4.28}$$

Epperson[7] showed that iterative method (4.28) converges given an initial guess $\mathbf{x}^0$ when

$$\rho(\mathbf{B}) < 1$$

where $\rho(\mathbf{B})$ is the spectral radius of matrix $\mathbf{B}$. Different choices of matrix $\mathbf{M}$ will yield to different methods with different convergence rate. The Jacobi method defines $\mathbf{M}$ and $\mathbf{N}$ such as $\mathbf{A} = \mathbf{M} - \mathbf{N} = \mathbf{D} - (\mathbf{L} + \mathbf{U})$, yielding the iterative method

$$\mathbf{D}\mathbf{x}^k = (\mathbf{L} + \mathbf{U})\mathbf{x}^{k-1} + \mathbf{M}^{-1}\mathbf{b} \tag{4.29}$$

According to Epperson[7], the Jacaobi method has the slowest convergence of all the other iterative methods that we could come with. Similarly, the Gauss-Seidel method is given by defining $\mathbf{M} := \mathbf{D} - \mathbf{L}$ and $\mathbf{N} := \mathbf{U}$.

$$(\mathbf{D} - \mathbf{L})\mathbf{x}^k = \mathbf{U}\mathbf{x}^{k-1} + \mathbf{M}^{-1}\mathbf{b} \tag{4.30}$$

A generalization of the Gauss-Seidel (4.30) method is given by is the successive over-relaxation (SOR) method. The SOR method defines M and N as

$$\mathbf{M} := \frac{1}{\omega}\mathbf{D} - \mathbf{L}, \qquad \mathbf{N} = \left(\frac{1}{\omega} - 1\right)\mathbf{D} + \mathbf{U}$$

where ω is the relaxation parameter. Thus, resulting in the iterative method

$$\left(\frac{1}{\omega}\mathbf{D} - \mathbf{L}\right)\mathbf{x}^k = \left(\left(\frac{1}{\omega} - 1\right)\mathbf{D} + \mathbf{U}\right)\mathbf{x}^{k-1} + \mathbf{b} \tag{4.31}$$

Note that for $\omega = 1$, the Gauss-Seidel (4.31) method is obtained. Suppose, we want to solve $\mathbf{A}\mathbf{v} = \mathbf{b}$ for $\mathbf{A}$ given as in (4.24), $\mathbf{v} \in \mathbb{R}^M$, and $\mathbf{b} \in \mathbb{R}^M$. Then, a componentwise version of the (4.31) is given by

$$\frac{1}{\omega}(1 + 2\alpha)v_i^k = \left(\frac{1}{\omega} - 1\right)(1 + 2\alpha)v_i^{k-1} + \alpha(v_{i-1}^k + v_{i+1}^{k-1}) + b_i \tag{4.32}$$

for $\alpha = \lambda\theta$ and $i = 1, \ldots, M$. Moreover, an iterative algorithm will be given as

Algorithm 4.3 SOR for the theta method.

1: **for** $k = 1, \ldots$ **do**
2: **for** $i = 1, \ldots, M$ **do**
3: $r_i^k = (\alpha(v_{i-1}^k + v_{i+1}^{k-1}) + b_i)/(1 + 2\alpha)$
4: $v_i^k = v_i^{k-1} + \omega(r_i^k - v_i^{k-1})$

The project SOR (PSOR) method is a slight modification of SOR method in which the positivity of v_i^k is enforced at line number 5 of (4.3).

Algorithm 4.4 PSOR for the theta method.

1: **for** $k = 1, \ldots$ **do**
2: **for** $i = 1, \ldots, M$ **do**
3: $r_i^k = (\alpha(v_{i-1}^k + v_{i+1}^{k-1}) + b_i)/(1 + 2\alpha)$
4: $v_i^k = \max\left\{0, v_i^{k-1} + \omega(r_i^k - v_i^{k-1})\right\}$

Moreover, Seydel et al. [14] and Dewynne et al. [15] showed that the PSOR method is equivalent to LCP algorithm (4.2). Moreover, they also showed that the fasted convergence is given by having $\omega = 1$. Putting algorithm (4.1), (4.2) and (4.4) together, we have

Algorithm 4.5 Iterative method for solving heat diffusion variational inequalities.

Define $h(x, \tau)$ as in (4.5).
Define grid as in (4.11).
Set ω to 1.
Set ϵ to some value closes to zero.
$\mathbf{y}^n = h(x_i, 0)$ for $i = 1, \ldots, M$
for $n = 0, \ldots, N$ **do** ▷ *Beginning of LCP problem*
    Define $\mathbf{h}^{n+1}$ as in (4.20).
    Define $\hat{\mathbf{b}}^n$ as in (4.22).
    $v_i^0 := \max(y_i^n, h_i^{n+1})$ for $i = 1, \ldots, M$ ▷ *Beginning of PSOR method*
    for $k = 1, \ldots$ **do**
        for $i = 1, \ldots, M$ **do**
            $r_i^k := (\alpha(v_{i-1}^k + v_{i+1}^{k-1}) + b_i)/(1 + 2\alpha)$
            $v_i^k := \max\left\{0, v_i^{k-1} + \omega(r_i^k - v_i^{k-1})\right\}$
        if $||\mathbf{v}^k - \mathbf{v}^{k-1}||_2 \leq \epsilon$ **then**
            $\mathbf{v}^{\text{new}} := \mathbf{v}^k$
            Break out of the loop
    $\mathbf{y}^{n+1} := \mathbf{v}^k$

5 Numerical results

6 Conclusion

A Explicit scheme for Company transformation

The explicit scheme is given by

$$\frac{v_i^{n+1} - v_i^n}{\Delta t} - \frac{1}{2}\sigma^2 \frac{v_{i-1}^n - 2v_i^n + v_{i+1}^n}{(\Delta x)^2} - \left((r-\delta) - \frac{\sigma^2}{2} - \frac{\bar{S}_f^{n+1} - \bar{S}_f^n}{\Delta t \bar{S}_f^n}\right) \frac{v_{i+1}^n - v_{i-1}^n}{2\Delta x} + r v_i^n = 0$$

for $i = 1 \ldots, M$ and $n = 0, \ldots, N$.

$$v_i^{n+1} = a v_{i-1}^n + b v_i^n + c v_{i+1}^n + \frac{\bar{S}_f^{n+1} - \bar{S}_f^n}{2\Delta x \bar{S}_f^n}(v_{i+1}^n - v_{i-1}^n)$$

where

$$\begin{aligned}
\lambda =& \frac{\Delta t}{\Delta x^2} \\
a =& \frac{\lambda}{2}\left(\sigma^2 - \left(r - \delta - \frac{\sigma^2}{2}\right)\Delta x\right) \\
b =& 1 - \sigma^2 \lambda - r\Delta t \\
c =& \frac{\lambda}{2}\left(\sigma^2 + \left(r - \delta - \frac{\sigma^2}{2}\right)\Delta x\right)
\end{aligned}$$

Moreover, the boundary conditions

$$\begin{aligned}
&\textbf{Call:} \qquad v_0^n = 0, \quad v_{M+1}^n = \bar{S}_f^n - 1 \\
&\textbf{Put:} \qquad v_0^n = 1 - \bar{S}_f^n, \quad v_{M+1}^n = 0
\end{aligned}$$

for $n = 0, \ldots, N$. Moreover, the contact point condition is approximated using central finite difference

$$\begin{aligned}
&\textbf{Call:} \qquad \frac{v_{M+2}^n - v_M^n}{2\Delta x^2} = \frac{\partial v}{\partial x}(0,t) + O(\Delta x^2) \\
&\textbf{Put:} \qquad \frac{v_1^n - v_{-1}^n}{2\Delta x} = \frac{\partial v}{\partial x}(0,t) + O(\Delta x^2)
\end{aligned}$$

for $n = 0, \ldots, N$. Moreover, the contact point condition is approximated using central finite

difference

$$\textbf{Call:} \qquad \frac{v_{M+2}^n - v_M^n}{2\Delta x^2} = \bar{S}_f^n \tag{A.1}$$

$$\textbf{Put:} \qquad \frac{v_1^n - v_{-1}^n}{2\Delta x} = -\bar{S}_f^n \tag{A.2}$$

By combining the central difference approximation for the PDE, the boundary conditions and the contact point, it is obtained

$$\textbf{Call:} \qquad v_M^n =$$

$$\textbf{Put:} \qquad v_1^n = \alpha - \beta \bar{S}_f^n$$

Algorithm A.1 Explicit method for put options

for $i = 0, \ldots, M+1$ **do**

$v_i^0 = 0$

$\bar{S}_f^0 = K$

$a = \frac{\lambda}{2}\left(\sigma^2 - \left(r - \delta - \frac{\sigma^2}{2}\right)\Delta x\right)$

$b = 1 - \sigma^2\lambda - r\Delta t$

$c = \frac{\lambda}{2}\left(\sigma^2 + \left(r - \delta - \frac{\sigma^2}{2}\right)\Delta x\right)$

$\alpha = 1 + \frac{r\Delta x^2}{\sigma^2}$

$\beta = 1 + \Delta x + \frac{1}{2}\Delta x^2$

for $n = 0, \ldots, N$ **do**

$d^n = \frac{\alpha - (av_0^n + bv_1^0 + cv_2^n - (v_2^n - v_0^n)/(2\Delta x))}{(v_2^n - v_0^n)/(2\Delta x) + \beta\bar{S}_f^n}$

$\bar{S}_f^{n+1} = d^n\bar{S}_f^n$

$a^n = a - \frac{\bar{S}_f^{n+1} - \bar{S}_f^n}{2\Delta x\bar{S}_f^n}$

$c^n = c - \frac{\bar{S}^{n+1} - \bar{S}^n}{2\Delta x\bar{S}_f^n}$

$v_0^{n+1} = 1 - \bar{S}_f^{n+1}$

$v_1^{n+1} = \alpha - \beta\bar{S}_f^{n+1}$

$v_{M+1}^{n+1} = 0$

for $i = 2, \ldots, M$ **do**

$v_i^{n+1} = a^n v_{i-1}^n + bv_i^n + c^n v_{i+1}^n$

B Python implementation

The following python code is simplified version of the final implementation. A complete version can be found at `https://github.com/Alcruz/math4062-dissertation`.

Listing B.1: Base classses for options solver.

```
class Option(ABC):
    def __init__(self, type: OptionType, K: float, T: float):
        self.type = type
        self.K = K # Strike price
```

```
            self.T = T # Maturity

        @abstractmethod
        def payoff(self, S: float):
            pass
    class CallOption(Option):
        def __init__(self, K: float, T: float):
            super().__init__(OptionType.CALL, K, T)
        def payoff(self, S: float):
            return np.maximum(S - self.K, 0)
    class PutOption(Option):
        def __init__(self, K: float, T: float):
            super().__init__(OptionType.PUT, K, T)
        def payoff(self, S: float):
            return np.maximum(self.K - S, 0)
    class Solver(ABC):
        def __init__(self,
            option: Option,
            r: float,  # risk-free interest rate
            sigma: float,  # sigma price volatitliy
            dx: float,  # grid resolution along x-axis
            dt: float,  # grid resolution along t-axis
            delta # dividends
        ) -> None:
            self.option = option
            self.r = r
            self.sigma = sigma
            self.dx = dx
            self.dt = dt
            self.delta = delta
```

Listing B.2: Explicit solver for Nielsen transformation.

```
    class ExplicitSolver(Solver):
        def __init__(self,
            option: Option,
```

```
            r: float,  # risk-free interest rate
            sigma: float,  # sigma price volatitliy
            dx: float,  # grid resolution along x-axis
            dt: float,  # grid resolution along t-axis
            delta # dividends
        ) -> None:
            self.option = option
            self.r = r
            self.sigma = sigma
            self.dx = dx
            self.dt = dt
            self.delta = delta
            lambd = dt / np.power(dx, 2)
            alpha = dt / dx
            self.A = 0.5 * np.power(sigma*self.x_axis, 2) * lambd - 0.5
    * self.x_axis * ((r-delta) - (1/dt)) * alpha
            self.B = 1 - np.power(sigma*self.x_axis, 2) * lambd - r*dt
            self.C = 0.5 * np.power(sigma*self.x_axis, 2) * lambd + 0.5
    * self.x_axis * ((r-delta) - (1/dt)) * alpha
        def solve(self):
            V = np.zeros_like(self.x_axis)
            S_bar = self.option.K
            for _ in np.arange(0, self.option.T, self.dt):
                D = 0.5*self.x_axis/self.dx
                D[1:-1] *= (V[2:]-V[:-2]) * (1/S_bar)

                S_bar = self.compute_time_iteration(V, D)
            return S_bar*self.x_axis, V, S_bar
        @abstractmethod
        def compute_time_iteration(self, V: np.ndarray, D: np.ndarray):
            pass
     class CallOptionExplicitSolver(ExplicitSolver):
        def __init__(self,
            option: CallOption,
            r: float,  # risk-free interest rate
```

```
            sigma: float,  # sigma price volatitliy
            dx: float,  # grid resolution along x-axis
            dt: float,  # grid resolution along t-axis
            delta=0  # dividends
        ):
            self.x_axis = np.arange(0, 1+dx, dx)
            super().__init__(option, r, sigma, dx, dt, delta)
        def compute_time_iteration(self, V: np.ndarray, D: np.ndarray):
            S_bar = self.option.K + self.A[-2]*V[-3] + self.B[-2]*V[-2]
    + self.C[-2]*V[-1]
            S_bar /= 1 - self.dx - D[-2]

            V[1:-2] = self.A[1:-2]*V[:-3] + self.B[1:-2]*V[1:-2] \
                + self.C[1:-2]*V[2:-1] + D[1:-2]*S_bar
            V[-2] = (1-self.dx)*S_bar - self.option.K
            V[-1] = S_bar - self.option.K
            return S_bar
     class PutOptionExplicitSolver(ExplicitSolver):
        def __init__(self,
            option: PutOption,
            r: float,  # risk-free interest rate
            sigma: float,  # sigma price volatitliy
            dx: float,  # grid resolution along x-axis
            dt: float,  # grid resolution along t-axis
            x_max: 2.,  # sufficiently large value for x
            delta=0  # dividends
        ):
            self.x_axis = np.arange(1, x_max+dx, dx)
            super().__init__(option, r, sigma, dx, dt, delta)
        def compute_time_iteration(self, V: np.ndarray, D: np.ndarray)
    -> float:
            S_bar = self.option.K - (self.A[1]*V[0] + self.B[1]*V[1] +
    self.C[1]*V[2])
            S_bar /= D[1] + 1 + self.dx
            V[2:-1] = self.A[2:-1]*V[1:-2] + self.B[2:-1] * \
```

```
                V[2:-1] + self.C[2:-1]*V[3:] + D[2:-1]*S_bar
            V[0] = self.option.payoff(S_bar)
            V[1] = self.option.payoff((1+self.dx)*S_bar)
            return S_bar
```

Listing B.3: Implicit solver for Nielsen transformation

```
class ImplicitSolver(Solver):
    def __init__(self,
        option: Option,
        r: float,  # risk-free interest rate
        sigma: float,  # sigma price volatitliy
        dx: float,  # grid resolution along x-axis
        dt: float,  # grid resolution along t-axis
        delta=0, # dividends
    ) -> None:
        self.option = option
        self.r = r
        self.sigma = sigma
        self.dx = dx
        self.dt = dt
        self.delta = delta
        self.lambd = self.dt/np.power(self.dx, 2)
        self.kappa = self.dt/self.dx
        self.alpha = 1 + self.lambd*np.power(self.sigma*self.x_axis,
 2) + self.r*self.dt
        self.M = self.x_axis.size
    def beta(self, S, S_bar):
        return -0.5*self.lambd*np.power(self.sigma, 2)*np.power(self
.x_axis, 2) + 0.5*self.kappa*self.x_axis*((self.r-self.delta) - (
S_bar - S)/(self.dt*S))
    def gamma(self, S, S_bar):
        return -0.5*self.lambd*np.power(self.sigma, 2)*np.power(self
.x_axis, 2) - 0.5*self.kappa*self.x_axis*((self.r-self.delta) - (
S_bar - S)/(self.dt*S))
    @abstractmethod
```

```
25          def solve_non_linear_system(self, V, S_bar) -> float:
26              pass
27          def solve(self):
28              S_bar = self.option.K
29              V = np.zeros_like(self.x_axis)
30              for _ in np.arange(0, self.option.T, self.dt):
31                  S_bar = self.solve_non_linear_system(V, S_bar)
32              return S_bar*self.x_axis, V, S_bar
33      class CallOptionImplicitSolver(ImplicitSolver):
34          def __init__(self,
35              option: Option,
36              r: float,  # risk-free interest rate
37              sigma: float,  # sigma price volatitliy
38              dx: float,  # grid resolution along x-axis
39              dt: float,  # grid resolution along t-axis
40              delta=0, #dividends
41              maxiter=1000,
42              tolerance=1e-24,
43              method='lm'
44          ) -> None:
45              self.x_axis = np.arange(0, 1+dx, dx)
46              super().__init__(option, r, sigma, dx, dt, delta)
47              self.maxiter=maxiter
48              self.tolerance=tolerance
49              self.method=method
50          def jacobian(self, y, S_bar):
51              p, s, = y[:-1], y[-1]
52              beta = self.beta(s, S_bar)
53              gamma = self.gamma(s, S_bar)
54              dgamma_ds = - 0.5 * (1/self.dx) * self.x_axis * S_bar / s**2
55              dbeta_ds = 0.5 * (1/self.dx) * self.x_axis * S_bar / s**2
56              retVal = diags([self.alpha[2:], beta[1:-1], gamma[1:-1]],
57                             [-1, 0, 1], shape=(self.M-2, self.M-2)).toarray
    ()
58              retVal[-1, :] = 0
```

```
            retVal[:, -1] = 0
            retVal[0, -1] = dbeta_ds[1]*p[0] + dgamma_ds[1]*p[1]
            retVal[1:-2, -1] = dbeta_ds[2:-3]*p[1:-1] + dgamma_ds[2:-3]*
    p[2:]
            retVal[-2, -1] = dbeta_ds[-3]*p[-2]
            retVal[-2, -1] -= dgamma_ds[-3]*self.option.payoff((1-self.
    dx)*s) + gamma[-3]*(1 - self.dx)
            retVal[-1, -2] = self.alpha[-2]
            retVal[-1, -1] -= dgamma_ds[-2]*self.option.payoff(s) +
    gamma[-2] + dbeta_ds[-2]*self.option.payoff((1-self.dx)*s) - beta
    [1]*(1-self.dx)
            return retVal
        def system(self, y, b, S_bar):
            *v, s = y
            beta = self.beta(s, S_bar)
            gamma = self.gamma(s, S_bar)
            A = diags([self.alpha[2:-1], beta[1:-2], gamma[1:-3]],
                    [-1, 0, 1], shape=(self.M-2, self.M-3)).toarray()
            f = b[1:-1]
            f[-2] -= gamma[-3]*((1-self.dx)*s-self.option.K)
            f[-1] -= gamma[-2]*(s-self.option.K) + beta[-2]*((1-self.dx)
    *s - self.option.K)
            res = A@v - f
            return res
        def solve_non_linear_system(self, V: np.ndarray, S_bar: np.
    ndarray):
            *V[1:-2], S_bar = root(
                lambda y: self.system(y, np.copy(V[:]), S_bar),
                # jac=lambda y: self.jacobian(y, S_bar),
                x0=np.concatenate([V[1:-2], [S_bar]]),
                method=self.method,
                options=dict(xtol=self.tolerance, maxiter=self.maxiter)
            )['x']
            V[-2] = (1-self.dx)*S_bar-self.option.K
            V[-1] = S_bar-self.option.K
```

```
88            return S_bar
89    class PutOptionImplicitSolver(ImplicitSolver):
90        def __init__(self,
91            option: Option,
92            r: float,  # risk-free interest rate
93            sigma: float,  # sigma price volatitliy
94            dx: float,  # grid resolution along x-axis
95            dt: float,  # grid resolution along t-axis
96            delta=0, # dividends
97            x_max=2,
98            maxiter=1000,
99            tolerance=1e-24,
100           method='lm'
101       ) -> None:
102           self.x_axis = np.arange(1, x_max+dx, dx)
103           super().__init__(option, r, sigma, dx, dt, delta)
104           self.maxiter=maxiter
105           self.tolerance=tolerance
106           self.method=method
107       def jacobian(self, y, S_bar):
108           p, s, = y[:-1], y[-1]
109           beta = self.beta(s, S_bar)
110           gamma = self.gamma(s, S_bar)
111
112           dgamma_ds = - 0.5 * (1/self.dx) * self.x_axis * S_bar / s**2
113           dbeta_ds = 0.5 * (1/self.dx) * self.x_axis * S_bar / s**2
114
115           retVal = diags([beta[3:-1], self.alpha[2:-1], gamma[1:-1]],
116                          [-2, -1, 0], shape=(self.M-2, self.M-2)).toarray
    ()
117           retVal[-1, :] = 0
118           retVal[:, -1] = 0
119           retVal[0, -1] = dgamma_ds[1]*p[0] + dbeta_ds[1] * \
120               (self.option.K - S_bar) - beta[1] - self.alpha[1]*(1+
    self.dx)
```

```
            retVal[1, -1] = dgamma_ds[2]*p[1] + dbeta_ds[2] * \
                (self.option.K - (1+self.dx)*S_bar) - beta[2]*(1+self.dx
)
            retVal[2:-1, -1] = dbeta_ds[3:-2]*p[:-2] + dgamma_ds[3:-2]*p
[2:]
            retVal[-1, -3] = beta[-2]
            retVal[-1, -2] = self.alpha[-2]
            retVal[-1, -1] = dbeta_ds[-2]*p[-2]
            return retVal
        def system(self, y, b, S_bar):
            p, s, = y[:-1], y[-1]
            _beta = self.beta(s, S_bar)
            _gamma = self.gamma(s, S_bar)
            A = diags([_beta[3:-1], self.alpha[2:-1], _gamma[1:-2]],
                    [-2, -1, 0], shape=(self.M-2, self.M-3)).toarray()
            f = b[1:-1]
            f[0] -= _beta[1]*(self.option.K-s) + self.alpha[1]*((self.
option.K-(1+self.dx)*s))
            f[1] -= _beta[2]*(self.option.K-(1+self.dx)*s)
            res = A@p - f
            return res
        def solve_non_linear_system(self, V: np.ndarray, S_bar: np.
ndarray):
            *V[2:-1], S_bar = root(
                lambda y: self.system(y, np.copy(V[:]), S_bar),
                # jac=lambda y: self.jacobian(y, S_bar),
                x0=np.concatenate([V[2:-1], [S_bar]]),
                method=self.method,
                options=dict(xtol=self.tolerance, maxiter=self.maxiter)
            )['x']
            V[0] = self.option.payoff(S_bar)
            V[1] = self.option.payoff((1+self.dx)*S_bar)
            return S_bar
```

C Code for numerical experiments

D Code for convergence analysis

1 Setup

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import option
from frontfixing import nielsen, company
import lcp

plt.style.use('ggplot')

K = 1
T = 1
put = option.put(K, T)
r = 0.1
sigma = 0.2
delta = 0.03
S_max = 2
S_min = 0
S_test = np.arange(S_min, S_max, step=0.2)

def plot_space_convergence(title, dx, V_pred):
    error = []
    for i in range(len(dx)-1):
        error.append(np.linalg.norm(V_pred[i+1][::2]-V_pred[i], ord=np.inf))
    log_error = np.log(error)
    log_dx = np.log(dx[:-1])
    p = np.polyfit(log_dx, log_error, deg=1)
    f = np.poly1d(p)
    log_dx_axis = np.linspace(np.min(log_dx), np.max(log_dx))
    plt.title(title)
    plt.scatter(log_dx, np.log(error), marker='x', color='black')
    plt.plot(log_dx_axis, f(log_dx_axis), '--k', linewidth=1)
    plt.ylabel("log error")
    plt.xlabel("log dx")
    text = plt.annotate(f"{p[0]:.2}x {'-' if p[1] <= 0 else '+'} {np.abs(p[1]):.
↪2}", # this is the text
```

```
                (log_dx[1],log_error[1]), # these are the coordinates to␣
 ↪position the label
                 textcoords="offset points", # how to position the text
                 xytext=(0,5), # distance from text to points (x,y)
                 ha='center')
    text.set_rotation(22)

def plot_time_convergence(title, dt, V_pred):
    error = []
    for i in range(len(dt)-1):
        error.append(np.linalg.norm(V_pred[i+1]-V_pred[i], ord=np.inf))
    log_error = np.log(error)
    log_dt = np.log(dt[:-1])
    plt.scatter(log_dt, np.log(error), marker='x', color='black')
    p = np.polyfit(log_dt, log_error, deg=1)
    f = np.poly1d(p)
    log_dt_axis = np.linspace(np.min(log_dt), np.max(log_dt))
    plt.title(title)
    plt.plot(log_dt_axis, f(log_dt_axis), '--k', linewidth=1)
    plt.ylabel("log error")
    plt.xlabel("log dt")
    text = plt.annotate(f"{p[0]:.2}x {'-' if p[1] <= 0 else '+'} {np.abs(p[1]):.
 ↪2}", # this is the text
                (log_dt[1],log_error[1]), # these are the coordinates to␣
 ↪position the label
                 textcoords="offset points", # how to position the text
                 xytext=(0,5), # distance from text to points (x,y)
                 ha='center')
    text.set_rotation(22)

def space_convergence_analysis(method, dx, dt, **kwargs):
    V_pred = []
    for _, dx_i in enumerate(dx):
        res = method(dx=dx_i, dt=dt, **kwargs)
        V_pred.append(res[1][:])
    return V_pred

def time_convergence_analysis(method, dx, dt, **kwargs):
    V_pred = []
    for i, dt_i in enumerate(dt):
        res = method(dx=dx, dt=dt_i, **kwargs)
        V_pred.append(res[1][:])
    return V_pred
```

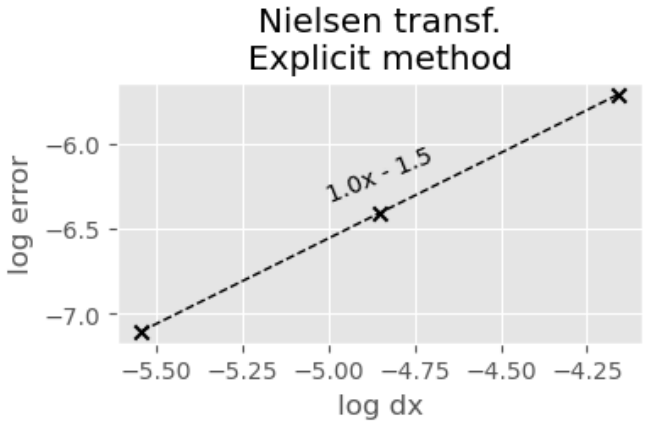
2 Front fixing method

2.1 Nielsen transformation

2.1.1 Explicit

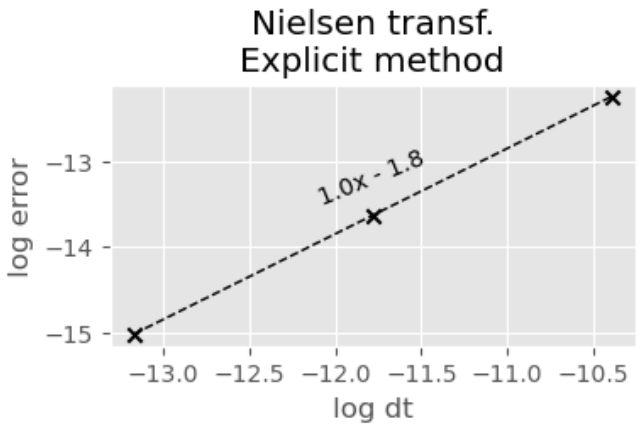
Space

```
dx = np.array([1/64, 1/128, 1/256, 1/512])
dt = 0.5*dx[-1]**2
V_pred = space_convergence_analysis(nielsen.solve_explicitly, dx, dt,␣
  ↪option=put, r=r, sigma=sigma, x_max=2, delta=delta)
plt.figure(figsize=(4,2))
plot_space_convergence("Nielsen transf.\nExplicit method", dx, V_pred)
```



Time

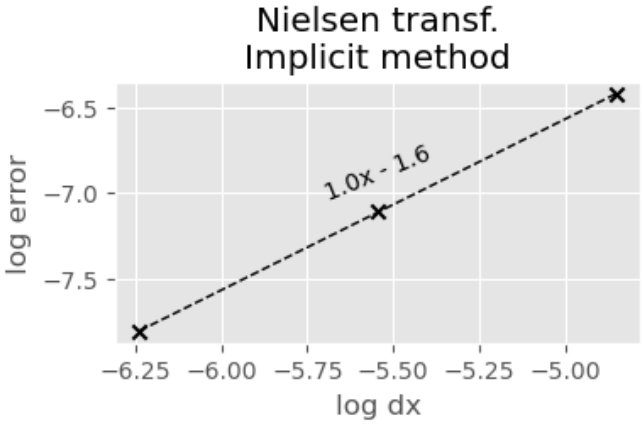
```
dx = 1/128
dt = 0.5*np.power([1/128, 1/256, 1/512, 1/1024],2)
V_pred = time_convergence_analysis(nielsen.solve_explicitly, dx, dt, option␣
  ↪=put, r=r, sigma=sigma, x_max=3,delta=delta)
plt.figure(figsize=(4,2))
plot_time_convergence("Nielsen transf.\nExplicit method", dt, V_pred)
```



2.1.2 Implicit

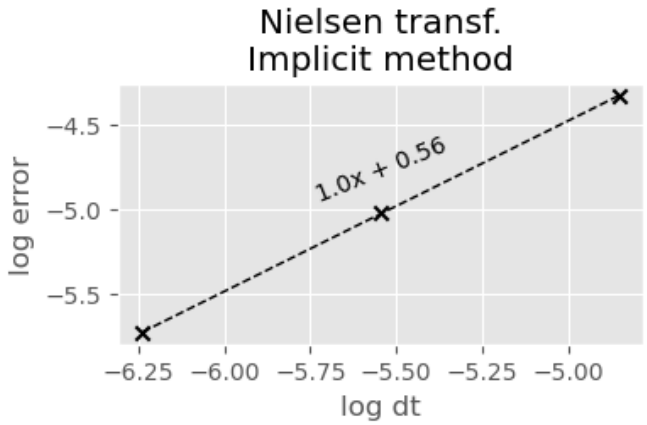
Space

```
dx = [1/128, 1/256, 1/512, 1/1024]
dt = 1/1024
V_pred = space_convergence_analysis(nielsen.solve_implicitly, dx, dt,␣
  ↪option=put, r=r, sigma=sigma, x_max=2, delta=delta, tolerance=1e-21,␣
  ↪maxiter=10*6)
plt.figure(figsize=(4,2))
plot_space_convergence("Nielsen transf.\nImplicit method", dx, V_pred)
```



Time

```
dx = 1/32
dt = [1/128, 1/256, 1/512, 1/1024]
V_pred = time_convergence_analysis(nielsen.solve_implicitly, dx, dt, option␣
 ↪=put, r=r, sigma=sigma, x_max=2, delta=delta, tolerance=1e-21)
plt.figure(figsize=(4,2))
plot_time_convergence("Nielsen transf.\nImplicit method", dt, V_pred)
```

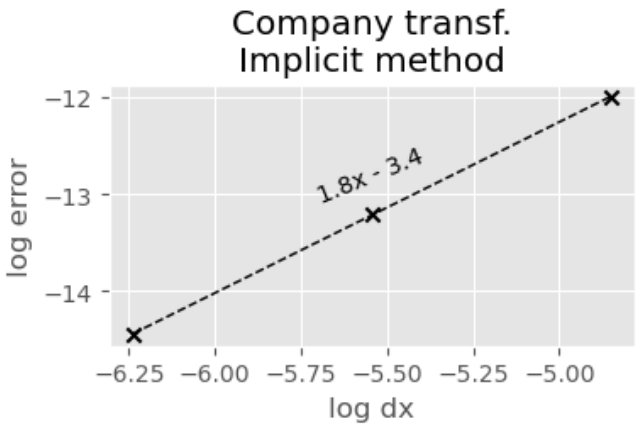


2.2 Company transformation

2.2.1 Explicit

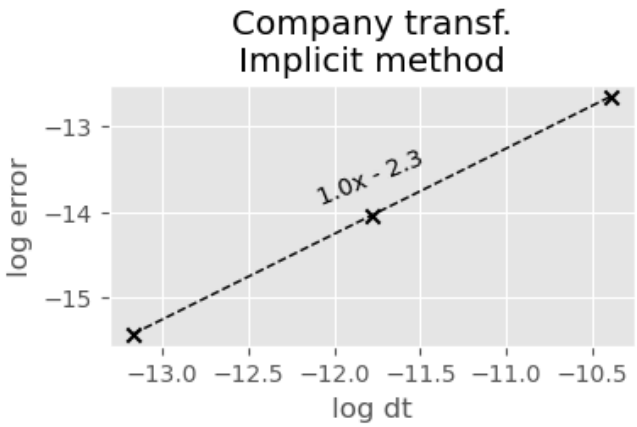
Space

```
dx = [1/128, 1/256, 1/512, 1/1024]
dt = 0.5*dx[-1]**2
V_pred = space_convergence_analysis(company.solve_explicitly, dx, dt,␣
 ↪option=put, r=r, sigma=sigma, x_max=2, delta=delta)
plt.figure(figsize=(4,2))
plot_space_convergence("Company transf.\nImplicit method", dx, V_pred)
```



Time

```
dx = 1/128
dt = 0.5*np.array([1/128, 1/256, 1/512, 1/1024])**2
V_pred = time_convergence_analysis(nielsen.solve_explicitly, dx, dt,␣
  ↪option=put, r=r, sigma=sigma, x_max=3)
plt.figure(figsize=(4,2))
plot_time_convergence("Company transf.\nImplicit method", dt, V_pred)
```

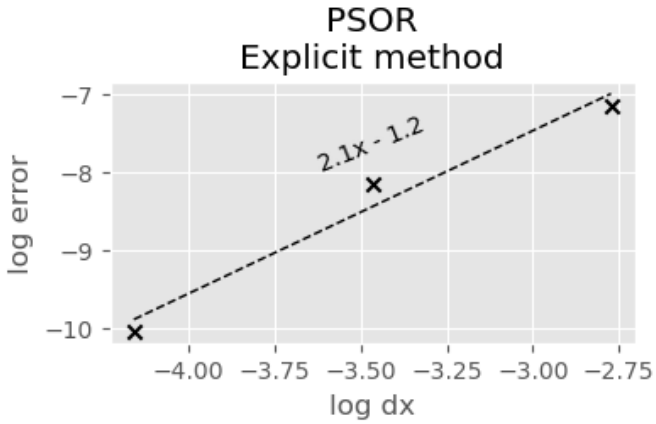


3 LCP + PSOR

3.1 Explicit

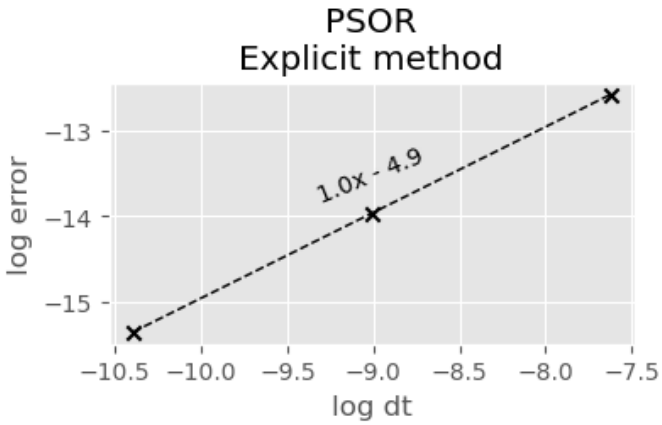
3.1.1 Space

```
dx = [1/16, 1/32, 1/64, 1/128]
dt = 0.5*dx[-1]**2
V_pred = space_convergence_analysis(lcp.solve, dx, dt, option=put, r=r,␣
 ↪sigma=sigma, x_max=2, delta=delta, theta=0)
plt.figure(figsize=(4,2))
plot_space_convergence("PSOR\nExplicit method", dx, V_pred)
```



Time

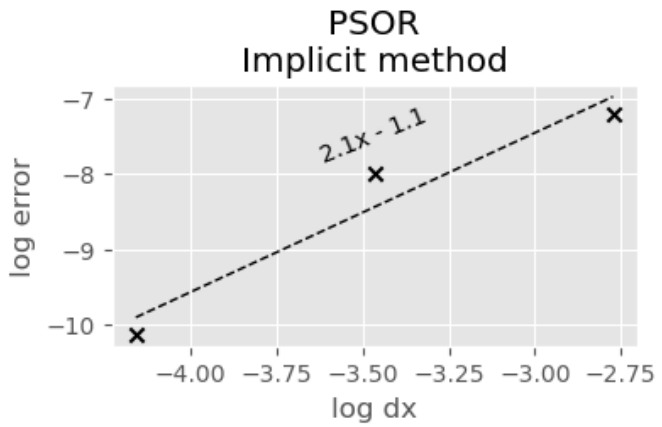
```
dx = 1/32
dt = 0.5*np.array([1/32, 1/64, 1/128, 1/256])**2
V_pred = time_convergence_analysis(lcp.solve, dx, dt, option=put, r=r,␣
 ↪sigma=sigma, theta=0, delta=delta)
plt.figure(figsize=(4,2))
plot_time_convergence("PSOR\nExplicit method", dt, V_pred)
```



3.2 Implicit

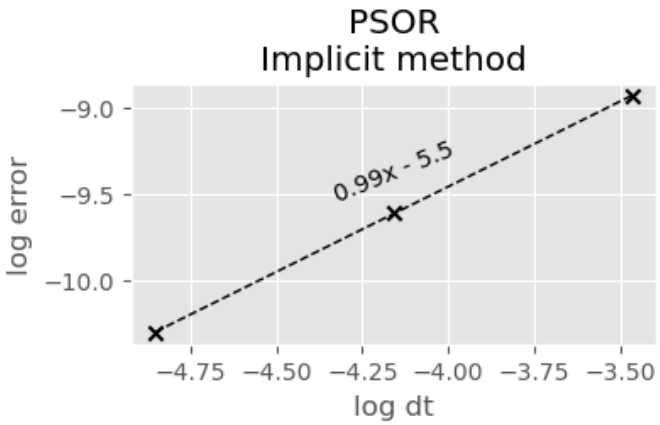
Space

```
dx = [1/16, 1/32, 1/64, 1/128]
dt = 1/4
V_pred = space_convergence_analysis(lcp.solve, dx, dt, option=put, r=r,␣
 ↪sigma=sigma, x_max=2, delta=delta, theta=1)
plt.figure(figsize=(4,2))
plot_space_convergence("PSOR\nImplicit method", dx, V_pred)
```



Time

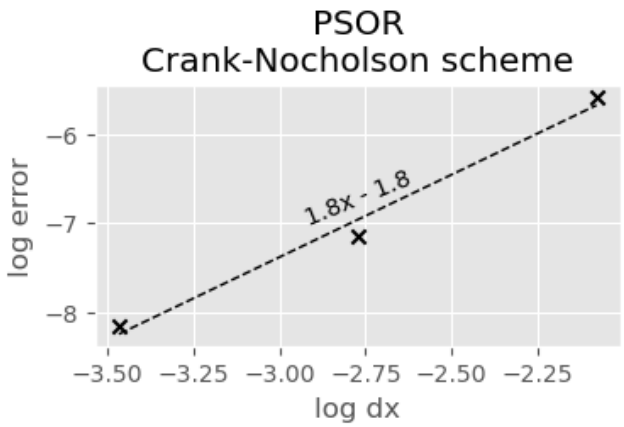
```
dx = 1/16
dt = [1/32, 1/64, 1/128, 1/256]
V_pred = time_convergence_analysis(lcp.solve, dx, dt, option=put, r=r,␣
 ↪sigma=sigma, theta=1, delta=delta)
plt.figure(figsize=(4,2))
plot_time_convergence("PSOR\nImplicit method", dt, V_pred)
```



3.3 Crank nicholson

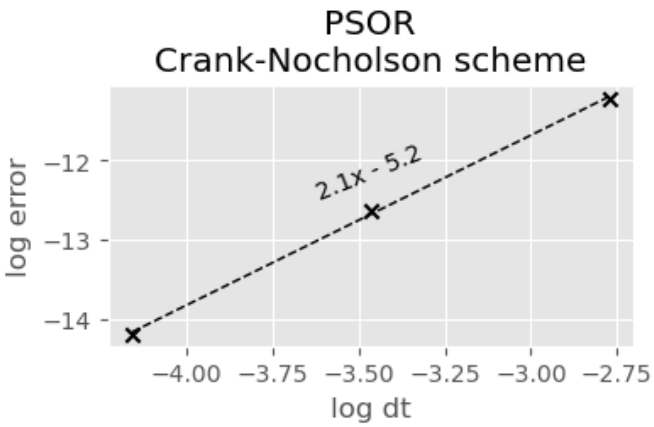
Space

```
dx = [1/8, 1/16, 1/32, 1/64]
dt = 1/64
V_pred = space_convergence_analysis(lcp.solve, dx, dt, option=put, r=r,␣
 ↪sigma=sigma, x_max=2, delta=delta, theta=.5)
plt.figure(figsize=(4,2))
plot_space_convergence("PSOR\nCrank-Nocholson scheme", dx, V_pred)
```



Time

```
dx = 1/16
dt = [1/16, 1/32, 1/64, 1/128]
V_pred = time_convergence_analysis(lcp.solve, dx, dt, option=put, r=r,␣
 ↪sigma=sigma, theta=.5, delta=delta)
plt.figure(figsize=(4,2))
plot_time_convergence("PSOR\nCrank-Nocholson scheme", dt, V_pred)
```



References

[1] Fischer Black and Myron Scholes. "The Pricing of Options and Corporate Liabilities". In: *Journal of Political Economy* 81.3 (1973), pp. 637–654. ISSN: 00223808, 1537534X. URL: http://www.jstor.org/stable/1831029 (visited on 08/21/2023).

[2] Michael J. Brennan and Eduardo S. Schwartz. "The Valuation of American Put Options". In: *The Journal of Finance* 32.2 (1977), pp. 449–462. ISSN: 00221082, 15406261. URL: http://www.jstor.org/stable/2326779 (visited on 08/22/2023).

[3] Rafael Company, Vera Egorova, and Lucas Jódar. "Solving American Option Pricing Models by the Front Fixing Method: Numerical Analysis and Computing". In: *Abstract and Applied Analysis* 2014 (Apr. 2014). DOI: 10.1155/2014/146745.

[4] Richard W. Cottle and George B. Dantzig. "Complementary pivot theory of mathematical programming". In: *Linear Algebra and its Applications* 1.1 (1968), pp. 103–125. ISSN: 0024-3795. DOI: https://doi.org/10.1016/0024-3795(68)90052-9. URL: https://www.sciencedirect.com/science/article/pii/0024379568900529.

[5] John C. Cox, Stephen A. Ross, and Mark Rubinstein. "Option pricing: A simplified approach". In: *Journal of Financial Economics* 7.3 (1979), pp. 229–263. ISSN: 0304-405X. DOI: https://doi.org/10.1016/0304-405X(79)90015-1. URL: https://www.sciencedirect.com/science/article/pii/0304405X79900151.

[6] J. N. Dewynne et al. "Some mathematical results in the pricing of American options". In: *European Journal of Applied Mathematics* 4.4 (1993), pp. 381–398. DOI: 10.1017/S0956792500001194.

[7] James F. Epperson. *An Introduction to Numerical Methods and Analysis*. 2nd. Wiley Publishing, 2013. ISBN: 1118367596.

[8] "Futures Industry Association. (January 16, 2020). Number of derivatives traded globally in 2019, by category (in billions) [Graph]." In: *Statista* (2019). URL: https://www.statista.com/statistics/380324/global-futures-and-options-volume-by-category.

[9] Jacqueline Huang. "American options and complementarity problems". English. In: *ProQuest Dissertations and Theses* (2000). Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2023-02-23, p. 139. URL: http://nottingham.idm.oclc.org/login?url=https://www.proquest.com/dissertations-theses/american-options-complementarity-problems/docview/304625569/se-2.

[10] H. G. Landau. "Heat conduction in a melting solid". In: *Quarterly of Applied Mathematics* 8 (1950), pp. 81–94. URL: https://api.semanticscholar.org/CorpusID:126349227.

[11] Robert C. Merton. "Theory of Rational Option Pricing". In: *The Bell Journal of Economics and Management Science* 4.1 (1973), pp. 141–183. ISSN: 00058556. URL: http://www.jstor.org/stable/3003143 (visited on 08/21/2023).

[12] Bjrn Nielsen, Ola Skavhaug, and Aslak Tveito. "Penalty and front-fixing methods for the numerical solution of American option problems". In: *Journal of Computational Finance* 5 (Sept. 2001). DOI: 10.21314/JCF.2002.084.

[13] Eduardo S. Schwartz. "The valuation of warrants: Implementing a new approach". In: *Journal of Financial Economics* 4.1 (1977), pp. 79–93. ISSN: 0304-405X. DOI: https://doi.org/10.1016/0304-405X(77)90037-X. URL: https://www.sciencedirect.com/science/article/pii/0304405X7790037X.

[14] Rüdiger U. Seydel. *Tools for Computational Finance*. Springer Berlin, Heidelberg, 2009. DOI: 10.1007/978-3-540-92929-1.

[15] Paul Wilmott, Sam Howison, and Jeff Dewynne. *The Mathematics of Financial Derivatives: A Student Introduction*. Cambridge University Press, 1995. DOI: 10.1017/CBO9780511812545.

[16] Lixin Wu and Yue-Kuen Kwok. "A front-fixing finite difference method for the valuation of American options". In: *Journal of Financial Engineering* 6.4 (1997), pp. 83–97.