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 assets. We focus on two important formulations of the pricing problem: the free boundary problem and the Black-Scholes variational inequality. In the free boundary formulation, we solve the Black-Scholes PDE with a moving boundary condition. By applying the front fixing method, we transform the free boundary problem to a domain where moving boundary is fixed. Then, we derived explicit and implicit schemes for the transformed PDE. On the other hand, for the variational inequality formulation, we solve the heat diffusion variational inequality derived from transforming the Black-Scholes PDE into the heat diffusion equation. By applying the theta method, the variational inequality is reformulated as a linear complementary problem. Then, we derived PSOR method to solve the LCP problem iteratively. Finally, by conducting a series of numerical experiments, we conclude that the front fixing explicit schemes derived are efficient and accurate, the front fixing implicit scheme produces significant approximation errors, and the PSOR method yields to accurate approximations though is relatively inefficient compared to the front fixing explicit schemes.

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[9]. 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[13] was the first to consider the Black-Scholes[1] model 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[16] 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 formulation of the pricing problem. 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[12], Wu et al.[18] 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 moving boundary into a fixed boundary. Multiple transformations have been proposed by Huang et al.[10], Nielsen et al.[15], 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.[15] 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.[15] 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, and consider underlying assets that have a continuous dividend yield, such as the S&P500. Furthermore, we consider the PSOR method proposed by Dewynne[6][7] to solve for the variational inequalities formulation of the pricing problem. Finally, we conduct a convergence analysis for each the schemes derived.

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.[15] and Company et al.[3] for call and put options with underlying assets featuring a continuous dividend yield. Moreover, we implemented explicit and implicit front fixing schemes for the Nielsen transformation method[15], an explicit front fixing scheme for Company transformation, and the theta PSOR scheme for the linear complementary problem proposed by Dewynne[7]. 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 the changes of variables proposed by Company et al.[3] and Nielsen et al.[15]. 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 in

section 3, and appendix A. In section 4, we explore a reformulation of the pricing problem as a variational inequality. Additionally, we derived the PSOR method as way to solve the linear complementary problem that arises from applying the theta method to the variational inequality. Finally, in the same section, we discuss the results obtained by explicit, implicit and Crank-Nicholson PSOR schemes.

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 a risky asset S_t such as a stock 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[13], 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[13] 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.[7] 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[13] 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

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

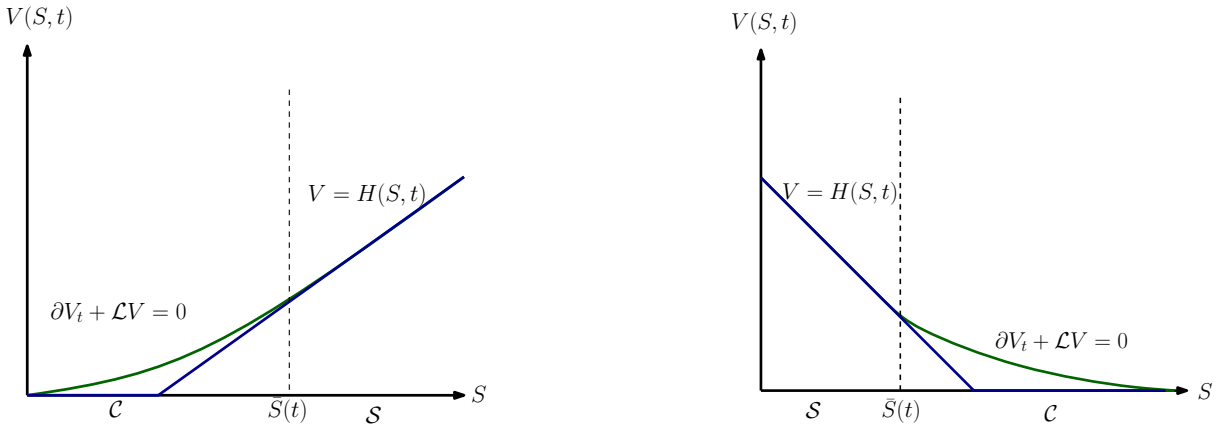
in which it is profitable for the holder to exercise the option, the continuation region

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

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 problem for 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. Also, we introduced the moving boundary condition $\bar{S}(t)$ for the Black-Scholes PDE. The moving boundary condition $\bar{S}(t)$ makes solving the Black-Scholes PDE more involving since the free boundary is an unknown function of time. 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 the moving boundary remains fixed as time changes. In this section, we explore two transformation based on the work of Nielsen et al.[15], and the work of Company and et al.[3].

2.2.1 Nielsen transformation

The Nielsen transformation suggests a really simple change of variable 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).

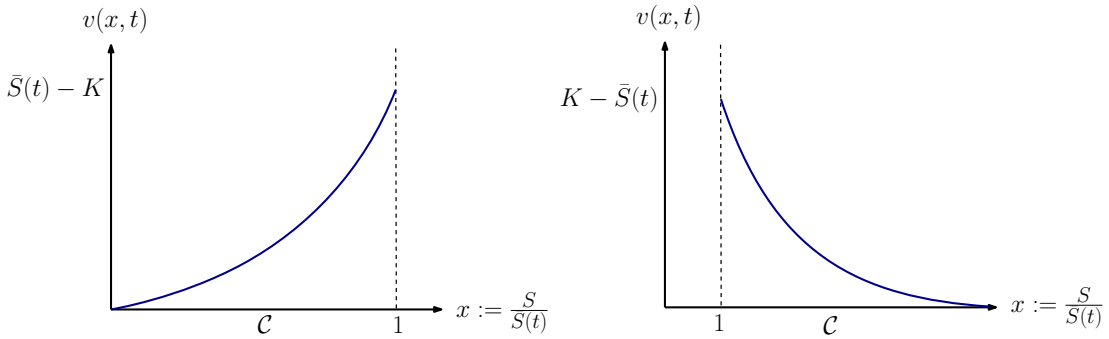
Firstly, we start with the Black-Scholes PDE which is defined at the interval $S \in (0, \bar{S}(t))$ for call options or the open interval $S \in (\bar{S}(t), \infty)$ for put options. Under the front fixing domain, the transformed PDE will be defined in the interval $x \in (0,1)$ for call and $x \in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

terms of $v(x,t)$

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

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

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 (See figure 2.1 and 2.2). Hence, 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 line segment in the payoff function (See figure 2.1). Hence, by the chain rule, the contact point condition of

$v(x,t)$ is

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

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

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 systems

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

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 variables 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 backward 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

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

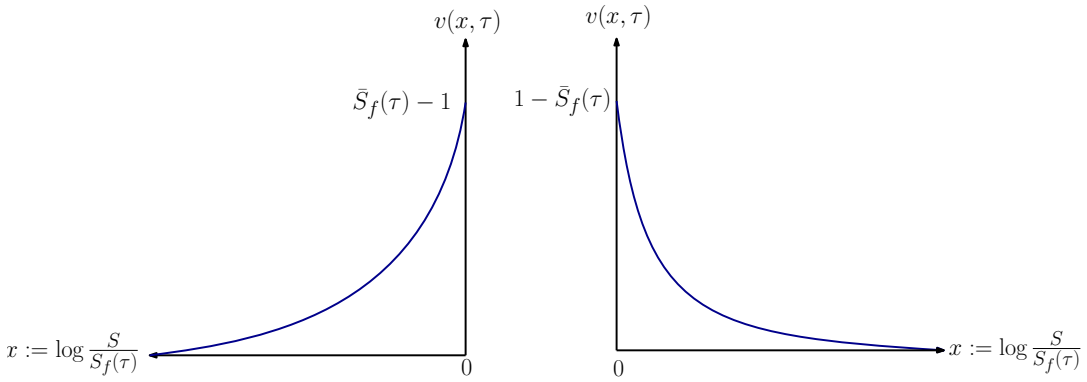
Put: $$\frac{\partial v}{\partial \tau} - \frac{1}{2}\sigma^2 \frac{\partial^2 v}{\partial x^2} - \left((r-\delta) - \frac{\sigma^2}{2} - \frac{\bar{S}'(\tau)}{\bar{S}(\tau)}\right)\frac{\partial v}{\partial x} + rv = 0 \quad \text{for } x > 0 \text{ and } \tau \in (0,T] \tag{2.23b}$$

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

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

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

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



(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 backward 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,

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

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

Finally, grouping the equations together, we have the system

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

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

3 Finite difference schemes

3.1 Overview

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 PDEs for call and put options that resulted from transforming (2.15) using the change of variable suggested by Nielsen et al.[15], and by Company et al.[3] which resulted in the systems (2.21) and (2.27), respectively. Now, we present numerical schemes 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

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

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

Suppose we discretize $\mathcal{F}$ into the grid $\mathcal{G}$ (See figure 3.2) 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 in $\mathcal{G}$ 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 grows. Therefore, we refer to Δx and Δt as the resolution of the grid.

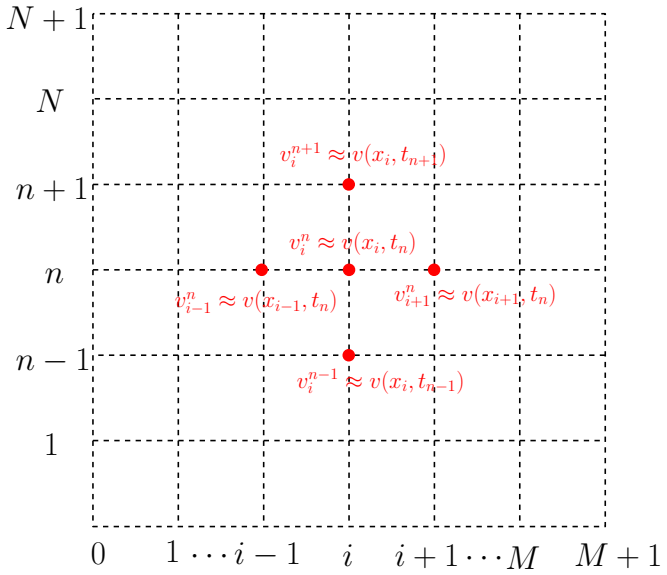


Figure 3.1: The grid $\mathcal{G}$ and the approximation $v_i^n \approx v(x_i, t_n)$ in each node.

From equation (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. Given the grid $\mathcal{G}$, our goal is to approximate the value function $v(x,t)$ and the optimal exercise price $\bar{S}(t)$ at each node (i,t)

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

Moreover, we want that the error of the approximation converges to zero as we decrease the discretization parameters. 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. Finally, within the grid we can approximate derivatives using is finite differences[14]. The idea of finite differences is to approximate derivatives as the difference of contiguous nodes in the grid $\mathcal{G}$. Let us say we are at node (x_i, t_n), then the forward differences approximate the derivative as

$$\frac{v(x_{i+1}, t_n) - v(x_i, t_n)}{\Delta x} = \frac{\partial v}{\partial x}(x_i, t_n) + O(\Delta x) \tag{3.8}$$

the backward difference approximate the derivative as

$$\frac{v(x_i,t_n)-v(x_{i-1},t_n)}{\Delta x}=\frac{\partial v}{\partial x}(x_i,t_n)+O(\Delta x) \tag{3.9}$$

the central finite difference approximate the first order derivative as

$$\frac{v(x_{i+1},t_n)-v(x_{i-1},t_n)}{2\Delta x}=\frac{\partial v}{\partial x}(x_i,t_n)+O(\Delta x^2) \tag{3.10}$$

and the second order central difference approximate second order derivatives as

$$\frac{v(x_{i+1},t_n)-2v(x_i,t_n)+v(x_{i-1},t_n)}{\Delta x^2}=\frac{\partial^2 v}{\partial x^2}(x_i,t_n)+O(\Delta x^2) \tag{3.11}$$

where $O(\Delta x)$ and $O(\Delta x^2)$ are high order terms (See Nassif[14] for proof).

3.2 Explicit scheme

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

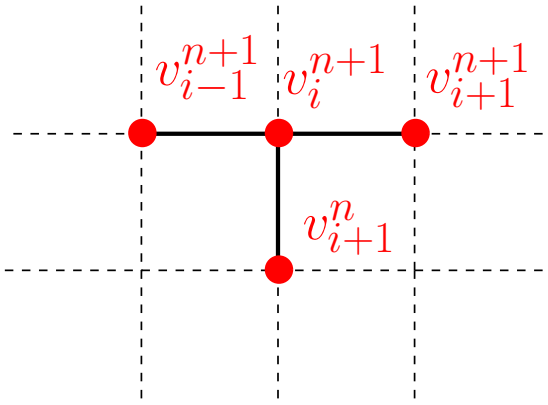


Figure 3.2: Stencil diagram of the explicit scheme.

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

$$\frac{v_i^{n+1} - v_i^n}{\Delta t} = \frac{\partial v}{\partial t} + O(\Delta t) \qquad \text{for } n = N, \ldots, 0 \tag{3.12}$$

$$\frac{\bar{S}^{n+1} - \bar{S}^n}{\Delta t} = \bar{S}'(t) + O(\Delta t) \qquad \text{for } n = N, \ldots, 0 \tag{3.13}$$

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 (3.10), (3.11), (3.12), and (3.12),the approximation of the PDE in (2.21) is given by

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

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

$$\lambda := \frac{\Delta t}{(\Delta x)^2}$$

$$A_i := \frac{\lambda}{2}\sigma^2 x_i^2 - \frac{\lambda}{2}\left((r - \delta) - \frac{1}{\Delta t} \right) x_i \Delta x \qquad \text{for } i = 1, \ldots, M$$

$$B_i := 1 - \lambda \sigma^2 x_i^2 - r\Delta t \qquad \text{for } i = 1, \ldots, M$$

$$C_i := \frac{\lambda}{2}\sigma^2 x_i^2 + \frac{\lambda}{2}\left((r - \delta) - \frac{1}{\Delta t} \right) x_i \Delta x \qquad \text{for } i = 1, \ldots, M$$

$$D_i^{n+1} := \frac{x_i}{2\Delta x} \frac{v_{i+1}^{n+1} - v_{i-1}^{n+1}}{\bar{S}^{n+1}} \qquad \text{for } i = 1, \ldots, M$$

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

$$v_i^n - D_i^{n+1}\bar{S}^n = A_i v_{i-1}^{n+1} + B_i v_i^{n+1} + C_i v_{i+1}^{n+1} \tag{3.14}$$

for $i = 1, \ldots, M$ and $t = N, \ldots, 0$. Moreover, the PDE problem in (2.21) have well-defined spatial boundary conditions. Remember that 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}.

$$\textbf{Call:} \qquad v_0^n = 0, \qquad v_{M+1}^n = \bar{S}^n - K \tag{3.15a}$$

$$\textbf{Put:} \qquad v_0^n = K - \bar{S}^n, \qquad v_{M+1}^n = 0 \tag{3.15b}$$

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

$$v_i^{N+1} = 0, \qquad \bar{S}^{N+1} = K \tag{3.16a}$$

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.

$$\textbf{Call:} \qquad \frac{v_{M+1}^n - v_M^n}{\Delta x} = \frac{\partial v}{\partial x}(1,t) + O(\Delta x)$$

$$\textbf{Put:} \qquad \frac{v_1^n - v_0^n}{\Delta x} = \frac{\partial v}{\partial x}(1,t) + O(\Delta x)$$

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

$$\textbf{Call:} \qquad v_M^n = v_{M+1}^n - \Delta x \bar{S}^n = (1 - \Delta x)\bar{S}^n - K \qquad \text{for } n = N, \ldots, 0 \tag{3.17a}$$

$$\textbf{Put:} \qquad v_1^n = v_0^n - \Delta x \bar{S}^n = K - (1 + \Delta x)\bar{S}^n \qquad \text{for } n = N, \ldots, 0 \tag{3.17b}$$

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 (3.14) at x_M and at x_1 for call and put options, respectively. Then, rearranging the resulting expression in terms of $\bar{S}^n$

$$\textbf{Call:} \qquad \bar{S}^n = \frac{K + A_M v_{M-1}^{n+1} + B_M v_M^{n+1} + C_M v_{M+1}^{n+1}}{(1-\Delta x) - D_M^{n+1}} \tag{3.18a}$$

$$\textbf{Put:} \qquad \bar{S}^n = \frac{K - (A_1 v_0^{n+1} + B_1 v_1^{n+1} + C_1 v_2^{n+1})}{D_1^{n+1} + (1+\Delta x)} \tag{3.18b}$$

for $n = N, \ldots, 0$. Thus, combining (3.14), (3.15), (3.16), (3.17), and (3.18), the explicit scheme of PDE problem (2.21) is given by

Call:

$$\begin{cases} v_i^n - D_i^{n+1}\bar{S}^n = A_i v_{i-1}^{n+1} + B_i v_i^{n+1} + C_i v_{i+1}^{n+1} & \text{for } i = 1, \ldots, M-1 \text{ and } n = N, \ldots, 0 \\ v_i^{N+1} = 0 & \text{for } i = 0, \ldots, M+1 \\ \bar{S}^{N+1} = K & \\ v_0^n = 0 & \text{for } n = N, \ldots, 0 \\ v_M^n = (1-\Delta x)\bar{S}^n - K & \text{for } n = N, \ldots, 0 \\ v_{M+1}^n = \bar{S}^n - K & \text{for } n = N, \ldots, 0 \end{cases} \tag{3.19a}$$

Put:

$$\begin{cases} v_i^n - D_i^{n+1}\bar{S}^n = A_i v_{i-1}^{n+1} + B_i v_i^{n+1} + C_i v_{i+1}^{n+1} & \text{for } i = 2, \ldots, M \text{ and } n = N, \ldots, 0 \\ v_i^{N+1} = 0 & \text{for } i = 0, \ldots, M+1 \\ \bar{S}^{N+1} = K & \\ v_0^n = K - \bar{S}^n & \text{for } n = N, \ldots, 0 \\ v_1^n = K - (1+\Delta x)\bar{S}^n & \text{for } n = N, \ldots, 0 \\ v_M^n = 0 & \text{for } n = N, \ldots 0 \end{cases} \tag{3.19b}$$

Finally, we formulate an algorithm for solving the system (3.19)

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.

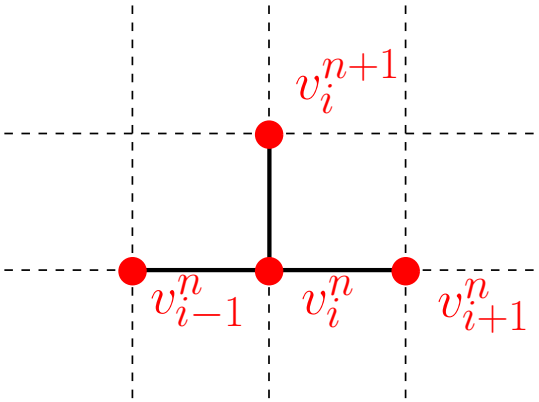


Figure 3.3: Stencil diagram of the implicit scheme.

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

$$\frac{v_{i+1}^n - v_{i-1}^n}{2\Delta x} = \frac{\partial v}{\partial x} + O(\Delta x^2) \tag{3.20}$$

$$\frac{v_{i+1}^n - 2v_i^n + v_{i-1}^n}{\Delta x^2} = \frac{\partial^2 v}{\partial x^2} + O(\Delta x^2) \tag{3.21}$$

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

$$\frac{v_i^{n+1} - v_i^n}{\Delta t} = \frac{\partial v}{\partial t} + O(\Delta t) \tag{3.22}$$

$$\frac{\bar{S}^{n+1} - \bar{S}^n}{\Delta t} = \bar{S}'(t) + O(\Delta t) \tag{3.23}$$

for $n = N, \ldots, 0$. Hence, combining (3.20), (3.21), (3.22) and (3.23),we obtain the implicit approximation of the PDE (2.21) as

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

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

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

$$\beta_i^n := 1 + \lambda \sigma^2 x_i^2 + r\Delta t \tag{3.25}$$

$$\gamma_i^n := -\frac{\lambda}{2}\sigma^2 x_i^2 + \frac{\lambda \Delta x}{2} x_i \left(r - \delta + \frac{\bar{S}^{n+1} - \bar{S}^n}{\Delta t \bar{S}^n} \right) \tag{3.26}$$

and rearrange the PDE as

$$\alpha_i^n v_{i-1}^n + \beta_i^n v_i^n + \gamma_i^n v_{i+1}^n = v_i^{n+1} \tag{3.27}$$

The boundary and terminal conditions are given by (3.15) and (3.16). Likewise, the approximation of v_M^n or v_1^n for put and call, respectively, is given by (3.17). 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 (3.15), (3.16), (3.17) and (3.27)

Call:

$$\begin{cases} \alpha_i^n v_{i-1}^n + \beta_i^n v_i^n + \gamma_i^n v_{i+1}^n = v_i^{n+1} & \text{for } i = 1, \ldots, M-1 \text{ and } n = N, \ldots, 0 \\ v_i^{N+1} = 0 & \text{for } i = 0, \ldots, M+1 \\ \bar{S}^{N+1} = K & \\ v_0^n = 0 & \text{for } n = N, \ldots 0 \\ v_M^n = (1 - \Delta x)\bar{S}^n - K & \text{for } n = N, \ldots 0 \\ v_{M+1}^n = \bar{S}^n - K & \text{for } n = N, \ldots 0 \end{cases} \tag{3.28a}$$

$$\textbf{Put:} \quad \begin{cases} \alpha_i^n v_{i-1}^n + \beta_i^n v_i^n + \gamma_i^n v_{i+1}^n = v_i^{n+1} & \text{for } i = 2, \dots, M \text{ and } n = N, \dots, 0 \\ v_i^{N+1} = 0 & \text{for } i = 0, \dots, M+1 \\ \bar{S}^{N+1} = K & \\ v_0^n = K - \bar{S}^n & \text{for } n = N, \dots 0 \\ v_1^n = K - (1 + \Delta x)\bar{S}^n & \text{for } n = N, \dots 0 \\ v_M^n = 0 & \text{for } n = N, \dots 0 \end{cases} \tag{3.28b}$$

Note that the system of equation above is cannot be solved explicitly as we did for (3.19). In normal circumstance, solving the system above would require to solve a system of linear equations consisting of $M-1$ unknowns (including $\bar{S}^n$). However, the terms α_i and γ_i are a nonlinear with respect of $\bar{S}^n$ which indicate us that instead we will have to solve a system nonlinear of equations. Let us define the vector $\mathbf{v}^n \in \mathbb{R}^{M-2}$ as

$$\textbf{Call:} \qquad \mathbf{v}^n := \begin{bmatrix} v_1^n, & v_2^n, & \cdots, & v_{M-1}^n \end{bmatrix}^\mathsf{T} \tag{3.29a}$$

$$\textbf{Put:} \qquad \mathbf{v}^n := \begin{bmatrix} v_2^n, & v_3^n, & \cdots, & v_M^n \end{bmatrix}^\mathsf{T} \tag{3.29b}$$

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

$$\textbf{Call:} \qquad \mathbf{\Lambda}^n = \begin{bmatrix} \beta_1^n & \gamma_1^n & & & & \\ \alpha_2^n & \beta_2^n & \gamma_2^n & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \alpha_{M-2}^n & \beta_{M-2}^n & \gamma_{M-2}^n \\ & & & & \alpha_{M-1}^n & \beta_{M-1}^n \\ & & & & & \alpha_M^n \end{bmatrix} \tag{3.30a}$$

$$\textbf{Put:} \qquad \Lambda^n := \begin{bmatrix} \gamma_1^n & & & & & \\ \beta_2^n & \gamma_2^n & & & & \\ \alpha_3^n & \beta_3^n & \gamma_3^n & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \alpha_{M-1}^n & \beta_{M-1}^n & \gamma_{M-1}^n \\ & & & & \alpha_M^n & \beta_M^n \end{bmatrix} \tag{3.30b}$$

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

$$\textbf{Call:} \qquad \mathbf{f}^n := \begin{bmatrix} v_1^{n+1} \\ \vdots \\ v_{M-1}^{n+1} - \gamma_{M-1}^n[(1-\Delta x)\bar{S}^n - K] \\ v_M^{n+1} - \gamma_M^n(\bar{S}^n - K) - \beta_M^n[(1-\Delta x)\bar{S}^n - K] \end{bmatrix} \tag{3.31a}$$

$$\textbf{Put:} \qquad \mathbf{f}^n := \begin{bmatrix} v_1^{n+1} - \alpha_1^n(K - \bar{S}^n) - \beta_1^n[K - (1+\Delta x)\bar{S}^n] \\ v_2^{n+1} - \beta_2^n[K - (1+\Delta x)\bar{S}^n] \\ v_3^{n+1} \\ \vdots \\ v_{M-1}^{n+1} \end{bmatrix} \tag{3.31b}$$

Thus, we have to solve the nonlinear system of equations

$$\mathbf{F}\Big(\Big[\mathbf{v}^n | \bar{S}^n\Big]^{\mathsf{T}}\Big) = \Lambda^n \mathbf{v}^n - \mathbf{f}^n = 0 \tag{3.32}$$

We can use newton's method to solve the nonlinear system iteratively

$$\mathbf{y}_{k+1} = \mathbf{y}_k - \mathbf{J}^{-1}(\mathbf{y}_k)\mathbf{F}(\mathbf{y}_k) \tag{3.33}$$

where $\mathbf{J}(\mathbf{y}_k) \in \mathbb{R}^{(M-1)\times(M-1)}$ is the Jacobian of $\mathbf{F}(\mathbf{y}_k)$, $\mathbf{y}_k \in \mathbb{R}^{M-1}$ is some approximation of true solution

$$\mathbf{y} = \Big[\mathbf{v}^n | \bar{S}^n\Big]^{\mathsf{T}} \tag{3.34}$$

an initial value

$$\mathbf{y}_0 = \left[\mathbf{v}^{n+1} | \bar{S}^{n+1}\right]^{\mathsf{T}} \tag{3.35}$$

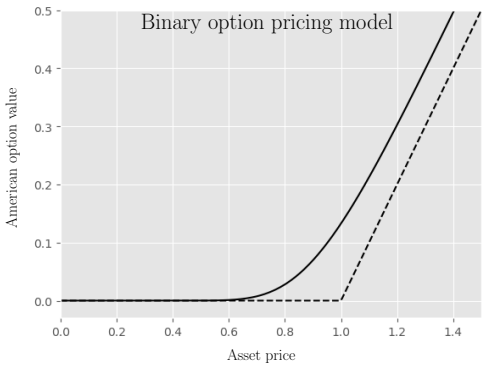
3.4 Numerical results

3.4.1 Numerical experiments

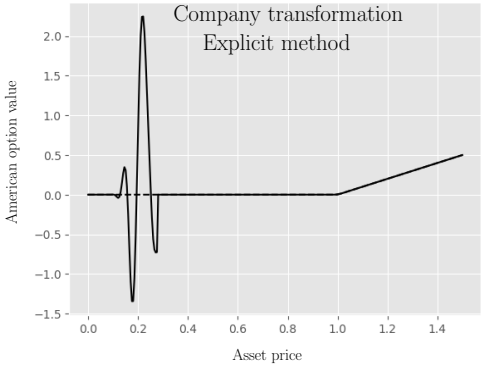
Generally, Datasets for American options are expensive to acquire. Therefore, to validate our implementation, we mainly relied on the data available in Company, et al.[3], Nielsen, et al.[15], Seydel[17], and Wilmott, et al.[7]. Moreover, used the approximations produced by binary option pricing model[5] (BOPM) as benchmark for assessing the consistency of our method. We chose the binomial model because it is widely used in the industry, and is simple to implement. Moreover, the BOPM uses a completely different approach to price options than the one considered in this dissertation. The first experiment that come into mind is to price call and put options without dividend. Therefore, let us define the parameter for the numerical experiment as

$$K = 1, \quad T = 1, \quad r = 0.2, \quad \sigma = 0.02, \quad \delta = 0 \tag{3.36}$$

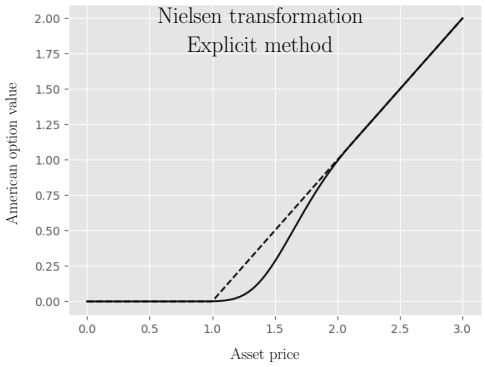
from Nielsen et al.[15], we know that this set of parameters produce the optimal exercise boundary $\bar{S}(t) \approx 0.86$. Also, we approximated $\bar{S}(t)$ using the binomial option pricing model. Finally, we priced call and put options using the explicit (3.19) and implicit (3.28) front fixing schemes based Nielsen transformation, and the explicit front fixing scheme based on the Company transformation (A.2) (See appendix). As you can observe in figure (3.4b), (3.4c), and (3.4d), both the Nielsen and Company schemes do not produce the value function as per figure (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 described by figure (2.1a). In fact, Merton[13] showed that, pricing American options without dividends is equivalent to solve (2.5) with boundary conditions $V(0,t) = 0$ and $\lim_{S->\infty} V(S,t) = S - K$. Therefore, the problem reduces from a free boundary problem to a boundary problem. Thus, making the front fixing Nielsen and Company PDEs inappropriate for American call options without



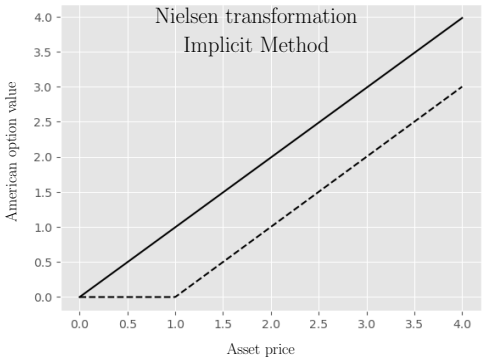
(a) Nodes $= 2^{500}$.



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



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

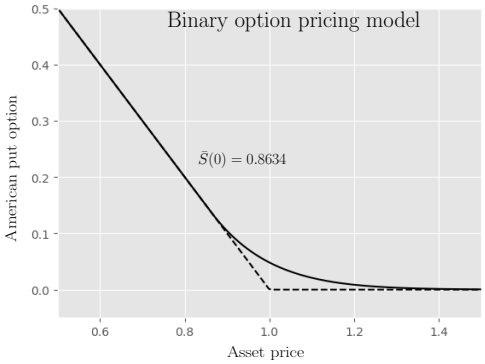


(d) $\Delta x = 1\text{e}{-3}, \Delta t = 0.5 \times 1\text{e}{-6}$

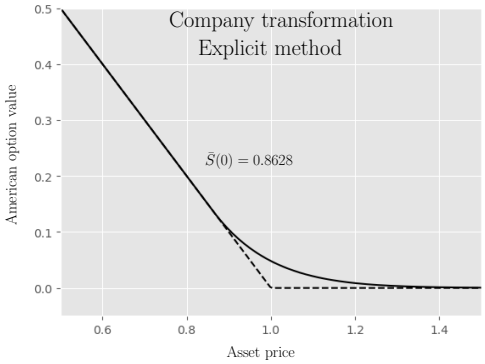
Figure 3.4: American call option value $V(S, 0)$ given parameters (3.36).

dividends.

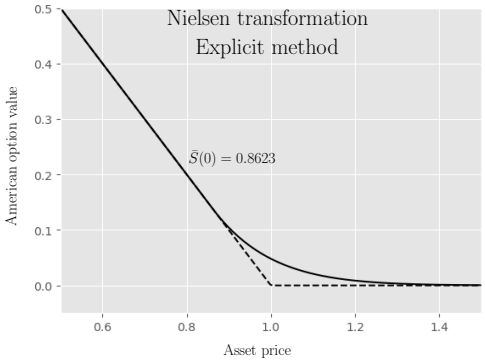
Now, let us consider the case of pricing put options without dividends given parameters (3.36). Figures (3.5) show the $V(S, 0)$ curve obtained by all the schemes. In each plot, we have listed the optimal exercise boundary $\bar{S}(0)$. Also, we have listed the corresponding value curve approximated 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), we listed the approximation at several point along the value curve $V(S, t)$. Contrary to the case for call options, the value curves in (3.5) captured 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 $V(S, 0) > \max(K - S, 0)$ and $V(S, 0) \to 0$ as $S \to 0$. In addition, the value curve is exactly the payoff in the exercise region $S < \bar{S}(0)$. Finally, the explicit schemes yielded a significantly small RSME with a very low execution time.



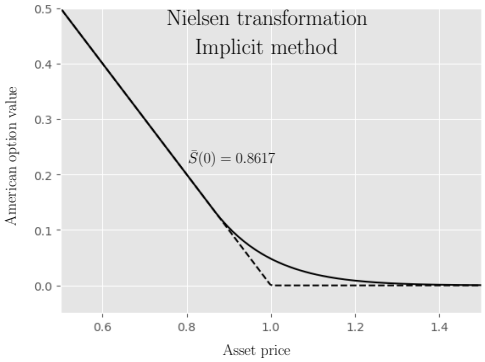
(a) Nodes $= 2^{500}$.



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

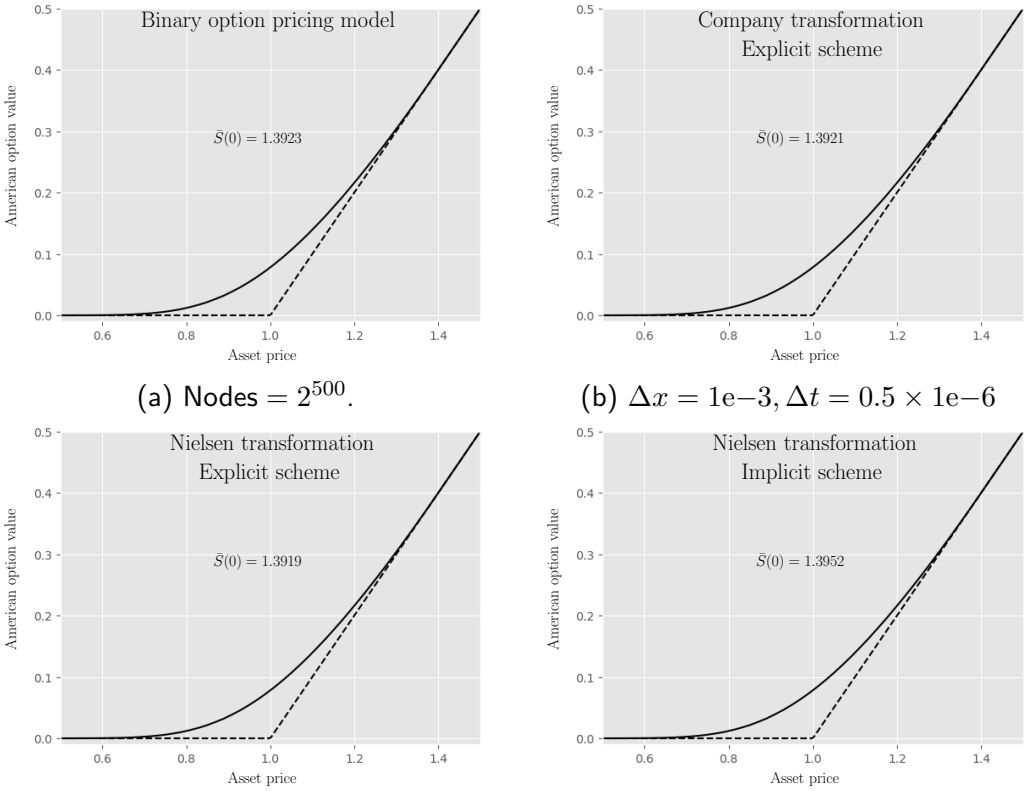
Figure 3.5: American put option value $V(S, 0)$ given parameters (3.36).

Asset Price	**BOPM**	**Explicit Nielsen**	**Implicit Nielsen**	**Explicit Company**
0.8	0.200000	0.200000	0.200000	0.200000
1.0	0.048167	0.048163	0.048944	0.048174
1.2	0.008666	0.008657	0.009290	0.008667
1.4	0.000167	0.001284	0.001519	0.001287
1.6	0.001285	0.000167	0.000229	0.000168
1.8	0.000020	0.000020	0.000033	0.000020
2.0	0.000002	0.000002	0.000005	0.000002
	RSME	0.000005	0.000388	0.000003
	Time	614ms	9891ms	68ms

Table 3.1: $V(S, 0)$ put approximation given parameters (3.36).

The third numerical experiment was to price a call option with dividends

$$K = 1, \quad T = 1, \quad r = 0.2, \quad \sigma = 0.2, \quad \delta = 0.09 \tag{3.37}$$



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

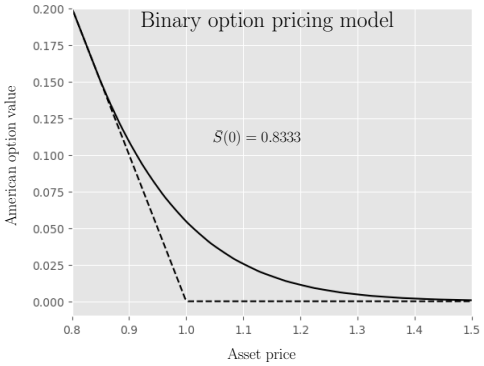
Figure 3.6: American call option value $V(S, 0)$ given parameters (3.37).

Asset Price	BOPM	Explicit Nielsen	Implicit Nielsen	Explicit Company
0.4	0.0000000	0.0000002	0.000083	0.0000000
0.6	0.0002771	0.0002712	0.002238	0.0002763
0.8	0.0119621	0.0119311	0.011796	0.0119633
1.0	0.0780630	0.0780630	0.078109	0.0780628
1.2	0.2159015	0.2159348	0.215718	0.2159007
1.4	0.4000000	0.3999589	0.400443	0.4000000
	RSME	0.000006	0.00203	0.000000
	Time	512ms	10132ms	83ms

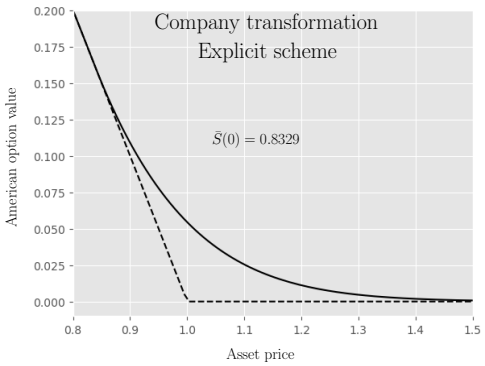
Table 3.2: $V(S,0)$ call approximation given parameters (3.37).

Finally, we also price a put option with dividends using parameters

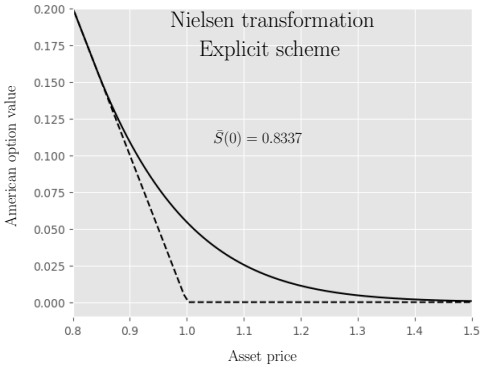
$$K = 1, \quad T = 1, \quad r = 0.2, \quad \sigma = 0.2, \quad \delta = 0.03 \tag{3.38}$$



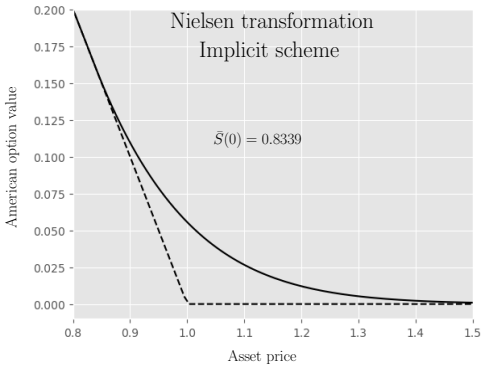
(a) Nodes $= 2^{500}$.



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

Asset Price	**BOPM**	**Explicit Nielsen**	**Implicit Nielsen**	**Explicit Company**
0.6	0.400000	0.400000	0.400000	0.400000
0.8	0.200000	0.200000	0.200000	0.200000
1.0	0.054464	0.054469	0.055593	0.054491
1.2	0.011255	0.011161	0.012096	0.011167
1.4	0.001855	0.001847	0.002225	0.001850
1.6	0.000262	0.000265	0.000373	0.000266
	RMSE	0.000031	0.000486	0.000031
	Time	416ms	29581ms	194ms

Table 3.3: $V(S, 0)$ put approximation given parameters (3.38).

Summarizing, none of the front fixing schemes can be used to price American call options without dividends. The explicit schemes are much faster than the implicit scheme. Nielsen

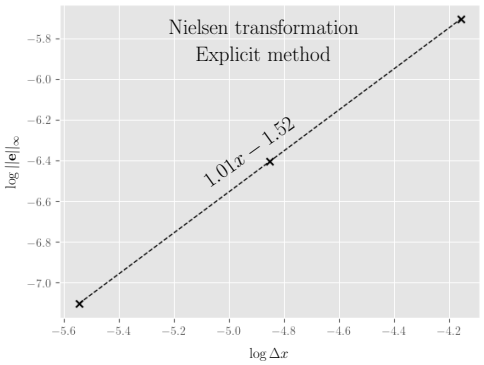
explicit scheme is less accurate than Company explicit scheme. We will see later that Company explicit scheme is second order in space. Therefore, it converges faster than Nielsen explicit scheme. Moreover, the Nielsen is slower because at each time step it has to compute the term D_i^{n+1} for $i = 0, \ldots, M+1$ (See line 9-10 in algorithms (3.1) and (3.2)) while the Company does not (See algorithms (A.1) and (A.2)).

3.4.2 Convergence analysis

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 (3.38). We expect that numerical experiments 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

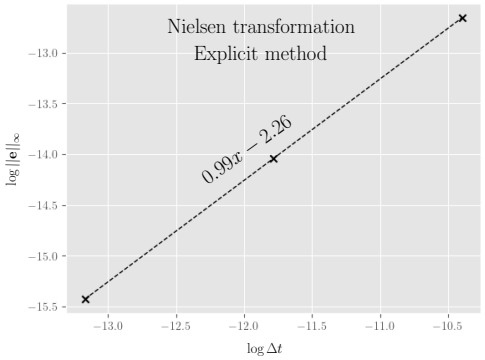
$$\mathbf{e}_k = ||\mathbf{v}_{k+1} - \mathbf{v}_k||_\infty \quad \text{for } k = 0, \ldots, 3 \tag{3.39}$$

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 (3.8), 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, the Nielsen schemes use forward (or backward for calls) difference to approximate the contact point condition (3.17) which is first order in space, hence, degrading the overall convergence of the method. In the other hand, figure (3.9) 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.5). Moreover, all the schemes are first order convergence in space as suggested by the forward/backward difference approximation used for the temporal axis.



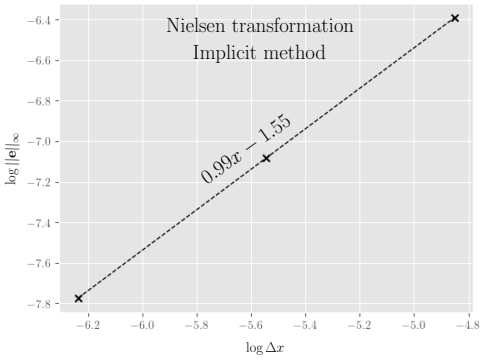
(a) $\Delta x = 2^{-7}, \ldots, 2^{-10}$

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



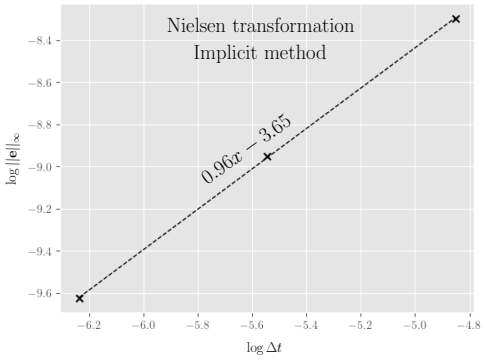
(b) $\Delta t = 2^{-15}, 2^{-17}, \ldots, 2^{-21}$

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



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

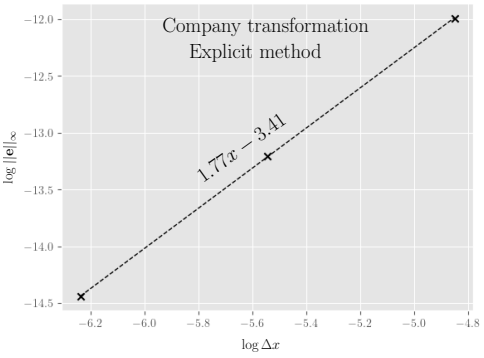
$\Delta t = 2^{-2}$



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

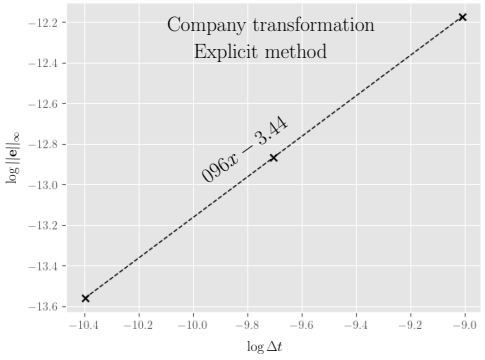
$\Delta x = 2^{-5}$

Figure 3.8: Convergence analysis for the explicit and implicit method for the Nielsen transformation.



(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 3.9: 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 $\mathbf{b} \in \mathbb{R}^d$, we have to 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. But first, 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

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

Secondly, 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 $H(S,t)$ in the Black-Scholes PDE, we obtain the Black-Scholes PDE inequality

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

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

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

where $\tau := T - t$. 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 the LCP problem defined at the beginning of this section. 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 inequality (4.3) transforms to the heat diffusion inequality

$$\frac{\partial y}{\partial \tau} - \frac{\partial^2 y}{\partial x^2} \leq 0 \qquad \text{for } x \in \mathbb{R} \text{ and } \tau \in (0, \sigma^2 T/2] \tag{4.7}$$

where $y \in C^{2,1}\Big(\mathbb{R} \times [0, \sigma^2 T/2]\Big)$ have boundary conditions

$$\lim_{x\to-\infty} y(x,\tau) = \lim_{x\to-\infty} h(0,\tau) = 0 \tag{4.8}$$

$$\lim_{x\to\infty} y(x,\tau) = \lim_{x\to\infty} h(0,\tau) = 0 \tag{4.9}$$

and initial condition

$$y(x,0) = h(x,0) \tag{4.10}$$

Although we might solve (4.4) without transforming the problem, Dewynne et al.[6] and Seydel[17] 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

$$\begin{cases} \left[\dfrac{\partial y}{\partial \tau} - \dfrac{\partial^2 y}{\partial x^2}\right] \cdot [y(x,\tau) - h(x,\tau)] = 0 & \text{for all } (x,\tau) \\ y(x,\tau) - h(x,\tau) \geq 0 & \text{for all } (x,\tau) \\ \dfrac{\partial y}{\partial \tau} - \dfrac{\partial^2 y}{\partial x^2} \geq 0 & \text{for all } (x,\tau) \\ y(x,0) = h(x,0) \end{cases} \tag{4.11}$$

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 node within the grid

$$\mathcal{G} := \{(x_i, \tau_n) : (i,n) \in \{0,\dots,M+1\} \times \{0,\dots,N+1\}\} \tag{4.12}$$

$$x_i := x_{\text{min}} + i\Delta x \quad \text{for } i = 0,\dots,M+1 \tag{4.13}$$

$$\tau_n := t_{\text{min}} + i\Delta\tau \quad \text{for } i = 0,\dots,N+1 \tag{4.14}$$

$$\Delta x := \frac{x_{\text{max}} - x_{\text{min}}}{M+1} \tag{4.15}$$

$$\Delta t := \frac{t_{\text{max}} - t_{\text{min}}}{N+1} \tag{4.16}$$

where x_{min} is set sufficiently small value, x_{max} to a sufficiently large value, τ_{min} to 0 and τ_{max} to $\dfrac{\sigma^2}{2}T$. 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

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

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

for $i = 1,\dots,M$ and $n = 0\dots,N$. Note that contrary to (2.18), the heat equation is written forward in time. Therefore, the explicit scheme correspond to equation (4.17), and the implicit scheme to equation (4.18). In addition, we introduce the Crank-Nicholson scheme[8][17] for

equation (4.7) as

$$\frac{y_i^{n+1} - y_i^n}{\Delta\tau} = \frac{1}{2}\left(\frac{y_{i-1}^n - 2y_i^n + y_{i+1}^n}{(\Delta x)^2} + \frac{y_{i-1}^{n+1} - 2y_i^{n+1} + y_{i+1}^{n+1}}{(\Delta x)^2}\right) \tag{4.19}$$

for $i = 1, \ldots, M$ and $n = 0 \ldots, N$. Epperson[8] and Seydel[17] showed that the Crank-Nicholson scheme is $O(\Delta x^2 + \Delta t^2)$.

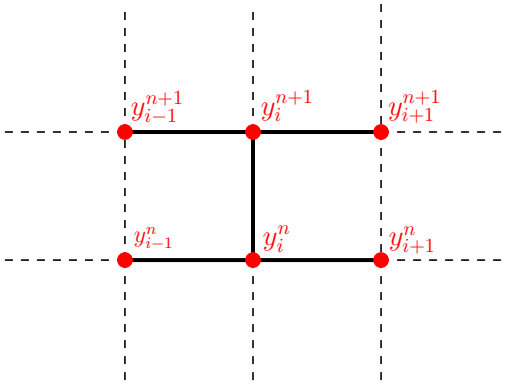


Figure 4.1: Stencil diagram for the Crank-Nicholson scheme.

We can generalize (4.17), (4.18), and (4.19) using a parametrized scheme called the theta method

$$\frac{y_i^{n+1} - y_i^n}{\Delta\tau} = (1-\theta)\frac{y_{i-1}^n - 2y_i^n + y_{i+1}^n}{(\Delta x)^2} + \theta\frac{y_{i-1}^{n+1} - 2y_i^{n+1} + y_{i+1}^{n+1}}{(\Delta x)^2} \tag{4.20}$$

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 (4.20) as

$$y_i^{n+1} - \lambda\theta(y_{i-1}^{n+1} - 2y_i^{n+1} + y_{i+1}^{n+1}) = y_i^n + (1-\theta)\lambda(y_{i-1}^n - 2y_i^n + y_{i+1}^n) \tag{4.21}$$

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

$$\hat{\mathbf{b}}^n := \left[\hat{b}_1^n, \ldots, \hat{b}_M^n\right]^\mathsf{T} \tag{4.22}$$

$$\mathbf{y}^{n+1} := \left[y_1^{n+1}, \ldots, y_M^{n+1}\right]^\mathsf{T} \tag{4.23}$$

$$\mathbf{h}^{n+1} := \left[h(x_1, \tau_{n+1}), \ldots, h(x_M, \tau_{n+1})\right]^\mathsf{T} \tag{4.24}$$

where

$$\hat{b}_i^n := y_i^n + (1-\theta)\lambda(y_{i-1}^n - 2y_i^n + y_{i+1}^n) \tag{4.25}$$

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

$$\mathbf{A} := \begin{bmatrix} 1+2\lambda\theta & -\lambda\theta & & 0 \\ -\lambda\theta & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ 0 & & \ddots & \ddots \end{bmatrix} \tag{4.26}$$

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 (4.11)using the finite difference scheme proposed as

$$\begin{cases} (\mathbf{A}\mathbf{y}^{n+1} - \hat{\mathbf{b}}^n)^\mathsf{T}(\mathbf{y}^{n+1} - \mathbf{h}^{n+1}) = 0 \\ \mathbf{A}\mathbf{y}^{n+1} - \hat{\mathbf{b}}^n \geq 0 \\ \mathbf{y}^{n+1} - \mathbf{h}^{n+1} \geq 0 \\ \mathbf{y}^0 = \mathbf{h}^0 \end{cases} \tag{4.27}$$

for $n = 0, \ldots, N$. Moreover, (4.27) 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

$$\mathbf{M}\mathbf{x} = \mathbf{N}\mathbf{x} + \mathbf{b} \tag{4.28}$$

or as

$$\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{M}^{-1}\mathbf{b} \tag{4.29}$$

where $\mathbf{B} := \mathbf{M}^{-1}\mathbf{N} = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A}$. Clearly, equation (4.29) constitute fixed-point problem[8] which leads to the iterative method

$$\mathbf{x}^k = \mathbf{B}\mathbf{x}^{k-1} + \mathbf{M}^{-1}\mathbf{b} \tag{4.30}$$

Epperson[8] showed that iterative method (4.30) 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

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

According to Epperson[8], 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 $\mathbf{N} := \mathbf{U}$.

$$(\mathbf{D} - \mathbf{L})\mathbf{x}^k = \mathbf{U}\mathbf{x}^{k-1} + \mathbf{M}^{-1}\mathbf{b} \tag{4.32}$$

A generalization of the Gauss-Seidel (4.32) 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

$$\left(\frac{1}{\omega}\mathbf{D} - \mathbf{L}\right)\mathbf{x}^k = \left(\left(\frac{1}{\omega} - 1\right)\mathbf{D} + \mathbf{U}\right)\mathbf{x}^{k-1} + \mathbf{b} \tag{4.33}$$

Note that for $\omega = 1$, the Gauss-Seidel (4.33) method is obtained. Suppose, we want to solve $\mathbf{A}\mathbf{v} = \mathbf{b}$ for $\mathbf{A}$ given as in (4.26), $\mathbf{v} \in \mathbb{R}^M$, and $\mathbf{b} \in \mathbb{R}^M$. Then, a componentwise version of the (4.33) is given by

$$\frac{1}{\omega}(1 + 2\alpha)v_i^k = \left(\frac{1}{\omega} - 1\right)(1 + 2\alpha)v_i^{k-1} + \alpha(v_{i-1}^k + v_{i+1}^{k-1}) + b_i \tag{4.34}$$

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.[17] and Dewynne et al.[7] showed that the PSOR method is equivalent to LCP algorithm (4.2). Moreover, they also showed that the fastest 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 (4.13).

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 (4.22).

    Define $\hat{\mathbf{b}}^n$ as in (4.24).

    $v_i^0 := \max(y_i^n, h_i^{n+1})$ 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**

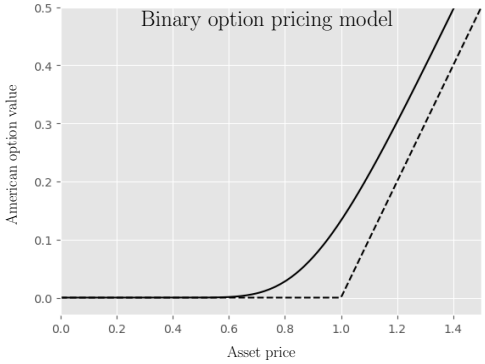
            $\mathbf{v}^{\text{new}} := \mathbf{v}^k$

            Break out of the loop

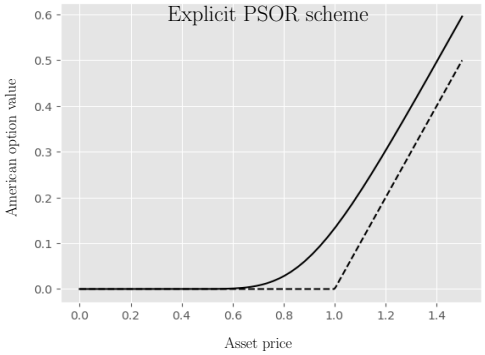
    $\mathbf{y}^{n+1} := \mathbf{v}^k$

4.3 Numerical results

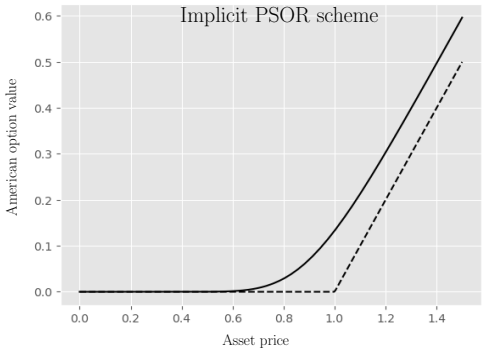
4.3.1 Numerical experiments



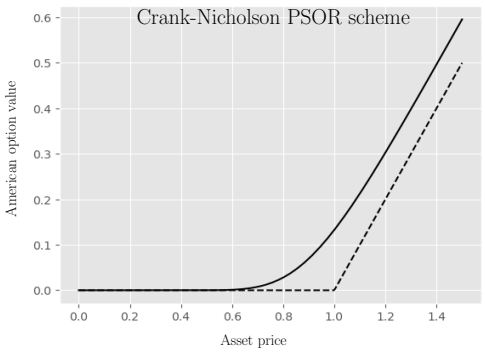
(a) Nodes $= 2^{500}$.



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



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

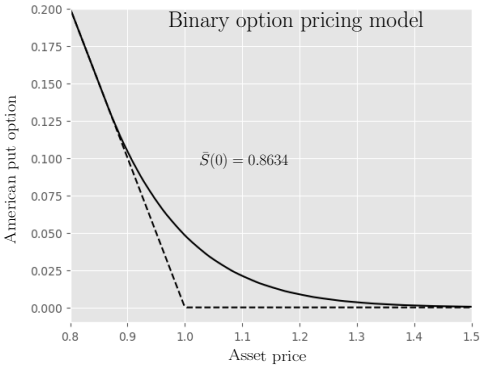


(d) $\Delta x = \Delta t = 1\text{e}{-3}$.

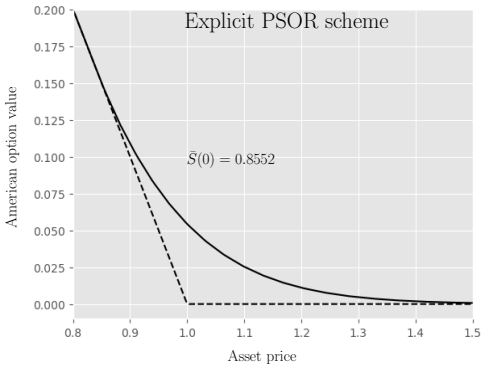
Figure 4.2: American call option value $V(S, 0)$ given parameters (3.36).

Asset Price	**BOPM**	**Explicit Nielsen**	**Implicit Nielsen**	**Explicit Company**
0.8	0.027907	0.028005	0.027907	0.028011
1.0	0.132688	0.132621	0.132688	0.132626
1.2	0.302596	0.302686	0.302596	0.302694
1.4	0.496317	0.496474	0.496317	0.496484
1.6	0.695316	0.695494	0.695316	0.695507
1.8	0.895181	0.895384	0.895181	0.895398
	RMSE	0.000156	0.000826	0.000166
	Time	5062ms	2168ms	1480ms

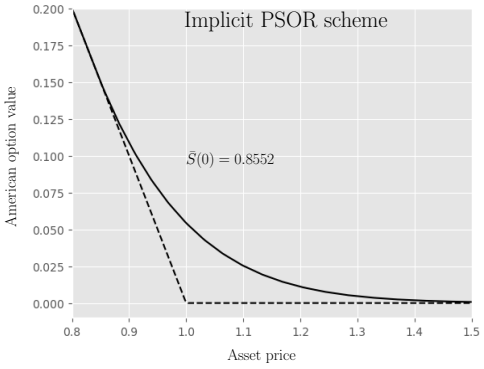
Table 4.1: $V(S, 0)$ call approximation given parameters (3.36).



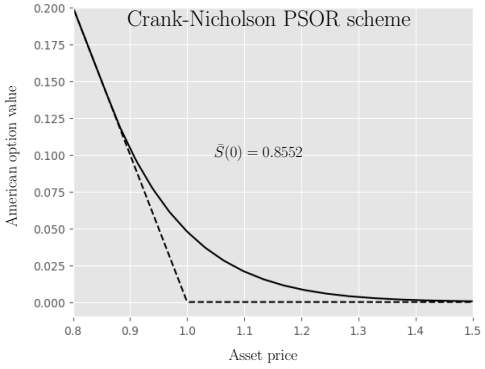
(a) Nodes $= 2^{500}$.



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



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

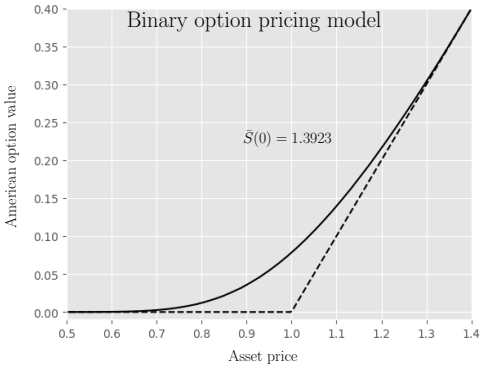


(d) $\Delta x = \Delta t = 1\text{e}{-3}$.

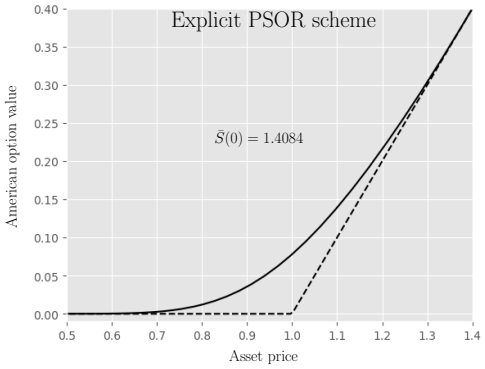
Figure 4.3: American put option value $V(S, 0)$ given parameters (3.36).

Asset Price	**BOPM**	**Explicit**	**Implicit**	**Crank-Nicholson**
0.8	0.200000	0.200000	0.200000	0.200000
1.0	0.048167	0.047919	0.047912	0.047916
1.2	0.008666	0.008646	0.008644	0.008645
1.4	0.001285	0.001296	0.001297	0.001296
1.6	0.000167	0.000170	0.000170	0.000170
	RMSE	0.000111	0.000115	0.000113
	Time	4609ms	19123ms	16741ms

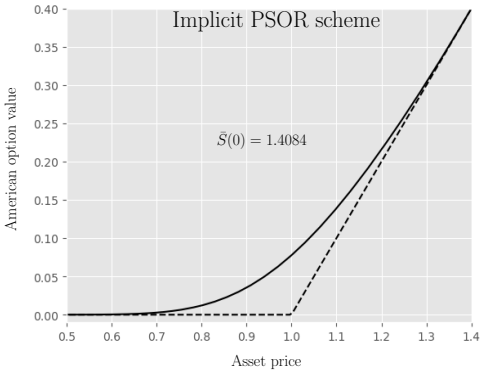
Table 4.2: $V(S, 0)$ put approximation given parameters (3.36).



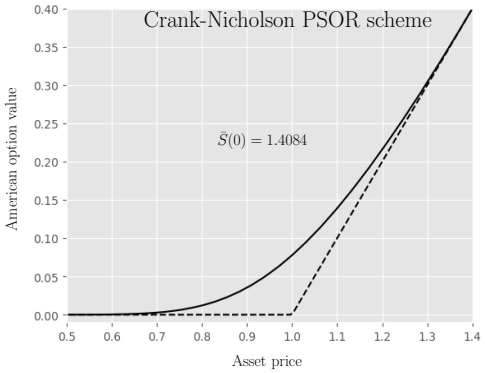
(a) Nodes $= 2^{500}$.



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



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

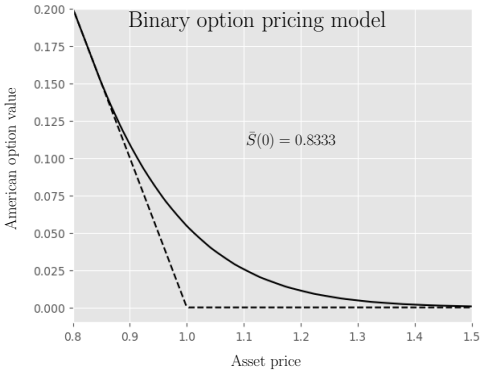


(d) $\Delta x = \Delta t = 1\text{e}{-3}$.

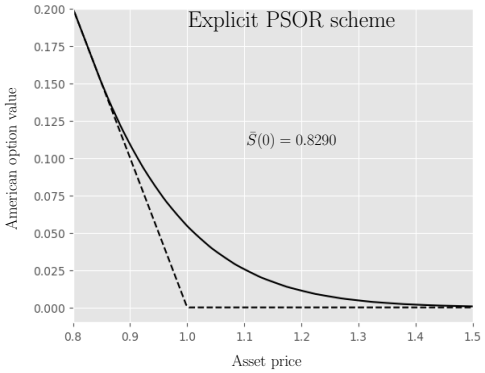
Figure 4.4: American call option value $V(S, 0)$ given parameters (3.37).

Asset Price	**BOPM**	**Explicit**	**Implicit**	**Crank-Nicholson**
0.8	0.011962	0.011949	0.0120106	0.011962
1.0	0.078062	0.077858	0.0775714	0.078062
1.2	0.215901	0.021586	0.2157617	0.215901
1.4	0.400000	0.400090	0.4000923	0.400000
1.6	0.600000	0.600000	0.6000000	0.600000
	RMSE	0.000072	0.000166	0.000074
	Time	4640ms	1821ms	1310ms

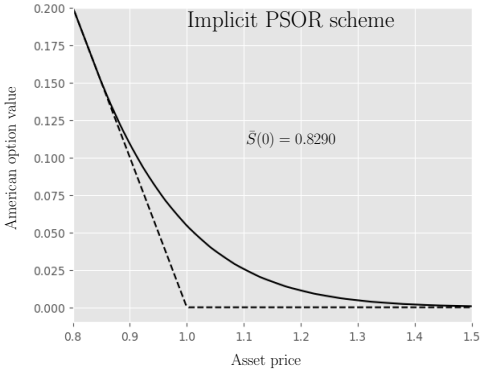
Table 4.3: $V(S, 0)$ put approximation given parameters (3.37).



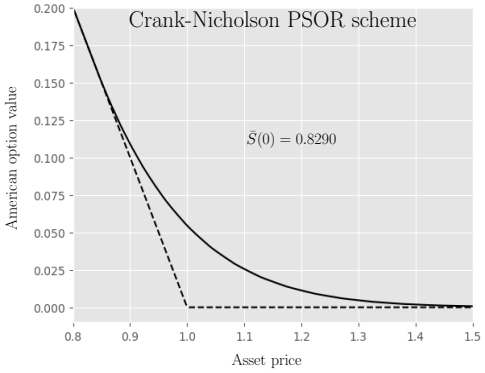
(a) Nodes $= 2^{500}$.



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



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



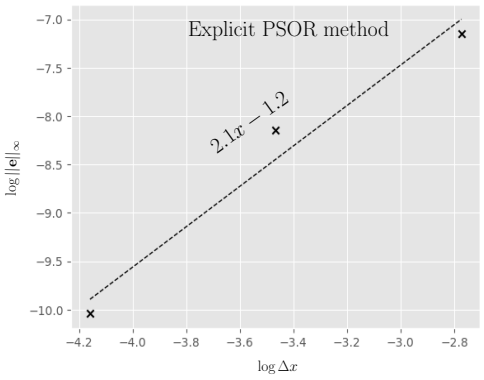
(d) $\Delta x = \Delta t = 1\text{e}{-3}$.

Figure 4.5: American put option value $V(S, 0)$ given parameters (3.38).

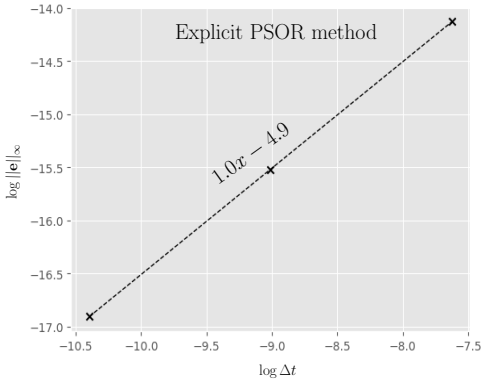
Asset Price	**BOPM**	**Explicit**	**Implicit**	**Crank-Nicholson**
0.8	0.200000	0.200000	0.200000	0.200000
1.0	0.054464	0.054246	0.053968	0.054239
1.2	0.011255	0.011134	0.011063	0.011129
1.4	0.001855	0.001860	0.001908	0.001858
1.6	0.000262	0.000268	0.000300	0.000268
	RMSE	0.000083	0.000179	0.000086
	Time	4585ms	1310ms	885ms

Table 4.4: $V(S, 0)$ put approximation given parameters (3.38).

4.3.2 Convergence Analysis

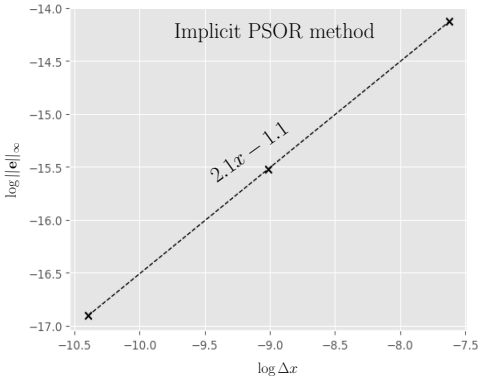


(a) $\Delta x = 2^{-7}, \ldots, 2^{-10}$
$\Delta t = 2^{-21}$

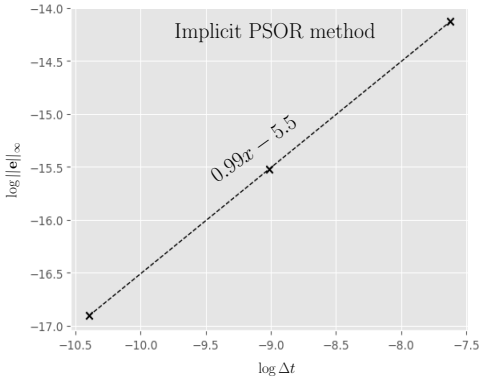


(b) $\Delta t = 2^{-15}, 2^{-17}, \ldots, 2^{-21}$
$\Delta x = 2^{-7}$

Figure 4.6: Convergence analysis for the explicit method for the Company transformation.

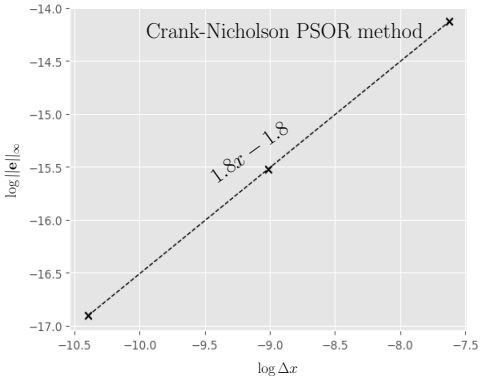


(a) $\Delta x = 2^{-7}, \ldots, 2^{-10}$
$\Delta t = 2^{-21}$

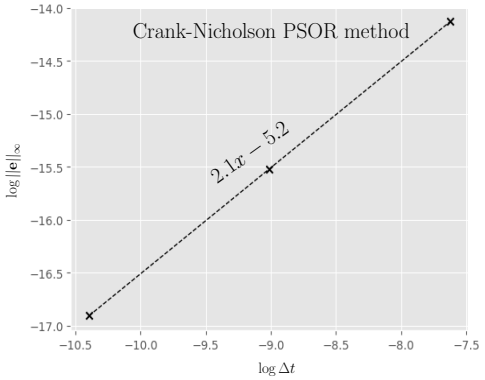


(b) $\Delta t = 2^{-15}, 2^{-17}, \ldots, 2^{-21}$
$\Delta x = 2^{-7}$

Figure 4.7: Convergence analysis for the explicit method for the Company transformation.



(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 4.8: Convergence analysis for the explicit method for the Company transformation.

5 Conclusions

An option provides the right to buy or sell an asset at a predetermined strike price in the future. Generally, investment firms are responsible for writing these contracts and selling them to investors. Subsequently, investors hold these contracts to hedge against potential changes in price. When a holder already owns an asset and wants to hedge against a potential price drop, they enter into a put option. On the contrary, if a holder aims to hedge against an increase in the price of an asset they intend to acquire, they enter into a call option contract. A wide variety of options are available in the market. Among the most notorious contracts, we have European and American options. European options are contracts that can exercise at the expiration date only. Likewise, American options are contracts that can be exercised before or at the maturity date.

Investment firms charge premium to investors for entering an option contract. Then, the writer uses the premium to hedge the possible claims that the holder will have in the futures. Charging the correct premium is important because it minimizes the arbitrage opportunities for either the holder and writer. Clearly, pricing schemes depend on the type of the contract. In general, The Black-Scholes formula is used to price European options. Sadly, no formula is available for American options. Therefore, firms rely on numerical methods to come up with some approximation of the price. Numerous numerical methods for pricing American options derive from the Black-Scholes PDE.

In this report we have implemented, and analyzed numerical schemes derived from the free boundary and the variational inequalities' formulation of the pricing problem. Specifically, we have discussed: An explicit and implicit front fixing schemes for solving the free boundary problem based on the Nielsen, an explicit front fixing scheme based on the Company transformation, and the PSOR method for solving the linear complementary problem derived from applying the theta method to the system of variational inequalities.***

The explicit and implicit front fixing schemes based on the Nielsen transformation, and the explicit front fixing scheme based on the Company transformation are derived from applying central finite differences. However, they differ in how the contact point condition is approximated. Specifically, The Nielsen front fixing schemes use forward difference (or backward

difference for call options) to approximate the contact point condition while the Company explicit scheme use central finite difference. This explains why Nielsen front fixing schemes yielded first order convergence in the spatial axis while Company explicit scheme is second order. Moreover, as it is normally the case for explicit and implicit central finite differences, both Nielsen and Company schemes exhibits first order convergence temporal axis. While the explicit front fixing schemes for Nielsen and Company transformation are conditionally stable, they both proved to be substantially faster and much more accurate than the implicit scheme. We discourage the use of the Nielsen implicit scheme. As we already told, the implicit scheme is less accurate by far. The reason behind this is that the implicit scheme requires to solve a non-linear system of equation at each time step in the grid. Therefore, the approximation errors produced by the non-linear solver get accumulated over time. Affecting the overall accuracy of the method. We could decrease the approximation error of the non-linear solver by decreasing its tolerance, and increasing its maximum number of iteration. However, we found that as we do that, the overall performance of the method reduces substantially. To make things even worse, the size of the non-linear equations is inversely proportional to the spatial discretization parameter. Therefore, the computational resources required by the implicit method grows substantially as we decrease the spatial discretization parameters. For instance, for a grid of M nodes in the spatial direction, the non-linear solver needs to invert a Jacobian matrix of $M \times M$ entries. In other words, decreasing the spatial discretization parameters by a decimal point, requires 100 times more memory. Consolidating, we can increase the accuracy of implicit method by decreasing the tolerance of the non-linear solver and by decreasing the discretization parameters of the grid but by sacrificing the performance of the method and increasing the memory consumption.

Similar to the Company front fixing schemes, the PSOR schemes showed to have second order convergence with respect the spatial discretization parameter. Moreover, it showed to have first order convergence for the explicit ($\theta = 0$) and implicit ($\theta = 1$) schemes, and second order convergence for the Crank-Nicholson ($\theta = 0.5$) with respect to the temporal discretization parameter. Analogous to the explicit front fixing schemes, the PSOR explicit scheme is conditionally stable. When comparing the performance of the explicit PSOR to the explicit

front fixing schemes, the explicit front fixing schemes showed to be substantially faster. In that regard, the issue with explicit PSOR is that as you decrease Δx, the complementary equations grow, hence, taking more time to solve the linear complementary problem. Similar argument can be done when comparing the implicit and Crank-Nicholson PSOR schemes. In spite of that, the Crank-Nicholson PSOR scheme is second order convergence in time, therefore, by choosing Δx to be smaller than Δt, we could have greater performance maintaining the accuracy.

To summarize, the numerical experiments conducted showed that the explicit front-fixing schemes offers superior performance and accuracy than the implicit front fixing scheme and the PSOR method. Between the Company explicit front fixing scheme and the Nielsen explicit front fixing scheme, we recommend using the Company transformation because it is second order convergence in space. Similarly, the Crank-Nicholson PSOR exhibited better results than explicit and implicit PSOR, and the Nielsen implicit front fixing scheme. Finally, we discourage using the Nielsen implicit front-fixing schemes because of poor performance and inaccuracy.

6 Further research

Further research opportunities arise from this report. Firstly, we saw that generally, people transform the Black-Scholes PDE to the heat diffusion equation. Although this was done for the LCP problem, the front fixing schemes derived within the option's domain which might led to worse approximations or slower performance. Moreover, we might derive Nielsen front fixing scheme that approximates the contact point condition using central finite differences. Likewise, for the Nielsen implicit front fixing scheme, we could explore using nonlinear methods for large scale-scale nonlinear systems such as the one proposed by[11]. Also, we might derive Crank-Nicholson schemes for the Nielsen and Company front fixing schemes. Finally, we might consider using real market data to evaluate how good are the numerical schemes presented under real market conditions.

A Explicit scheme for Company transformation

In this section, we will briefly derive the explicit scheme for the Company approximation.[3]. Similar as in section 3.2, the explicit front fixing Company scheme is obtained by approximating the second order spatial derivative using central finite difference at position x_i. Contrary to section 3.2, the first order spatial derivative using forward difference at time step t_n

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

for $i = 1\ldots, M$ and $n = 0, \ldots, N$. Similarly, we rewrite the expression above by writing the approximation of time step v_i^{n+1} with respect to the approximation at the current time step t_n

$$v_i^{n+1} = av_{i-1}^n + bv_i^n + cv_{i+1}^n + \frac{\bar{S}_f^{n+1}-\bar{S}_f^n}{2\Delta x \bar{S}_f^n}(v_{i+1}^n - v_{i-1}^n) \tag{A.1}$$

where

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

The explicit scheme (A.1) have boundary conditions (See section 2.2.2)

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

for $n = 0, \ldots, N$, and initial condition

$$\textbf{Call:} \qquad v_i^0 = 0$$

$$\textbf{Put:} \qquad v_i^0 = 0$$

Moreover, the contact point condition 2.24 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

$$\textbf{Call:} \qquad \frac{v_{M+2}^n - v_M^n}{2\Delta x^2} = \bar{S}_f^n \tag{A.3}$$

$$\textbf{Put:} \qquad \frac{v_1^n - v_{-1}^n}{2\Delta x} = -\bar{S}_f^n \tag{A.4}$$

Using central finite difference for approximating the contact point condition has the benefit that our scheme will be second order in space. However, we must come up with some way to approximate replace v_{M+2} and v_{-1} in (A.5). By plug the contact condition in (2.18) we have that

$$\textbf{Call:} \qquad \frac{\sigma^2}{2}\frac{\partial^2 v}{\partial x^2}(0^-, \tau) - \frac{\sigma^2}{2}\bar{S}_f(\tau) + r = 0 \tag{A.5}$$

$$\textbf{Put:} \qquad \frac{\sigma^2}{2}\frac{\partial^2 v}{\partial x^2}(0^+, \tau) + \frac{\sigma^2}{2}\bar{S}_f(\tau) - r = 0 \tag{A.6}$$

and by applying using central finite difference we have

$$\textbf{Call:} \qquad \frac{\sigma^2}{2}\frac{v_M^n - 2v_{M+1}^n + v_{M+2}^n}{\Delta x^2} - \left(\frac{\sigma^2}{2} - \delta\right)\bar{S}_f^n + r = 0 \tag{A.7}$$

$$\textbf{Put:} \qquad \frac{\sigma^2}{2}\frac{v_{-1}^n - 2v_0^n + v_1^n}{\Delta x^2} + \left(\frac{\sigma^2}{2} - \delta\right)\bar{S}_f^n - r = 0 \tag{A.8}$$

and by plug equation (A.5) in equation (A.7), we have that

$$\textbf{Call:} \qquad v_M = \alpha - \beta \bar{S}_f^n \tag{A.9}$$

$$\textbf{Put:} \qquad v_1 = \alpha - \beta \bar{S}_f^n \tag{A.10}$$

$$\tag{A.11}$$

where

$$\textbf{Call:} \qquad \alpha := -\Big(1 + \frac{r\Delta x^2}{\sigma^2}\Big), \quad \beta := \Delta x - 1 - \frac{1}{2}\Delta x^2 - \frac{\delta}{\sigma^2}\Delta x^2 \tag{A.12}$$

$$\textbf{Put:} \qquad \alpha := 1 + \frac{r\Delta x^2}{\sigma^2}, \quad \beta := 1 + \Delta x + \frac{1}{2}\Delta x^2 + \frac{\delta}{\sigma^2}\Delta x^2 \tag{A.13}$$

By combining all the equations above, we define algorithms for approximating $v(x,T)$ explicitly

Algorithm A.1 Explicit method for call options

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

$v_i^0 = 0$

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

Define a, b, c and λ as in (A.2)

Define α and β as in (A.12)

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

$$d^n = \frac{\alpha - (av_0^n + bv_1^0 + cv_2^n - (v_{M+1}^n - v_{M-1}^n)/(2\Delta x))}{(v_{M+1}^n - v_{M-1}^n)/(2\Delta x) + \beta \bar{S}_f^n}$$

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

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

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

$$v_0^{n+1} = 0$$

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

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

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

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

Algorithm A.2 Explicit method for put options

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

$v_i^0 = 0$

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

Define a, b, c and λ as in (A.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**

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

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
        ):
```

```
42            self.x_axis = np.arange(0, 1+dx, dx)
43            super().__init__(option, r, sigma, dx, dt, delta)
44        def compute_time_iteration(self, V: np.ndarray, D: np.ndarray):
45            S_bar = self.option.K + self.A[-2]*V[-3] + self.B[-2]*V[-2]
    + self.C[-2]*V[-1]
46            S_bar /= 1 - self.dx - D[-2]
47
48            V[1:-2] = self.A[1:-2]*V[:-3] + self.B[1:-2]*V[1:-2] \
49                + self.C[1:-2]*V[2:-1] + D[1:-2]*S_bar
50            V[-2] = (1-self.dx)*S_bar - self.option.K
51            V[-1] = S_bar - self.option.K
52            return S_bar
53     class PutOptionExplicitSolver(ExplicitSolver):
54        def __init__(self,
55            option: PutOption,
56            r: float,  # risk-free interest rate
57            sigma: float,  # sigma price volatitliy
58            dx: float,  # grid resolution along x-axis
59            dt: float,  # grid resolution along t-axis
60            x_max: 2.,  # sufficiently large value for x
61            delta=0  # dividends
62        ):
63            self.x_axis = np.arange(1, x_max+dx, dx)
64            super().__init__(option, r, sigma, dx, dt, delta)
65        def compute_time_iteration(self, V: np.ndarray, D: np.ndarray)
    -> float:
66            S_bar = self.option.K - (self.A[1]*V[0] + self.B[1]*V[1] +
    self.C[1]*V[2])
67            S_bar /= D[1] + 1 + self.dx
68            V[2:-1] = self.A[2:-1]*V[1:-2] + self.B[2:-1] * \
69                V[2:-1] + self.C[2:-1]*V[3:] + D[2:-1]*S_bar
70            V[0] = self.option.payoff(S_bar)
71            V[1] = self.option.payoff((1+self.dx)*S_bar)
72            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
    def solve_non_linear_system(self, V, S_bar) -> float:
        pass
    def solve(self):
        S_bar = self.option.K
        V = np.zeros_like(self.x_axis)
```

```
        for _ in np.arange(0, self.option.T, self.dt):
            S_bar = self.solve_non_linear_system(V, S_bar)
        return S_bar*self.x_axis, V, S_bar
class CallOptionImplicitSolver(ImplicitSolver):
    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
        maxiter=1000,
        tolerance=1e-24,
        method='lm'
    ) -> None:
        self.x_axis = np.arange(0, 1+dx, dx)
        super().__init__(option, r, sigma, dx, dt, delta)
        self.maxiter=maxiter
        self.tolerance=tolerance
        self.method=method
    def jacobian(self, y, S_bar):
        p, s, = y[:-1], y[-1]
        beta = self.beta(s, S_bar)
        gamma = self.gamma(s, S_bar)
        dgamma_ds = - 0.5 * (1/self.dx) * self.x_axis * S_bar / s**2
        dbeta_ds = 0.5 * (1/self.dx) * self.x_axis * S_bar / s**2
        retVal = diags([self.alpha[2:], beta[1:-1], gamma[1:-1]],
                    [-1, 0, 1], shape=(self.M-2, self.M-2)).toarray()
        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)
64          retVal[-1, -2] = self.alpha[-2]
65          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)
66          return retVal
67      def system(self, y, b, S_bar):
68          *v, s = y
69          beta = self.beta(s, S_bar)
70          gamma = self.gamma(s, S_bar)
71          A = diags([self.alpha[2:-1], beta[1:-2], gamma[1:-3]],
72                  [-1, 0, 1], shape=(self.M-2, self.M-3)).toarray()
73          f = b[1:-1]
74          f[-2] -= gamma[-3]*((1-self.dx)*s-self.option.K)
75          f[-1] -= gamma[-2]*(s-self.option.K) + beta[-2]*((1-self.dx)*s -
     self.option.K)
76          res = A@v - f
77          return res
78      def solve_non_linear_system(self, V: np.ndarray, S_bar: np.ndarray):
79          *V[1:-2], S_bar = root(
80              lambda y: self.system(y, np.copy(V[:]), S_bar),
81              # jac=lambda y: self.jacobian(y, S_bar),
82              x0=np.concatenate([V[1:-2], [S_bar]]),
83              method=self.method,
84              options=dict(xtol=self.tolerance, maxiter=self.maxiter)
85          )['x']
86          V[-2] = (1-self.dx)*S_bar-self.option.K
87          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
```

```
        dt: float,  # grid resolution along t-axis
        delta=0, # dividends
        x_max=2,
        maxiter=1000,
        tolerance=1e-24,
        method='lm'
    ) -> None:
        self.x_axis = np.arange(1, x_max+dx, dx)
        super().__init__(option, r, sigma, dx, dt, delta)
        self.maxiter=maxiter
        self.tolerance=tolerance
        self.method=method
    def jacobian(self, y, S_bar):
        p, s, = y[:-1], y[-1]
        beta = self.beta(s, S_bar)
        gamma = self.gamma(s, S_bar)

        dgamma_ds = - 0.5 * (1/self.dx) * self.x_axis * S_bar / s**2
        dbeta_ds = 0.5 * (1/self.dx) * self.x_axis * S_bar / s**2

        retVal = diags([beta[3:-1], self.alpha[2:-1], gamma[1:-1]],
                    [-2, -1, 0], shape=(self.M-2, self.M-2)).toarray()
        retVal[-1, :] = 0
        retVal[:, -1] = 0
        retVal[0, -1] = dgamma_ds[1]*p[0] + dbeta_ds[1] * \
            (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
```

Listing B.4: Explicit solver for Company transformation.

```
class ExplicitSolver(Solver):
    def __init__(
        self,
        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
    ):
        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.A = np.power(self.sigma, 2)
        self.A -= (self.r-self.delta-0.5*np.power(self.sigma,2))*self.dx
        self.A *= self.lambd / 2
        self.B = 1 - np.power(self.sigma, 2) * self.lambd - self.r*self.
    dt
        self.C = np.power(self.sigma, 2)
        self.C += (self.r-self.delta-0.5*np.power(self.sigma,2))*self.dx
        self.C *= self.lambd / 2
    @abstractmethod
    def compute_time_iteration(self, p: np.ndarray, S_bar: float) ->
    float:
        pass
    def solve(self, option: Option):
            """Solve front-fixing method explicitly.

            Parameters:
                option (AmericanOption): the option to price.
                r (float): the stock price risk-free interest rate.
                sigma (float): the stock price volatility.
                x_max (float): large value used as infity in the spatial
    .
                dx (float): Grid resolution in the spatial direction.
                dt (float): Grid resolution in the time direction.
            """
            p = np.zeros_like(self.x_axis)
            S_bar = 1
            for _ in np.arange(0, option.T, self.dt):
                S_bar = self.compute_time_iteration(p, S_bar)
            S = S_bar * np.exp(self.x_axis)
            return option.K*S, option.K*p, S_bar
class CallExplicitSolver(ExplicitSolver):
```

```
    def __init__(
        self,
        r: float,  # risk-free interest rate
        sigma: float,  # sigma price volatitliy
        x_min: float,  # sufficiently large value for x
        dx: float,  # grid resolution along x-axis
        dt: float,  # grid resolution along t-axis
        delta=0  # dividends
    ):
        self.x_axis = np.arange(x_min, dx, dx)
        self.x_axis[-1] = 0
        super().__init__(r=r, sigma=sigma, dx=dx, dt=dt, delta=delta)
        self.alpha = -self.r*np.power(self.dx/self.sigma, 2) - 1
        self.beta = -1 + self.dx - 0.5*np.power(self.dx, 2) - np.power(
    self.dx/self.sigma, 2)*self.delta
    def compute_time_iteration(self, p: np.ndarray, S_bar: float) ->
    float:
        d = self.alpha - (self.A*p[-3] + self.B*p[-2] + self.C*p[-1] - (
    p[-1]-p[-3])/(2*self.dx))
        d /= (p[-1]-p[-3])/(2*self.dx) + self.beta*S_bar
        S_bar_new = d*S_bar
        a = self.A - (S_bar_new - S_bar)/(2*self.dx*S_bar)
        c = self.C + (S_bar_new- S_bar)/(2*self.dx*S_bar)
        S_bar = S_bar_new
        p[1:-2] = a*p[:-3] + self.B*p[1:-2] + c*p[2:-1]
        p[-2] = self.alpha - self.beta*S_bar
        p[-1] = S_bar - 1
        return S_bar_new
class PutExplicitSolver(ExplicitSolver):
    def __init__(
        self,
        r: float,  # risk-free interest rate
        sigma: float,  # sigma price volatitliy
        x_max: float,  # sufficiently large value for x
        dx: float,  # grid resolution along x-axis
```

```
        dt: float,  # grid resolution along t-axis
        delta=0  # dividends
    ):
        self.x_axis = np.arange(0, x_max+dx, dx)
        super().__init__(r=r, sigma=sigma, dx=dx, dt=dt, delta=delta)
        self.alpha = 1 + self.r*np.power(self.dx/self.sigma, 2)
        self.beta = 1 + self.dx + 0.5*np.power(self.dx, 2) + np.power(
    self.dx/self.sigma, 2)*self.delta
    def compute_time_iteration(self, p: np.ndarray, S_bar: float) ->
    float:
        d = self.alpha - (self.A*p[0] + self.B*p[1] + self.C*p[2] - (p
    [2]-p[0])/(2*self.dx))
        d /= (p[2]-p[0])/(2*self.dx) + self.beta*S_bar
        S_bar_new = d*S_bar

        a = self.A - (S_bar_new - S_bar)/(2*self.dx*S_bar)
        c = self.C + (S_bar_new- S_bar)/(2*self.dx*S_bar)

        p[2:-1] = a*p[1:-2] + self.B*p[2:-1] + c*p[3:]
        p[1] = self.alpha - self.beta*S_bar_new
        p[0] = 1 - S_bar_new
        return S_bar_new
```

Listing B.5: PSOR-LCP solver for Company transformation.

```
class LCPSolver:
    def __init__(
        self,
        r: float,
        sigma: float,
        dx: float,
        dt: float,
        x_min = -3.,
        x_max = 3.,
        theta = 0.,
        delta=0.
```

```
    ):
        self.r = r
        self.sigma = sigma
        self.dx = dx
        self.dt = dt
        self.x_min = x_min
        self.x_max = x_max
        self.theta = theta
        self.delta = delta
        self.x_axis = np.arange(self.x_min, self.x_max + self.dx, step=
    self.dx)
    @abstractmethod
    def get_g(self, x_axis, tao_axis, q_delta, q):
        pass
    def solve(
        self,
        option: Option,
        wR = 1,
        eps = 1e-24
    ):
        # Parameters artificail space
        q = (2*self.r) / np.power(self.sigma, 2)
        q_delta = 2*(self.r-self.delta) / np.power(self.sigma, 2)
        t_axis = np.arange(0, option.T+self.dt, step=self.dt)
        tao_axis = 0.5*np.power(self.sigma,2)*t_axis
        dtao = np.diff(tao_axis)[0]
        # calculate lambda and alpha
        lambd = dtao/np.power(self.dx, 2)
        alpha = lambd*self.theta
        g = self.get_g(self.x_axis, tao_axis, q_delta, q)
        # compute initial condition
        w = np.empty(self.x_axis.size)
        w = g[:, 0]
        b = np.empty_like(self.x_axis)
        v_new = np.empty_like(self.x_axis)
```

```
46          for i in range(tao_axis.size - 1):
47              b[2:-2] = w[2:-2] + lambd*(1-self.theta)*(w[3:-1] - 2*w
    [2:-2] + w[1:-3])
48              b[1] = w[1] + lambd*(1-self.theta)*(w[2] -2*w[1] + g[0, i])
    + alpha*g[0, i+1]
49              b[-2] = w[-2] + lambd*(1-self.theta)*(g[-1, i]-2*w[-2] + w
    [-3]) + alpha*g[-1, i+1]
50              v = np.maximum(w, g[:, i+1])
51              while True:
52                  v_new[0] = v_new[-1] = 0
53                  for j in range(1, v.size - 1):
54                      rho = b[j] + alpha*(v_new[j-1] + v[j+1])
55                      rho /= 1+2*alpha
56                      v_new[j] = np.maximum(g[j, i+1], v[j] + wR*(rho - v[
    j]))
57                  if np.linalg.norm(v_new - v, ord=2) <= eps:
58                      break
59                  v = v_new.copy()
60              w = v.copy()
61          S_axis = option.K*np.exp(self.x_axis)
62          S_axis[0] = 0
63          V = self.get_surface(option, w, tao_axis, q, q_delta)
64          return S_axis, V
65  class CallLCPSolver(LCPSolver):
66      def __init__(self, r: float, sigma: float, dx: float, dt: float,
    x_min=-3, x_max=3, theta=0, delta=0):
67          super().__init__(r, sigma, dx, dt, x_min, x_max, theta, delta)
68      def get_g(self, x_axis, tao_axis, q_delta, q):
69          return CallLCPSolver.GUtil(x_axis, tao_axis, q_delta, q)
70      def get_early_exercise(self, option, S, V):
71          eps = 1e-5
72          S_bar = S[np.abs(option.K - S + V) <= eps][-1]
73          return S_bar
74      def get_surface(self, option, w, tao_axis, q, q_delta, ):
75          V = option.K*w*np.exp(-(self.x_axis/2)*(q_delta - 1)) * np.exp(-
```

```
    tao_axis[-1]*((1/4)*np.power(q_delta - 1, 2) + q))
76          V[0] = 0
77          return V[:]
78      class GUtil:
79          def __init__(self, x, tao, q_delta, q):
80              self.x = x
81              self.tao = tao
82              self.q_delta = q_delta
83              self.q = q
84          def __getitem__(self, indx):
85              xi, ti = indx
86              return self.payoff(self.x[xi], self.tao[ti])
87          def payoff(self, x, tao): return np.exp((tao/4)*(np.power(self.
    q_delta-1, 2) + 4*self.q)) * \
88              np.maximum(np.exp((x/2)*(self.q_delta + 1)) - np.exp((x/2)*(
    self.q_delta - 1)), 0)
89  class PutLCPSolver(LCPSolver):
90      def __init__(self, r: float, sigma: float, dx: float, dt: float,
    x_min=-3, x_max=3, theta=0, delta=0):
91          super().__init__(r, sigma, dx, dt, x_min, x_max, theta, delta)
92      def get_g(self, x_axis, tao_axis, q_delta, q):
93          return PutLCPSolver.GUtil(x_axis, tao_axis, q_delta, q)
94      def get_early_exercise(self, option, S, V):
95          eps = 1e-5
96          S_bar = S[np.abs(V + S - option.K) <= eps][-1]
97          return S_bar
98      def get_surface(self, option, w, tao_axis, q, q_delta):
99          V = option.K*w*np.exp(-(self.x_axis/2)*(q_delta - 1)) * np.exp(-
    tao_axis[-1]*((1/4)*np.power(q_delta - 1, 2) + q))
100         V[0] = option.K
101         return V
102     class GUtil:
103         def __init__(self, x, tao, q_delta, q):
104             self.x = x
105             self.tao = tao
```

```
106             self.q_delta = q_delta
107             self.q = q
108 
109         def __getitem__(self, indx):
110             xi, ti = indx
111             return self.payoff(self.x[xi], self.tao[ti])
112 
113         def payoff(self, x, tao): return np.exp((tao/4)*(np.power(self.
    q_delta-1, 2) + 4*self.q)) * \
114             np.maximum(np.exp((x/2)*(self.q_delta - 1)) - np.exp((x/2)*(
    self.q_delta + 1)), 0)
```

C 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
```

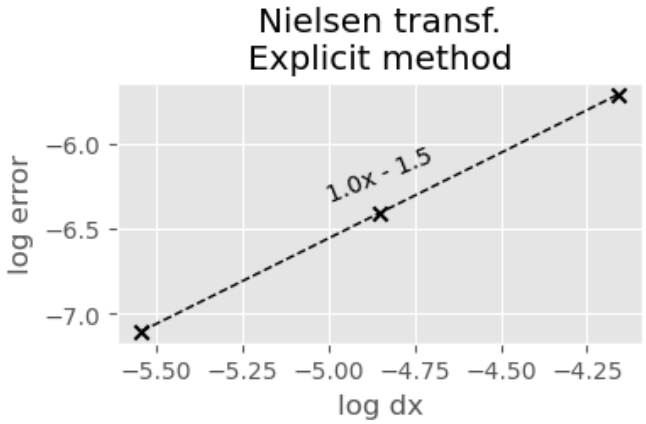
2 Front fixing method

2.1 Nielsen transformation

2.1.1 Explicit

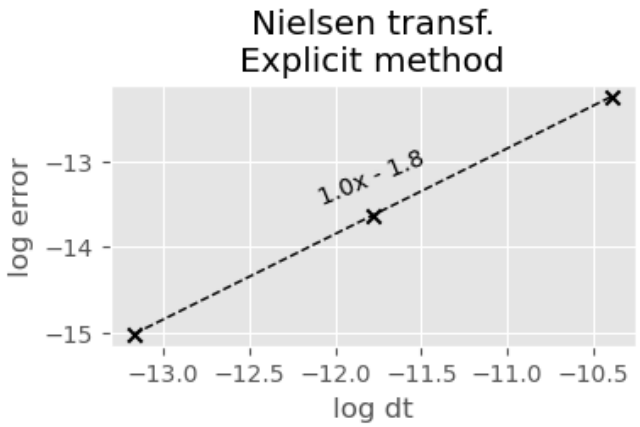
Space

```
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

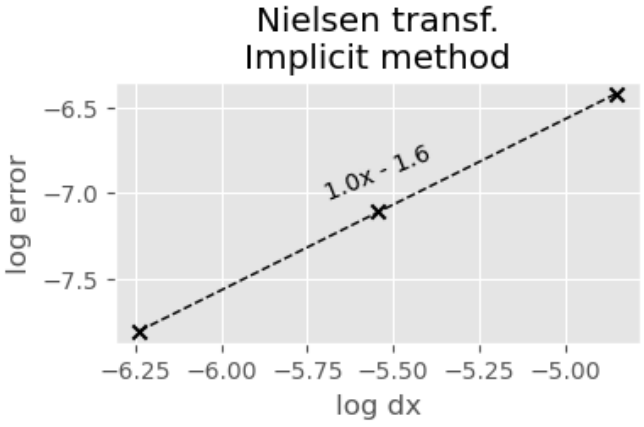
```
dx = 1/128
dt = 0.5*np.power([1/128, 1/256, 1/512, 1/1024],2)
V_pred = time_convergence_analysis(nielsen.solve_explicitly, dx, dt, option␣
 ↪=put, r=r, sigma=sigma, x_max=3,delta=delta)
plt.figure(figsize=(4,2))
plot_time_convergence("Nielsen transf.\nExplicit method", dt, V_pred)
```



2.1.2 Implicit

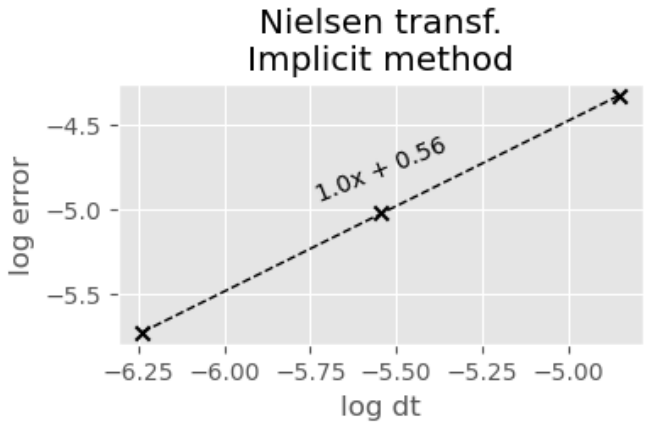
Space

```
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)
```

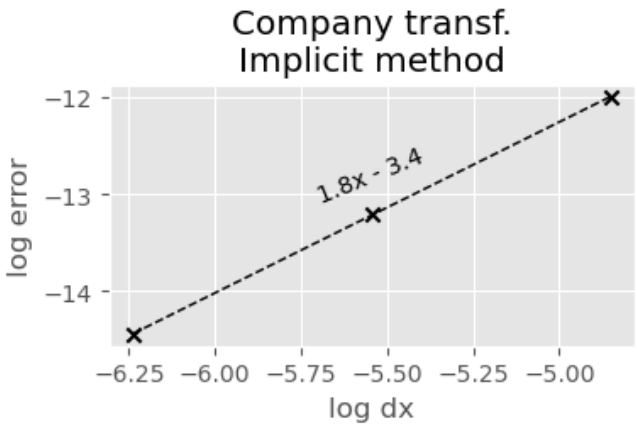


2.2 Company transformation

2.2.1 Explicit

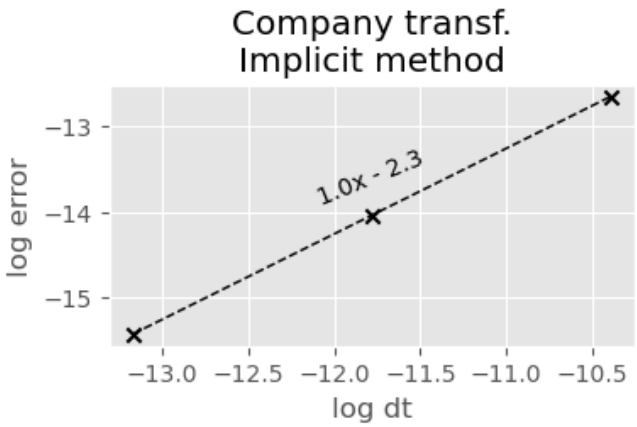
Space

```
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)
```

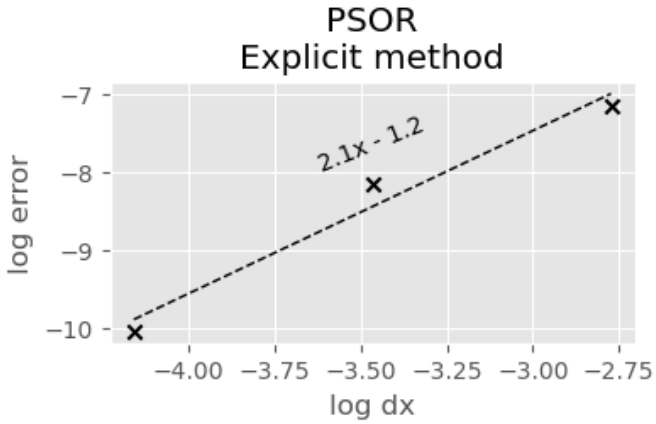


3 LCP + PSOR

3.1 Explicit

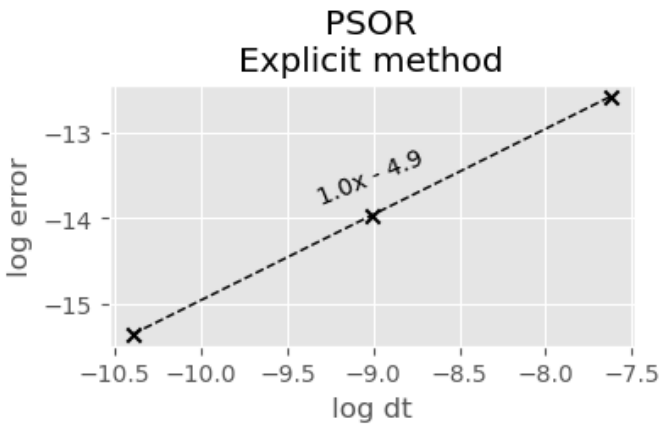
3.1.1 Space

```
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

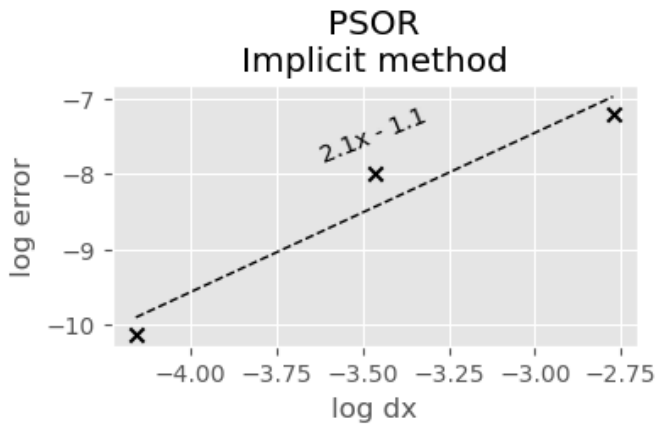
```
dx = 1/32
dt = 0.5*np.array([1/32, 1/64, 1/128, 1/256])**2
V_pred = time_convergence_analysis(lcp.solve, dx, dt, option=put, r=r,␣
 ↪sigma=sigma, theta=0, delta=delta)
plt.figure(figsize=(4,2))
plot_time_convergence("PSOR\nExplicit method", dt, V_pred)
```



3.2 Implicit

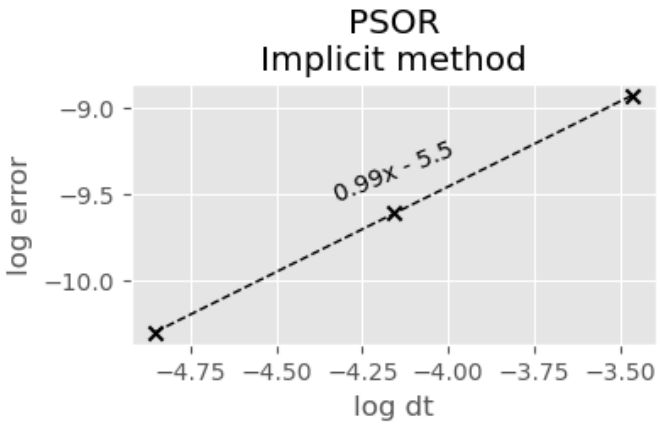
Space

```
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

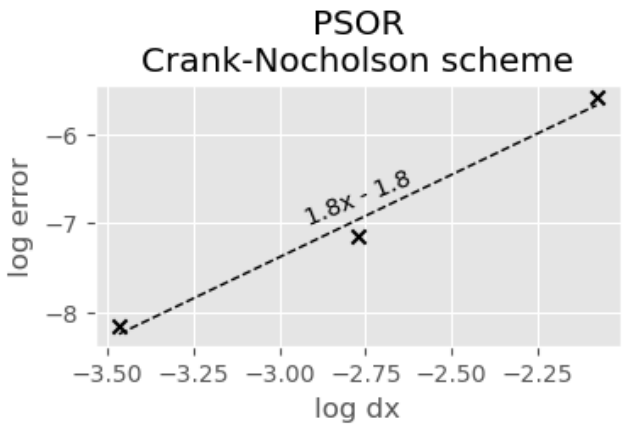
```
dx = 1/16
dt = [1/32, 1/64, 1/128, 1/256]
V_pred = time_convergence_analysis(lcp.solve, dx, dt, option=put, r=r,␣
 ↪sigma=sigma, theta=1, delta=delta)
plt.figure(figsize=(4,2))
plot_time_convergence("PSOR\nImplicit method", dt, V_pred)
```



3.3 Crank nicholson

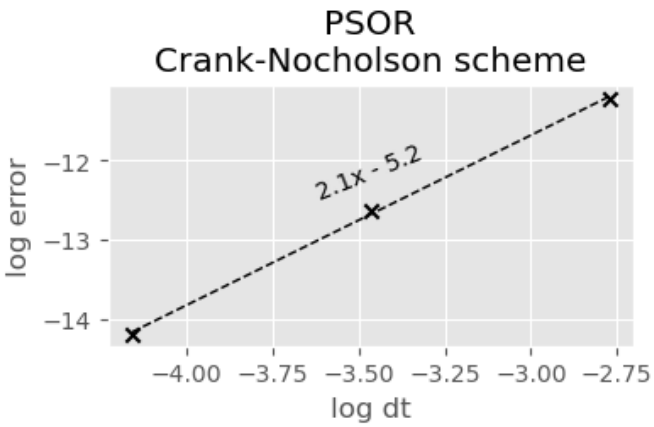
Space

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

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

[9] "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.

[10] 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.

[11] William La Cruz, José M. Martínez, and Marcos Raydan. "Spectral Residual Method without Gradient Information for Solving Large-Scale Nonlinear Systems of Equations". In: *Mathematics of Computation* 75.255 (2006), pp. 1429–1448. ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/4100282 (visited on 08/30/2023).

[12] Hyman 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.

[13] 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).

[14] N. Nassif and D.K. Fayyad. *Introduction to Numerical Analysis and Scientific Computing*. CRC Press, 2016. ISBN: 9781466589490. URL: https://books.google.co.uk/books?id=93DSBQAAQBAJ.

[15] 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.

[16] 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.

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

[18] 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.