University of Padua

SCHOOL OF ENGINEERING
FACULTY OF MATHEMATICAL ENGINEERING
Master Degree in Financial Engineering

Master Thesis

Option Pricing through Numerical Methods for PDEs

An Object-Oriented Python Implementation

Candidate:
Alessandro Reami

Thesis advisor:
Luca Bergamaschi

Academic Year 2023-2024

Abstract

This thesis explores the numerical methods used for solving partial differential equations (PDEs) that arise in financial mathematics, specifically in the context of option pricing. We begin by introducing the Black-Scholes framework, which serves as the foundation for pricing European options, as well as more complex derivatives such as Barrier and Asian options. The work then delves into the finite difference and finite element methods (FEM), focusing on their application to the Black-Scholes PDE and its variants.

Our approach includes the development of an object-oriented Python implementation that leverages the NumPy and SciPy libraries for efficient numerical computation. We also integrate the PyFreeFem API to utilize the capabilities of the FreeFem++ language for finite element analysis. Through this implementation, we solve a variety of option pricing problems, examining the convergence and accuracy of different numerical methods.

The results demonstrate the effectiveness of these methods in approximating option prices under various market conditions. We provide detailed comparisons between numerical and analytical solutions where available, as well as an analysis of the computational efficiency of different techniques. The thesis concludes with a discussion on the limitations of the current implementation and suggestions for future enhancements, particularly in extending the framework to more complex financial instruments and higher-dimensional problems.

Contents

Introduction

In this introductory section, we will summarize the main concepts that underpin the subsequent chapters, aiming to present the problem, explore how it can be approached, and demonstrate its resolution using the associated partial differential equations (PDEs) for specific types of options. To ensure this work is self-contained, we will establish notation rules that will assist the reader in understanding how the different chapters interconnect.

The first section introduces the Black-Scholes framework and the derivation of the Feynman-Kač PDE for European options. Additionally, we will discuss the PDEs relevant to other types of options, such as Barrier Options, Asian Options, and American Options.

The second chapter focuses on a review of the most commonly used numerical methods for solving PDEs: the finite difference method and the finite element method (FEM). While these topics are broad, we will narrow our focus to the specific aspects that are essential for the applications discussed later in this work.

These chapters are not intended to introduce new concepts to the expert reader but rather to provide a focused overview of the necessary background. This will ensure that all the required results are clearly stated and contextualized within the scope of this thesis. We encourage all readers to go through these introductory sections to grasp the overarching ideas that have guided this project.

In preparing the first two chapters, we have relied heavily on two seminal works in the field. For the introduction to the Black-Scholes model, we have drawn primarily from Bj"ork's *"Arbitrage Theory in Continuous Time"* [5]. For the formalism of numerical methods, we have referred to Achdou and Pironneau's *"Computational Methods for Option Pricing"* [2]. In the course of the discussion, we will omit certain technical details and proofs that are beyond the scope of this work; these can be found in the aforementioned texts or other cited sources.

Chapter 1

Black-Scholes model: a brief recap

1.1 Single Asset Process

As mentioned in the introduction, we will dedicate this preliminary section to a summary of the Black-Scholes model, first introduced in 1973 [12]. Given the extensive theory developed following this initial paper, we will focus on highlighting the key aspects of the well-known results and emphasizing the connection between the pricing problem and the solution of a specific PDE arising from the model. In this preliminary setting, we will concentrate on European Options. Later, we will generalize our model to include more complex options such as Barrier Options, Asian Options, options depending on several underlying assets, and American Options.

Let's now consider a simple setting where the Black-Scholes model holds true. In this case, we must assume that the following hypotheses are satisfied, though it is well-known that these assumptions are not generally true.

Hypothesis 1.1.

- Consider a dividend-free asset $S_t = S(t)$ at time t, following the geometric Brownian motion law:

$$dS_t = \mu S_t \, dt + \sigma S_t \, dW_t \tag{1.1}$$

  where the volatility σ and the drift term μ are constants, and W_t is a standard Brownian motion.

- The risk-free interest rate is constant: $r = r_t = r(t) \, \forall \, t$.

- Short selling is always allowed, it is possible to trade arbitrarily small fractions of any financial instrument, and there are no transaction costs. This assumption is usually referred to as the "absence of friction hypothesis."

- There are no arbitrage opportunities (for *market efficiency*).

We already know that these assumptions are not generally true; for example, it is known that σ is not constant, as it is usually a function of the moneyness S_t/K and has a skewed shape (also called a *smile*), and that the risk-free interest

rate (if it exists) is almost never constant. These two aspects are not significant limitations because a similar discussion can be developed even with non-constant volatility and interest rates. On the other hand, the market efficiency and absence of friction hypotheses are also not generally true, and other papers attempt to discuss these scenarios.

Observation 1.2. Supposing that the asset follows the geometric Brownian motion law 1.1, it is possible to define the **Stochastic Differential Equation** (SDE):

$$\begin{cases} dS_t = \mu S_t\, dt + \sigma S_t\, dW_t \\ S_0 = s_0 \end{cases} \tag{1.2}$$

This can be rewritten as

$$\frac{dS_t}{S_t} = \mu\, dt + \sigma\, dW_t$$

implying that the real assumption of the Black-Scholes model is that the returns of S_t follow an Arithmetic Brownian motion with drift μ and diffusion σ.

To complete our setting, we introduce a risk-free asset, which has no risk and provides a return related to the risk-free interest rate r_t.

Definition 1.3 (*Risk-free asset*)**.** Since we assumed that a risk-free interest rate r_t exists, we can define the **risk-free asset** B_t as

$$\begin{cases} \dfrac{dB_t}{B_t} = r_t dt \\ B_0 = 1 \end{cases},$$

Furthermore, we also assumed that this interest rate is constant, so $r_t = r$ for all t.

It is easy to notice that this SDE has the solution $B_t = e^{rt}$, representing the continuous compounding formula, and that the risk-free asset is essentially a zero-coupon bond (appropriately rescaled).

Proposition 1.4. The SDE related to the Geometric Brownian motion 1.2 has a closed-form solution:

$$S_t = s_0\, e^{\mu t + \sigma W_t - \frac{1}{2}\sigma^2 t}. \tag{1.3}$$

Moreover, it has a closed-form expected value given by:

$$\mathbb{E}\left[S_t\right] = s_0\, e^{\mu t}.$$

Proof. Recalling that the solution of the corresponding linear equation is an exponential function of time, we consider the transformation $X_t = \ln(S_t)$ (since the process S_t is strictly positive by construction). Applying Ito's Lemma, we have that:

$$\begin{aligned} dX &= \frac{1}{S} dS + \frac{1}{2}\left(-\frac{1}{S^2}\right)[dS]^2 \\ &= \frac{1}{S}\left(\mu S dt + \sigma S dW\right) - \frac{1}{2S^2}\sigma^2 S^2 dt \\ &= \left(\mu dt + \sigma dW\right) - \frac{1}{2}\sigma^2 dt \end{aligned}$$

Thus, we have:

$$\begin{cases} dX_t = \left(\mu - \frac{1}{2}\sigma^2\right) dt + \sigma dW \\ X_0 = \ln s_0 \end{cases}$$

This equation is extremely simple: on the right-hand side, it does not contain X, and therefore, it can be integrated directly to:

$$X_t = \ln s_0 + \left(\mu - \frac{1}{2}\sigma^2\right) t + \sigma W_t$$

Returning to S_t, we have:

$$S_t = s_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma W_t}.$$

Computing $\mathbb{E}[S_t]$ is then straightforward, recalling that $e^{\sigma W_t}$ is a log-normal random variable with a known expected value $e^{\frac{\sigma^2}{2}t}$. □

We have now derived the two dynamics fundamental to define the **Black-Scholes Model**: the risk-free asset B_t and the risky one S_t:

$$\begin{cases} dB_t = rB_t dt \\ dS_t = \mu S_t dt + \sigma S_t dW_t \end{cases} \tag{1.4}$$

From Proposition 1.4, it is clear that the discounted risky asset $e^{-r(T-t)}S_t$ cannot be a martingale unless $\mu = r$, which is generally not true. Moreover, the value of μ is typically unknown, and different market agents may have different opinions on its value. Indeed, the value of μ is subjective, as is the probability density $\mathbb{P}$ used to compute the expected value $\mathbb{E}$. With this consideration in mind, it is clear that, if we want results acceptable to all market agents, we need to change the probability density to the so-called **risk-neutral probability** $\mathbb{Q}$: a new probability density depending only on the market and not on agents' opinions, where the probability of upward and downward movements is no longer subjective but objective.

Under $\mathbb{Q}$, all discounted risky assets behave as martingales, meaning that the drift of each asset is equal to the risk-free rate used for discounting. To achieve this density, it is sufficient to define the stochastic process $\tilde{W}_t = W_t + \frac{\mu - r}{\sigma}t$ and the Radon-Nikodym density

$$\left(\frac{d\mathbb{Q}}{d\mathbb{P}}\right)_{\mathcal{F}_t} = \exp\left\{\int_0^t \frac{r-\mu}{\sigma}\, dW_s - \frac{1}{2}\int_0^t \left(\frac{r-\mu}{\sigma}\right)^2 ds\right\}.$$

In this new probability space, the Black-Scholes model initially stated under $\mathbb{P}$ in 1.4 is equivalent under $\mathbb{Q}$ to

$$\begin{cases} dB_t = rB_t dt \\ dS_t = rS_t dt + \sigma S_t d\tilde{W}_t \end{cases}$$

where we see that the drift is now given by r and not by μ.

For the following discussion, we will always consider, without loss of generality, the model under the risk-neutral density $\mathbb{Q}$, and we will then use r instead of μ to denote the drift term.

Definition 1.5 (*Contingent claim*)**.** Consider a financial market with an asset S_t. A **contingent claim** with *maturity date* (exercise date) T is any stochastic variable $\mathcal{X}$ measurable with respect to the filtration $\mathcal{F}_T^S$ induced by the underlying S_t at time T. A contingent claim is called **simple** if it has the form $\mathcal{X} = \Phi(S(T))$, where the function Φ is called the **payoff function**.

The interpretation of this definition is that a contingent claim is a contract that stipulates that the holder of the contract will obtain $\mathcal{X}$ (which can be positive or negative) at the time of maturity T. The requirement that $\mathcal{X} \in \mathcal{F}_T^S$ simply means that, at maturity, it is always possible to determine the amount of money corresponding to the payoff, which means that at time T all the information needed to compute the payoff function is known and available to everyone. A simple contingent claim is nothing more than a claim where the payoff only depends on the value of the asset at the maturity instant, which is S_T.

For example, a European Call is a simple contingent claim, for which the payoff function is given by

$$\Phi_{\text{Call}}(S_T) = \max(S_T - K, 0).$$

Our aim in this paper is to price some claims using the Black-Scholes model stated above. We will use the notation $\Pi(t, \mathcal{X})$ for the price process of the claim $\mathcal{X}$, but sometimes we will also use the notation $\Pi(t, \Phi)$ to underline the dependency of the price on the payoff. We will also use $F(t, S_t = s, T, \Phi)$ to denote the price function of a contingent claim. Later on, we will find claims with different dependencies, like Asian Options, but for this initial discussion, this assumption is sufficient for our purposes and does not lead to a loss of generality.

To provide a **fair price** for a contingent claim, a classical approach is to consider a **replicating portfolio**, which is a portfolio able to achieve almost surely (in the measure-theoretic sense, not in the common language sense!) the contingent claim payoff. We will prove that, in the case of an **efficient market** with **absence of arbitrage opportunities**, the value of this portfolio must be equal to the price of the contingent claim. To achieve this conclusion, we need the following definitions.

Definition 1.6 (*Self-financing Portfolio*)**.** Let $\tilde{\boldsymbol{S}}_{\boldsymbol{t}} = (S_1, S_2, \ldots, S_n)$ be an asset vector and $\boldsymbol{\omega_t} = (\omega_1, \omega_2, \ldots, \omega_n)$ a weight vector, both in $\mathbb{R}^n$. The value V_t of a **Self-financing Portfolio** satisfies

$$dV_t = \boldsymbol{\omega_t} d\tilde{\boldsymbol{S}}_{\boldsymbol{t}}$$

i.e., it is a portfolio where the only variations are due to the variation of the asset's value.

Observation 1.7. A self-financing portfolio is uniquely defined by the weight vector $\boldsymbol{\omega_t}$, and we will use the notation $V_t^{\boldsymbol{\omega}}$ to denote it. In these pages, whenever we consider a portfolio, we will always consider a self-financing portfolio because we are interested in portfolios where the only price variation is due to a movement of the underlying and not caused by the consumption/introduction of new capital.

For example, we can build a self-financing portfolio using the two assets B_t and S_t with the vector weight $\boldsymbol{\omega_t} = (\alpha_t, \beta_t)$. We can define the vector $\tilde{\boldsymbol{S}}_{\boldsymbol{t}} = (S_t, B_t)$ and the value of our portfolio as

$$V_t = \boldsymbol{\omega_t}\tilde{\boldsymbol{S}}_{\boldsymbol{t}} = \alpha_t S_t + \beta_t B_t.$$

Definition 1.8 (*Arbitrage opportunity*)**.** An **arbitrage opportunity** in a financial market is a self-financing portfolio with weight $\boldsymbol{\omega_t}$ such that

$$\begin{aligned} V_0^{\boldsymbol{\omega}} &= 0 \\ P(V_T^{\boldsymbol{\omega}} \geq 0) &= 1 \\ P(V_T^{\boldsymbol{\omega}} > 0) &> 0. \end{aligned}$$

We say that there is an **absence of arbitrage opportunities** if there are no arbitrage opportunities.

In short, we can interpret this definition by saying that an arbitrage opportunity is a portfolio with a null initial value that can have a positive value at time T with a positive probability and *never* has a value less than zero at time T.

The absence of arbitrage opportunities is fundamental in this discussion because it enables several results to be proved. For example, the following is a direct consequence of this assumption.

Observation 1.9 (*Linearity of the pricing function on the payoff*)**.** Suppose now that we have the pricing function $F(t, s, T, \Phi)$ for a contingent claim $\mathcal{X}$. It is quite trivial to notice that F is a linear function of the payoff function, which means that $\forall a, b \in \mathbb{R}$ and for all functions Φ, Ψ, we have

$$F(t, s, a\Phi + b\Psi) = aF(t, s, \Phi) + bF(t, s, \Psi).$$

This observation is also quite intuitive using the classical absence of arbitrage opportunity argument: since we can compose one payoff as the sum of two simpler payoffs, the sum of the prices of each term must be equal to the price of the original payoff to avoid arbitrage opportunities.

Returning to the process $\Pi(t, \mathcal{X})$, we assume that the process can be expressed as a smooth price function F of $(t, S(t))$:

$$\Pi(t) = F(t, S(t)),$$

then we look for a function F such that the absence of arbitrage opportunities is preserved. With these assumptions, it was proven in the Black-Scholes paper [12] that the key idea is to impose the existence of a replicating portfolio for the contingent claim payoff, and that the value of the portfolio $V^{\boldsymbol{\omega_t}}$ and the price $\Pi(t)$ must have the same dynamics at each time instant. This is equivalent to saying

$$dV^{\boldsymbol{\omega_t}} = d\Pi(t) = dF(t, S_t),$$

From this equation, it is possible to obtain the fundamental theorem of the Black-Scholes model:

Theorem 1.10 (*Black-Scholes equation*)**.** Assuming that the market is specified by the equation 1.4 and that $\mathcal{X}$ is a simple contingent claim (with $\mathcal{X} = \Phi(S_T)$), the only pricing function $\Pi(t) = F(t, S_t)$ consistent with the absence of arbitrage opportunities is the solution of the following PDE with boundary values on $[0, T] \times \mathbb{R}^+$:

$$\begin{cases} \dfrac{\partial F(t,s)}{\partial t} + rs\dfrac{\partial F(t,s)}{\partial s} + \dfrac{1}{2}s^2\sigma^2(t,s)\dfrac{\partial^2 F(t,s)}{\partial s^2} - rF(t,s) = 0 \\ F(T,s) = \Phi(s) \quad \text{for all } s \in (0, +\infty) \end{cases} \tag{1.5}$$

Clearly, this result is powerful but not without weaknesses. Firstly, we assumed that the price of the contingent claim is a function depending only on (t, S_t), which implies that we supposed a Markovian property of the price. This assumption is generally not convincing, as market agents usually tend to price every instrument also by looking at its historical prices. Another objection is to the hypothesis that there exists a market where the underlying asset is traded; indeed, we assumed that we could replicate the contingent claim with a portfolio on the asset itself, which we can only build if we have access to a market where the stock is traded. However, it is possible to provide an alternative argument for equation 1.5 that is not open to the previous criticism. The surprising fact is that the pricing equation does not contain the local mean rate of return μ of the underlying asset. This means that, when it comes to pricing derivatives, the local rate of return of the underlying asset plays no role whatsoever. The only aspect of the underlying price process that is of any importance is the volatility $\sigma(t, s)$. Thus, for a given volatility, the price of a fixed derivative (like a European Call Option) will be exactly the same regardless of the underlying stock's return.

This alternative procedure uses a **Risk-Neutral Valuation** technique and a Martingale approach: it starts from the stochastic representation formula 1.4 and proves that the Black-Scholes model is arbitrage-free if and only if there exists a local martingale measure where both the assets are local martingales.

Returning to equation 1.5, it is possible to compute the function F using the Feynman-Kač stochastic representation formula.

Observation 1.11. The Black-Scholes PDE 1.5 can also be stated for: a free-risk interest rate function $r(t)$, a volatility function $\sigma(t, s)$, and dividend-paying assets where the dividend is infinitesimal and paid at any time instant. In this case, the dividend in dt is $q(t)S_t dt$, and the PDE becomes

$$\begin{cases} \dfrac{\partial F(t,s)}{\partial t} + (r - q(t))s\dfrac{\partial F(t,s)}{\partial s} + \dfrac{1}{2}s^2\sigma^2(t,s)\dfrac{\partial^2 F(t,s)}{\partial s^2} - rF(t,s) = 0 \\ F(T,s) = \Phi(s) \quad \text{for all } s \in (0, +\infty) \end{cases} \tag{1.6}$$

Theorem 1.12 (*Representation formula for the solution of 1.5*)**.** As in the discrete case, it is also possible in the continuous framework to prove that the price F of a contingent claim can be computed as the expected value of the discounted payoff:

$$F(t, S_t) = e^{-r(T-t)}\,\mathbb{E}^{\mathbb{Q}}_{t,s}[\Phi(S_T)]$$

and thus the solution of 1.5 is this function.

This PDE has the same structure as the classical Black-Scholes PDE if we define $\tilde{r} = r - q(t)$, so, without loss of generality, we will not consider dividend-paying assets in the following discussion.

Theorem 1.13 (*Feynman-Kač*)**.** Assume that F is a solution to the boundary value problem:

$$\begin{cases} \dfrac{\partial F(t,x)}{\partial t} + \mu(t,x)\dfrac{\partial F(t,x)}{\partial x} + \dfrac{1}{2}\sigma^2(t,x)\dfrac{\partial^2 F(t,x)}{\partial x^2} - rF(t,x) = 0 \\ F(T,x) = \Phi(x) \quad \text{for all } x \in (0,+\infty) \end{cases} \tag{1.7}$$

Assume furthermore that the process $e^{-rs}\sigma(s,X)\frac{\partial F}{\partial x}(s,X_s)$ is in L^2, where X satisfies the SDE

$$\begin{cases} dX_s = \mu(s,X_s)ds + \sigma(s,X_s)dW_s \\ X_t = x \end{cases}.$$

Then F has the representation

$$F(t,x) = e^{-r(T-t)}\,\mathbb{E}_{t,x}[\Phi(X_T)].$$

Using this theorem, we find that the desired price function is

$$F(t,s) = e^{-r(T-t)}\,\mathbb{E}_{t,x}[\Phi(X(T))], \tag{1.8}$$

where X is defined by the dynamics:

$$\begin{cases} dX(s) = r(s,X(s))ds + \sigma(s,X(s))dW(s) \\ X(t) = x \end{cases}$$

which has the same dynamics as the process S but with a different drift term r instead of μ.

An important consequence of the representation formula 1.8 for F is that it is possible to compute an analytical fair price for a simple contingent claim when r and σ are constants and the function $\Phi(S_T)$ is simple enough to be integrated. Indeed, recalling the dynamics of the BS model 1.4 under the probability $\mathbb{Q}$, which we report here:

$$\begin{cases} dB_t = rB_t dt \\ dS_t = rS_t dt + \sigma S_t dW_t \end{cases} \tag{1.9}$$

and recalling the solution 1.3 of the geometric Brownian motion with initial condition $S_t = s$

$$S_T = s\,e^{(r-\frac{1}{2}\sigma^2)(T-t)+\sigma(W_T-W_t)},$$

we have the pricing function

$$F(t,s) = e^{-r(T-t)}\int_{-\infty}^{\infty}\Phi(se^z)f(z)\,dz,$$

where f is the density of a random variable Z with normal distribution

$$N\left(\left(r-\frac{1}{2}\sigma^2\right)(T-t),\,\sigma\sqrt{T-t}\right).$$

In general, the integral of the function Φ is not computable and must be evaluated numerically, but in a few particular cases, the solution can be computed analytically. Two known examples where this is possible are for the European Call and Put Options, where the payoff function is respectively $\Phi(x) = (x-K)^+$ and $\Phi(x) = (K-x)^+$.

Example 1.14 (*Black-Scholes formula for European Call and Put Option*)**.** For the European Call Option, we know that $\Phi(x) = (x-K)^+$, then

$$\begin{aligned}\mathbb{E}^{\mathbb{Q}}_{t,s}[(S_T-K)^+] &= \int_{-\infty}^{\infty} (S_T-K)^+ f(z)\,dz \\ &= \int_{-\infty}^{\infty} (se^{(r-\frac{1}{2}\sigma^2)(T-t)+\sigma(W_T-W_t)} - K)^+ f(z)\,dz \\ &= \int_{\ln(\frac{K}{s})}^{\infty} (se^{(r-\frac{1}{2}\sigma^2)(T-t)+\sigma\sqrt{T-t}z} - K) f(z)\,dz\end{aligned}$$

which is an integral easy to compute, using the linearity of the integral operator and completing the square in the exponent, so that we obtain two cumulative distribution functions N of the standard normal distribution $N(0,1)$. At the end, we achieve the famous **Black-Scholes formula**:

$$F(t,s) = sN[d_1(t,s)] - e^{-r(T-t)}KN[d_2(t,s)]$$

with

$$\begin{aligned}d_1(t,s) &= \frac{\ln(\frac{s}{K}) + (r+\frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}} \\ d_2(t,s) &= d_1(t,s) - \sigma\sqrt{T-t}.\end{aligned}$$

A similar computation can be made for the European Put Option, whose payoff function is $\Phi(x) = (K-x)^+$, obtaining:

$$\begin{aligned}\mathbb{E}^{\mathbb{Q}}_{t,s}[(K-S_T)^+] &= \int_{-\infty}^{\infty} (K-S_T)^+ f(z)\,dz \\ &= \int_{-\infty}^{\infty} (K - se^{(r-\frac{1}{2}\sigma^2)(T-t)+\sigma(W_T-W_t)})^+ f(z)\,dz \\ &= \int_{-\infty}^{\ln(\frac{K}{s})} (K - se^{(r-\frac{1}{2}\sigma^2)(T-t)+\sigma\sqrt{T-t}z}) f(z)\,dz\end{aligned}$$

and after some computation, the result is:

$$F(t,s) = Ke^{-r(T-t)}\big(1-N(d_2(s,t))\big) - s\big(1-N(d_1(s,t))\big),$$

with the same d_1 and d_2 as before.

Another famous type of Options is the so-called European Digital (or binary) Options. These Options can be Call Options or Put Options, and they grant the owner the right to earn 1 if the underlying is in a specific region and 0 otherwise. For the Call Option, the owner has the right to get a payoff of 1 if the underlying is above a given strike K at maturity T and 0 if the underlying

is below it (resp. 0 and 1 for the Put Option with the same parameters). Then we have that these payoffs can be expressed as

$$\Phi_{\text{Digital Call}}(S_T) = \mathbb{1}_{S_T>K} = \begin{cases} 1 & S_T > K \\ 0 & \text{otherwise} \end{cases}$$

$$\Phi_{\text{Digital Put}}(S_T) = \mathbb{1}_{S_T\leq K} = \begin{cases} 0 & S_T > K \\ 1 & \text{otherwise} \end{cases}$$

An easy observation of these Options is that a portfolio made by a European Digital Call and a European Digital Put with the same strike K and maturity T will almost surely get a payoff of 1 at maturity because

$$\mathbb{1}_{S_T>K} + \mathbb{1}_{S_T\leq K} = 1,$$

and then we have the Put-Call parity between the Digital Options, taking the pricing function on both sides:

$$F_{\text{Digital Call}}(s,K,T) + F_{\text{Digital Put}}(s,K,T) = e^{-r(T-t)}.$$

Example 1.15 (*Black-Scholes formula for European Call and Put Digital Option*)**.** For these Options, we can derive a closed pricing formula using the formula for European Call and Put Options that we just derived. For a European Call payoff, we have

$$(S_T - K)^+ = (S_T - K)\mathbb{1}_{S_T>K} = S_T\mathbb{1}_{S_T>K} - K\mathbb{1}_{S_T>K}$$

which means that the Call payoff can be expressed in terms of two different terms, one of which is a digital Option multiplied by K. Since during the computation of the Call price the two terms were computed separately, we can conclude from this observation that the price of the European Digital Call is just the second term with $K = 1$. It means its price is

$$F_{\text{Digital Call}}(t,s) = e^{-r(T-t)}N[d_2(t,s)],$$

with d_2 defined above.

Then we can also compute the price of a European Digital Put Option using the Put-Call Parity and obtain:

$$F_{\text{Digital Put}}(t,s) = e^{-r(T-t)} - F_{\text{Digital Call}}(t,s) = e^{-r(T-t)}\left(1 - N[d_2(t,s)]\right).$$

Clearly, the Black-Scholes PDE 1.5 has been studied in detail with extensive literature, and the results we reported here are just a few that are useful to introduce the problem and to give key information about this theory. Several other formulas have been proved for other payoffs or for compositions of these simple Options.

To conclude, we report here a general theorem with some sufficient conditions to ensure the existence and uniqueness of a strong solution for the pricing problem [14] (a strong solution is a solution in the classical sense).

Theorem 1.16 (**∃! strong solution [14]**)**.** There exists a unique strong solution $F \in C^0(\mathbb{R}^+ \times [0,T])$ for the Black-Scholes PDE 1.5 if the following conditions are respected:

- The function $\sigma(t, S)$ is bounded on $\mathbb{R}^+ \times [0, T]$ and bounded from below by a positive constant;
- The function $t \mapsto r(t)$ is bounded and Lipschitz continuous (this assumption can be relaxed);
- The payoff $\Phi(S_T)$ satisfies $0 \leq \Phi(S_T) \leq C(1 + S_T)$ for a given constant C.

Moreover, it is true that F is: C^1-regular with respect to t in $\mathbb{R}^+ \times (0, T]$, C^2-regular with respect to S in $\mathbb{R}^+ \times (0, T]$, and it satisfies $0 \leq F(t, S_t) \leq C'(1+S_t)$ for a given constant C'.

Now that we have seen how it is possible, with the Black-Scholes assumptions, to solve the option pricing problem by solving a PDE numerically (in general), we would like to investigate further the connection between SDEs and PDEs. In particular, we want to focus our attention on the **Infinitesimal Operator**, which is a partial differential operator closely connected with the Ito formula and is useful for our purpose because it is known for its property of connecting the probabilistic properties of a process with a specific PDE.

Definition 1.17 (*Infinitesimal generator*)**.** Consider an n-dimensional SDE:

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dW_t,$$

the **infinitesimal generator** of X, denoted by $\mathcal{A}$, is a partial differential operator defined for every function $h \in C^2(\mathbb{R}^n)$ by

$$\mathcal{A}h(t, x) = \sum_{i=1}^{n} \mu_i(t, x) \frac{\partial h}{\partial x_i}(x) + \frac{1}{2} \sum_{i,j=1}^{n} C_{ij}(t, x) \frac{\partial^2 h}{\partial x_i \partial x_j}(x),$$

where $C(t, x)$ is the matrix

$$C(t, x) = \sigma(t, x)\sigma^T(t, x).$$

(Alternative names for this operator are **Dynkin operator**, **Ito operator**, or **Kolmogorov backward operator**.)

Consider now a 1-dimensional SDE:

$$\begin{cases} dX_s = \mu(s, X_s)ds + \sigma(s, X_s)dW_s \\ X_t = x \end{cases}$$

and its infinitesimal generator $\mathcal{A}$ given by

$$\mathcal{A} = \mu(t, x) \frac{\partial}{\partial x} + \frac{1}{2}\sigma^2(t, x) \frac{\partial^2}{\partial x^2}.$$

If we consider the boundary value problem defined by the following PDE and initial condition $F(T, x) = \Phi(x)$:

$$\frac{\partial F}{\partial t}(t, x) + \mathcal{A}F(t, x) = 0,$$

that in explicit form is:

$$\frac{\partial F}{\partial t}(t,x) + \mu(t,x)\frac{\partial F}{\partial x}(t,x) + \frac{1}{2}\sigma^2(t,x)\frac{\partial^2 F}{\partial x^2}(t,x) = 0,$$

and we apply Ito's lemma to the process $F(s, X(s))$, we obtain:

$$F(T,X_T) = F(t,X_t) + \int_t^T \left\{ \frac{\partial F}{\partial t}(s,X_s) + \mathcal{A}F(s,X_s) \right\} ds + \int_t^T \sigma(s,X_s)\frac{\partial F}{\partial x}(s,X_s)\, dW_s.$$

The time integral vanishes due to our definition of the PDE, and the stochastic integral will also vanish if $\sigma(s,X_s)\frac{\partial F}{\partial x}(s,X_s)$ is sufficiently integrable, and we take the expected value. Applying the boundary condition $F(T,x) = \Phi(x)$ and the initial value $X_t = x$, we conclude that

$$F(t,x) = \mathbb{E}_{t,x}[\Phi(X_T)].$$

In this way, we have proved a simplified version of the Feynman-Kač theorem, which can be generalized to the result we presented earlier with a little more effort. Through these steps, we have demonstrated the role of the infinitesimal generator $\mathcal{A}$ in the derivation of the theorem.

1.2 Multi-Asset Processes

Let us now consider a framework with two risky assets, whose prices are S_1 and S_2 (keeping in mind that all the following discussion can be generalized to a framework with several assets). We assume that the prices of the underlying assets obey the following system of stochastic equations:

$$\begin{cases} dS_{1t} = S_{1t}\left(\mu_1 + \dfrac{\sigma_1}{\sqrt{1+\rho^2}}\left(dW_{1t} + \rho dW_{2t}\right)\right) \\ dS_{2t} = S_{2t}\left(\mu_2 + \dfrac{\sigma_2}{\sqrt{1+\rho^2}}\left(\rho dW_{1t} + dW_{2t}\right)\right) \end{cases} \tag{1.10}$$

where W_1 and W_2 are two independent standard Brownian motions, and ρ is the correlation factor such that $-1 < \rho < 1$. For simplicity, we assume that σ_1 and σ_2 are positive constants, but a generalization to volatilities that are functions $\sigma_1(S_1,S_2,t)$ and $\sigma_2(S_1,S_2,t)$ can be considered. For the same reason, we also assume that the risk-free interest rate r_t is constant.

Consider a European Option on this two-asset basket, whose payoff is $\Phi(S_1,S_2)$. As in the single-asset case, it is possible to find a probability measure $\mathbb{Q}$ under which the price of the option is

$$F(t,S_1,S_2) = e^{-r(T-t)}\,\mathbb{E}^*(\Phi(S_{1T},S_{2T})|\mathcal{F}_t).$$

Applying the two-dimensional Ito's formula, it is possible to recover the price of the option $F(t,S_1,S_2)$ as the solution of a partial differential equation (PDE). Replacing time with time to maturity, we end up with the PDE:

$$\begin{cases} \dfrac{\partial F}{\partial t} - \dfrac{1}{2}\displaystyle\sum_{k,l=1}^{2} M_{k,l}\, S_k S_l \frac{\partial^2 F}{\partial S_k \partial S_l} - \sum_{k=1}^{2} r S_k \frac{\partial F}{\partial S_k} + rF = 0, \quad t \in (0,T],\ S_{1,2} > 0 \\ F(0,S_1,S_2) = \Phi(S_1,S_2), \quad S_1, S_2 > 0 \end{cases} \tag{1.11}$$

where the matrix M,

$$M = \begin{bmatrix} \sigma_1^2 & \frac{2\rho}{1+\rho^2}\sigma_1\sigma_2 \\ \frac{2\rho}{1+\rho^2}\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$$

is clearly positive definite.

Among the usual payoff functions, we can mention, for a Put option:

$$\Phi_0(S_1,S_2) = (K - (S_1+S_2))^+ \tag{1.12}$$

$$\Phi_0(S_1,S_2) = (K - \max(S_1,S_2))^+ \tag{1.13}$$

$$\Phi_0(S_1,S_2) = (K - \min(S_1,S_2))^+ \tag{1.14}$$

We can notice here that the first two payoffs belong to L^2, the space of square integrable functions, while the third does not. Indeed, this last payoff function is constantly equal to K on the whole lines $x = 0, y = 0$. As a consequence of this, the theory of weak solutions that we will study later cannot be applied to this function, while it holds true for the first two payoffs.

For the Call options, we can mention the payoff functions:

$$\Phi_0(S_1,S_2) = ((S_1+S_2) - K)^+ \tag{1.15}$$

$$\Phi_0(S_1,S_2) = (\max(S_1,S_2) - K)^+ \tag{1.16}$$

$$\Phi_0(S_1,S_2) = (\min(S_1,S_2) - K)^+ \tag{1.17}$$

which are clearly not in L^2 since they are all unbounded, for example, on the line $S_1 = S_2$.

It will be useful later to consider the behavior of the payoffs in the degenerate cases $S_1 = 0$ or $S_2 = 0$. It is easy to notice that some payoffs become the one-dimensional usual Call/Put, while others become constant 0 or K.

1.3 Other Options: Barrier, Lookback, Asian, and American Options

1.3.1 Barrier Options

A Barrier Call (resp. Put) Option is a contract that gives its owner the right to buy (resp. sell) a share of a specific stock at a fixed price K at maturity T if a **barrier event** has not occurred. Several types of barrier events exist in the financial markets, the most famous being the **Up-and-Out** with barrier B (resp. the **Down-and-Out** with barrier b), where the barrier event occurs if the underlying price S_i reaches the high value B at any point during the Option's life (resp. if S_i reaches the low value b). Another type of barrier includes the

Up-and-In with barrier B (resp. the **Down-and-In** with barrier b), where the barrier event occurs if the underlying price S_i *does not reach* the high value B at any point during the Option's life (resp. if S_i reaches the low value b). Barrier Options of the first type are also called **Knock-Out** Options, while those of the second type are called **Knock-In** Options.

Observation 1.18 (**In-Out Parity**)**.** It is easy to see that the sum of the payoffs of an Up-and-Out and an Up-and-In Barrier Option with the same barrier B on the same initial contingent claim $\mathcal{Z}$ will equal the payoff of $\mathcal{Z}$. Therefore, the sum of the prices of these two Barrier Options must equal the price of the initial contingent claim, in the absence of arbitrage opportunities. A similar observation holds true for the Down-and-Out and Down-and-In Barrier Options.

To illustrate, consider the Barrier Options on the stock plot in Figure 1.1. In this case, an Up-and-Out Barrier Call Option with a barrier at B and strike $K = S_0$ would not pay out, while a similar Up-and-In Barrier Call Option would. On the other hand, a Down-and-Out Option with a barrier at b would pay out, while a Down-and-In Option would not.

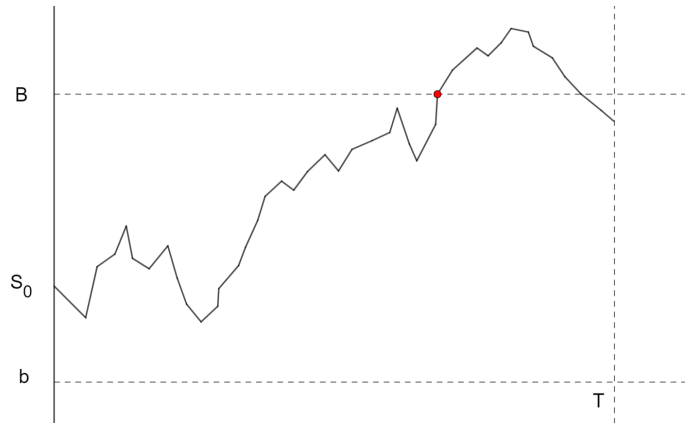


Figure 1.1: Simulation of the evolution of a stock value.

To study these kinds of Options, some definitions are necessary, and we will use the standard results proposed by Borodin-Salminen [6]. Let $\{X(t); 0 \leq t \leq \infty\}$ be a generic process with continuous trajectories taking values in $\mathbb{R}$.

Definition 1.19 (*Hitting Time*)**.** For any $y \in \mathbb{R}$, the **hitting time** of y, $\tau(X, y)$, also denoted by $\tau(y)$ or τ_y, is defined by

$$\tau(y) = \inf\{t \geq 0 \mid X(t) = y\}.$$

The X-process **absorbed at** y is defined by

$$X_y(t) = X(t \wedge \tau),$$

where we have used the notation $a \wedge b = \min(a, b)$.

Definition 1.20 (*Running Maximum, Running Minimum*)**.** The **running maximum** and **minimum** processes $M_X(t)$ and $m_X(t)$ are defined by

$$M_X(t) = \sup_{0 \le s \le t} X(s),$$

$$m_X(t) = \inf_{0 \le s \le t} X(s).$$

Since we are primarily interested in Wiener processes, let X be a Wiener process with constant drift and constant diffusion. It is possible to obtain an explicit density function for the absorbed process $X_y(t)$ and deduce a distribution function for the running maximum and minimum in terms of the Normal distribution. The quoted formulas can be found in [6] and in [5].

With these new tools, it is possible to prove the following theorems, which tell us that the pricing of a Barrier Option can be reduced to the pricing of the related non-barrier claim. This means that if we can price the non-barrier claim, then we can also price its related Barrier Option.

We now introduce some notation: when considering an upper barrier B or a lower barrier b and an *Out* contract O or an *In* contract I on the initial contingent claim $\mathcal{Z}$ and the usual payoff Φ, we will use:

- $\mathcal{Z}^{BO}$ for the **Up-and-Out** related contract;
- $\mathcal{Z}^{BI}$ for the **Up-and-In** related contract;
- $\mathcal{Z}_{bO}$ for the **Down-and-Out** related contract;
- $\mathcal{Z}_{bI}$ for the **Down-and-In** related contract;

and similarly Φ_b and Φ^B for the chopped initial payoffs.

Example 1.21. From a mathematical point of view, the formula representing the payoff of $\mathcal{Z}^{BO}$ is:

$$\mathcal{Z}^{BO} \begin{cases} \Phi(S(T)) & \text{if } \forall\, t \in [0,T], S(t) < B \\ 0 & \text{if } \exists\, t \in [0,T], S(t) \ge B \end{cases} \tag{1.18}$$

We can also define Φ^B as the chopped version of the payoff Φ above the barrier B:

$$\Phi^B(x) \begin{cases} \Phi(x) & \text{for } x < B \\ 0 & \text{for } x \ge B \end{cases} \tag{1.19}$$

With this notation, we can rephrase the **In-Out Parity** in Observation 1.18 as:

$$F_{bI}(t,s,\Phi) + F_{bO}(t,s,\Phi) = F(t,s,\Phi) = F^{BI}(t,s,\Phi) + F^{BO}(t,s,\Phi). \tag{1.20}$$

We already discussed the linearity of the pricing function with respect to the payoff in Observation (1.9). The same linearity also applies to the chopping operation implied in Barrier Options. In formula, we have that $\forall\, a, b \in \mathbb{R}$ and for all functions Φ, Ψ, we have:

$$(a\Phi + b\Psi)_b = a\Phi_b + b\Psi_b,$$

$$(a\Phi + b\Psi)_B = a\Phi_B + b\Psi_B.$$

Theorem 1.22 (Pricing Barrier Contracts). Consider a claim $\mathcal{Z} = \Phi(S(T))$ with pricing function $F(t, s, \Phi)$. The pricing functions of the corresponding Down-and-Out contract for $s < B$ or $s > b$ are given by:

$$\begin{aligned} F_{bO}(t,s,\Phi) &= F(t,s,\Phi_b) - \left(\frac{b}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} F\left(t, \frac{b^2}{s}, \Phi_b\right) \\ F^{BO}(t,s,\Phi) &= F(t,s,\Phi^B) - \left(\frac{B}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} F\left(t, \frac{B^2}{s}, \Phi^B\right) \\ F_{bI}(t,s,\Phi) &= F(t,s,\Phi^b) + \left(\frac{b}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} F\left(t, \frac{b^2}{s}, \Phi_b\right) \\ F^{BI}(t,s,\Phi) &= F(t,s,\Phi_B) + \left(\frac{B}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} F\left(t, \frac{B^2}{s}, \Phi^B\right), \end{aligned} \tag{1.21}$$

where we used:

$$\tilde{r} = r - \frac{1}{2}\sigma^2.$$

Using this theorem, we can also prove that the linearity of F on the payoff implies the linearity of the pricing function for the Barrier Option. In formula, we have:

$$F_{bO}(t, s, a\Phi + b\Psi) = aF_{bO}(t, s, \Phi) + bF_{bO}(t, s, \Psi),$$

which holds true also for the other pricing functions for Barrier Options.

Observation 1.23. These pricing functions are consistent with Observation 1.18 due to the linearity of F:

$$\begin{aligned} F^{BO}(t,s,\Phi) + F^{BI}(t,s,\Phi) &= F(t,s,\Phi^B) + F(t,s,\Phi_B) \\ &= F(t,s,\Phi). \end{aligned}$$

Before considering some examples for (1.21), we recall the payoff and the prices for some basic payoffs like the stock ST, the zero-coupon BO, the digital Call H, the Call C, and the Put P. We have that (in bold are the pricing functions):

$$\begin{cases} ST(x) = x \\ BO(x) = 1 \\ H(x,L) = \begin{cases} 1, & x > L \\ 0, & x \le L \end{cases} \\ C(x,K) = (x-K)^+ \\ P(x,K) = (K-x)^+ \end{cases} \quad \begin{cases} \boldsymbol{ST}(t,s) = s \\ \boldsymbol{BO}(t,s) = e^{-r(T-t)} \\ \boldsymbol{H}(t,s,L) = e^{-r(T-t)} N\left[\dfrac{\tilde{r}(T-t) + \log\left(\frac{s}{L}\right)}{\sigma\sqrt{T-t}}\right] \\ \boldsymbol{C}(x,K) = c(t,s,K) \\ \boldsymbol{P}(x,K) = p(t,s,K) \end{cases} \tag{1.22}$$

where $c(t, s, K)$ and $p(t, s, K)$ are the usual Black-Scholes formulas.

Proposition 1.24 (Down-and-Out Zero Coupon Bond). The Down-and-Out zero coupon bond with barrier b, for $s > b$, is priced as:

$$\boldsymbol{BO}_{bO}(t,s) = \boldsymbol{H}(t,s,b) - \left(\frac{b}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} \boldsymbol{H}\left(t, \frac{b^2}{s}, b\right).$$

Proposition 1.25 (**Down-and-Out Stock**). The Down-and-Out contract on the stock with barrier b, for $s > b$, is priced as:

$$\begin{aligned}\boldsymbol{ST}_{bO}(t,s) = b\Bigg(&\boldsymbol{H}(t,s,b) - \left(\frac{b}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} \boldsymbol{H}\left(t, \frac{b^2}{s}, b\right)\Bigg)\\ &+ \boldsymbol{C}(t,s,b) - \left(\frac{b}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} \boldsymbol{C}\left(t, \frac{b^2}{s}, b\right).\end{aligned}$$

Proposition 1.26 (**Down-and-Out European Call**). The Down-and-Out European Call Option is priced as:

- For $L < K$:

$$\boldsymbol{C}_{bO}(t,s,K) = \boldsymbol{C}(t,s,K) - \left(\frac{b}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} \boldsymbol{C}\left(t, \frac{b^2}{s}, K\right).$$

- For $L > K$:

$$\begin{aligned}\boldsymbol{C}_{bO}(t,s,K) =&\boldsymbol{C}(t,s,K) + (L-K)\boldsymbol{H}(t,s,L)\\ &- \left(\frac{b}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} \left[\boldsymbol{C}\left(t, \frac{b^2}{s}, K\right) + (L-K)\boldsymbol{H}\left(t, \frac{b^2}{s}, b\right)\right].\end{aligned}$$

Proposition 1.27 (**Put-Call Parity**). The price of a Down-and-Out European Put $\boldsymbol{P}_{bO}$ and the price of a Down-and-Out European Call $\boldsymbol{C}_{bO}$ are related by the formula:

$$\boldsymbol{P}_{bO}(t,s,K) = K\boldsymbol{BO}_{bO}(t,s) - \boldsymbol{ST}_{bO}(t,s) + \boldsymbol{C}_{bO}(t,s,K). \tag{1.23}$$

Now that we have these key results that we can use to price both a Down-and-Out Call and a Down-and-Out Put, we would like to derive the analogous results for the Up-and-Out contracts. It is not surprising that these results can be obtained by following the same ideas, reversing the role of some instruments during the discussion. For example, if we consider the digital Put Option with payoff:

$$\tilde{H}(x,L) = \begin{cases} 0, & x > L \\ 1, & x \leq L \end{cases},$$

with pricing function:

$$\tilde{\boldsymbol{H}}(t,s,L) = e^{-r(T-t)} N\left[\frac{\ln\left(\frac{S_0}{K}\right) + \tilde{r}(T-t)}{\sigma\sqrt{T-t}}\right],$$

we have the following results:

Proposition 1.28 (**Up-and-Out Zero Coupon Bond**). The Up-and-Out zero coupon bond with barrier B, for $s < B$, is priced as:

$$\boldsymbol{BO}^{BO}(t,s) = \tilde{\boldsymbol{H}}(t,s,B) - \left(\frac{B}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} \tilde{\boldsymbol{H}}\left(t, \frac{B^2}{s}, B\right).$$

Proposition 1.29 (**Up-and-Out Stock**). The Up-and-Out contract on the stock with barrier B, for $s < B$, is priced as:

$$\begin{aligned}\boldsymbol{ST}_{BO}(t,s) = B&\left(\tilde{\boldsymbol{H}}(t,s,B) - \left(\frac{B}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} \tilde{\boldsymbol{H}}\left(t, \frac{B^2}{s}, B\right)\right)\\ &- \boldsymbol{P}(t,s,B) + \left(\frac{B}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} \boldsymbol{P}\left(t, \frac{B^2}{s}, B\right).\end{aligned}$$

Proposition 1.30 (**Up-and-Out Put**). The Up-and-Out European Put Option is priced as:

- If $L > K$ and $s < L$:

$$\boldsymbol{P}^{BO}(t,s,K) = \boldsymbol{P}(t,s,K) - \left(\frac{B}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} \boldsymbol{P}\left(t, \frac{B^2}{s}, K\right).$$

- If $L < K$ and $s < L$:

$$\begin{aligned}\boldsymbol{P}^{BO}(t,s,K) =&\boldsymbol{P}(t,s,B) - (K-B)\boldsymbol{H}(t,s,B)\\ &- \left(\frac{B}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} \left[\boldsymbol{P}\left(t, \frac{B^2}{s}, B\right) - (K-B)\boldsymbol{H}\left(t, \frac{B^2}{s}, B\right)\right]\\ &+ \left[1 - \left(\frac{B}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}}\right](K-B)e^{-r(T-t)}.\end{aligned}$$

To conclude this discussion, we just need the Put-Call Parity for this Option, which is given by:

Proposition 1.31 (**Put-Call Parity**). The price of an Up-and-Out European Put $\boldsymbol{P}^{BO}$ and the price of an Up-and-Out European Call $\boldsymbol{C}^{BO}$ are related by the formula:

$$\boldsymbol{P}^{BO}(t,s,K) = K\boldsymbol{BO}^{BO}(t,s) - \boldsymbol{ST}^{BO}(t,s) + \boldsymbol{C}^{BO}(t,s,K). \tag{1.24}$$

This last result concludes the discussion on Out contracts, at least for Call and Put Options.

Turning our attention to the **In** contracts, we can achieve similar pricing formulas simply by applying the In-Out Parity of Observation 1.18 and Formula 1.20.

Proposition 1.32 (**Down-and-In Call**). The Down-and-In European Call Option is priced as:

- If $L < K$:

$$\boldsymbol{C}_{bI}(t,s,K) = \left(\frac{b}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} \boldsymbol{C}\left(t, \frac{b^2}{s}, K\right).$$

- If $L > K$:

$$\begin{aligned}\boldsymbol{C}_{bI}(t,s,K) = &\left(\frac{b}{s}\right)^{\frac{2\tilde{r}}{\sigma^2}} \left[\boldsymbol{C}\left(t, \frac{b^2}{s}, K\right) + (b-K)\boldsymbol{H}\left(t, \frac{b^2}{s}, b\right)\right]\\ &- (b-K)\boldsymbol{H}(t,s,b).\end{aligned}$$

Now let's consider an Option with two Out barriers (*double knockout Option*) b, B (the cases $b = 0$ and $B = \infty$ can be treated similarly) on the asset S_t. To derive a PDE, we consider the stopping time $\tau = \inf\{t \in [0,T] : S_t \notin (b,B)\}$, which represents the first exit moment for S_t from the interval (b,B); after this moment, it is certain that the Option will pay 0 because the barrier event has occurred.

Instead of S_t, we will consider the absorbed process $S_{t\wedge\tau}$, which is also a Markov process, and its discounted version $e^{-r(T-t\wedge\tau)}S_{t\wedge\tau}$ is a martingale. It is easy to notice that the price of this Option will be a function of $t\wedge\tau$ and of the absorbed process. We have that its pricing function will be:

$$F(t\wedge\tau, S_{t\wedge\tau}) = \begin{cases} F(t\wedge\tau, S_{t\wedge\tau}) & \Omega = \text{on } [0,T]\times(b,B) \\ 0 & \text{otherwise} \end{cases}.$$

It satisfies the Black-Scholes equation where it is non-zero because, in that region, the barriers do not play any role, but it has two additional boundary conditions that force the price to be equal to 0 on the boundaries [1]. We achieve the Boundary Value Problem (BVP):

$$\begin{cases} \dfrac{\partial F}{\partial t} + rS\dfrac{\partial F}{\partial S} + \dfrac{\sigma^2 S^2}{2}\dfrac{\partial^2 F}{\partial S^2} = 0 & \text{on } \Omega = [0,T]\times(b,B) \\ F(T,S) = \Phi(S) & \text{final condition} \\ F(t,b) = F(t,B) = 0 & \text{boundary condition} \end{cases}. \tag{1.25}$$

A generalization of this BVP is possible if we consider an interest rate and a volatility that are not constant, in the same way that it is possible to generalize the Black-Scholes PDE without the boundary conditions.

It is also possible to easily generalize this BVP for Basket Options with barriers using the PDE 1.11 and imposing the value on all the boundaries.

1.3.2 Lookback Options

A European Lookback Option is a contract with maturity T that allows the owner to earn a payoff computed as a function of the realized maximum or minimum of the underlying stock price over the entire life of the contract. Typical examples are:

$$\begin{aligned} \text{Lookback Call} \quad & \Phi = S(T) - \min_{t\le T} S(t), \\ \text{Lookback Put} \quad & \Phi = \max_{t\le T} S(t) - S(T), \\ \text{Forward Lookback Call} \quad & \Phi = \big(\max_{t\le T} S(t) - K\big)^+, \\ \text{Forward Lookback Put} \quad & \Phi = \big(K - \min_{t\le T} S(t)\big)^+. \end{aligned}$$

We focus here on the Lookback Put, but it is possible to derive closed formulas for pricing the other three options as well. Other standard results can be found in Conze and Viswanathan (1991) [11] and Goldman et al. (1979) [16]. As with Barrier Options, these results can be proven using the running maximum and running minimum processes, which have a known distribution.

Proposition 1.33 (Lookback Put Price). The price of the Lookback Put at $t = 0$ is given by:

$$F(s,0) = -sN[-d] + se^{-rT}N\left[-d + \sigma\sqrt{T}\right] + s\frac{\sigma^2}{2r}N[d] - se^{-rT}\frac{\sigma^2}{2r}N\left[-d + \sigma\sqrt{T}\right],$$

where:

$$d = \frac{rT + \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}},$$

and N is the Normal distribution.

If we consider an option on the running maximum $M_{S_t}(t) = \sup_{s\in[0,t]} S_s$ (like the Lookback Put) with the usual Geometric Brownian Motion S_t, it is true that (S_t, M_t) is a Markov process. One can prove that the price of these Lookback Options at time t is a function $F(t, S_t, M_t)$ on the domain $\Omega = \{(t, S, M) \in \mathbb{R}^3 : t \in [0, T] \text{ and } 0 \leq S \leq M\}$ and satisfies:

$$\begin{cases} \dfrac{\partial F}{\partial t} + \dfrac{\sigma^2 S^2}{2}\dfrac{\partial^2 F}{\partial S^2} + rS\dfrac{\partial F}{\partial S} - rF = 0, \\ F(T, S, M) = \Phi(S, M), \\ \dfrac{\partial F}{\partial M}(t, S, S) = 0. \end{cases}.$$

This three-dimensional problem (one dimension in time and two in space) can be reduced to a two-dimensional problem (one in time and one in space) if the payoff is of the form $\Phi(S, M) = M\tilde{\Phi}(S/M)$ (as in the Lookback Put). Indeed, if we consider the change of variables $F(t, S, M) = M\omega(t, S/M)$, where $\omega(t, \xi)$ is a function on the new domain $\tilde{\Omega} = \{(t, \xi) : t \in [0, T], \xi \in [0, 1]\}$, it satisfies:

$$\begin{cases} \dfrac{\partial \omega}{\partial t} + \dfrac{\sigma^2 \xi^2}{2}\dfrac{\partial^2 \omega}{\partial \xi^2} + r\xi\dfrac{\partial \omega}{\partial \xi} + r\omega = 0, \\ \omega(T, \xi) = \tilde{\Phi}(\xi), \\ \dfrac{\partial \omega}{\partial \xi}(t, 1) = \omega(t, 1). \end{cases}. \tag{1.26}$$

Clearly, this reduction is not generally possible when (t, S, M) depends on the interest rate and the volatility. In that case, it will be necessary to solve the initial PDE.

It is also possible to deduce a PDE for options depending on the running minimum $m_x(t)$ (like the Lookback Call). The pricing function $F(t, S, m)$ on the domain $\Omega = \{(t, S, m) \in \mathbb{R}^3 : t \in [0, T] \text{ and } m \leq S \leq \infty\}$ satisfies:

$$\begin{cases} \dfrac{\partial F}{\partial t} + \dfrac{\sigma^2 S^2}{2}\dfrac{\partial^2 F}{\partial S^2} + rS\dfrac{\partial F}{\partial S} - rF = 0, \\ F(T, S, m) = \Phi(S, m), \\ \dfrac{\partial F}{\partial m}(t, S, S) = 0. \end{cases}.$$

We can apply a similar change of variables $F(t, S, m) = m\omega(t, S/m)$ when $\Phi(S, m) = m\tilde{\Phi}(S/m)$, with $\omega(t, \xi)$ a function on the new domain $\tilde{\Omega} = \{(t, \xi) :$

$t \in [0,T], \xi \in [1,\infty)\}$, which satisfies:

$$\begin{cases} \dfrac{\partial \omega}{\partial t} + \dfrac{\sigma^2 \xi^2}{2}\dfrac{\partial^2 \omega}{\partial \xi^2} + r\xi \dfrac{\partial \omega}{\partial \xi} + r\omega = 0, \\ \omega(T,\xi) = \tilde{\Phi}(\xi), . \\ \dfrac{\partial \omega}{\partial \xi}(t,1) = \omega(t,1). \end{cases} \tag{1.27}$$

The same considerations as before regarding the generalization of these results hold true.

1.3.3 Asian Options

A European Asian Option is a contract with maturity T that allows the owner to earn a payoff based on the average price of the risky asset over a specified period. The payoff can be written as $\Phi(S_T, A_T)$, where A_T is an average of S_t. There are different types of averages commonly used in the market:

- *Discrete Monitoring*: This is the most commonly used method, where the monitoring occurs at the times $0 = t_0, t_1, \ldots, t_n = T$:

$$A(0,T) = \frac{1}{n}\sum_{i=0}^{n} S(t_i);$$

- *Integral Average*: We will use this for our purposes as it is the continuous counterpart of discrete monitoring:

$$A(0,T) = \frac{1}{T}\int_0^T S_r \, dr;$$

- *Geometric Average*:

$$A(0,T) = \left(\prod_{i=0}^{n} S(t_i)\right)^{\frac{1}{n}},$$

  from a mathematical point of view, this average is preferable because the geometric average of log-normal distributions is still a log-normal distribution, making it possible to derive a pricing formula for the Asian Option using it;

- *Continuous Geometric Average*: This is the continuous version of the geometric average:

$$A(0,T) = \exp\left(\frac{1}{T}\int_0^T \ln(S(t))\, dt\right);$$

If we assume that S_t is a Geometric Brownian Motion, then we can prove that (S_t, A_t) is a Markov process.

The most famous examples of Asian Options belong to two families: fixed strike and floating strike. For example:

$$\begin{aligned}
\text{Fixed Strike Asian Call} \quad & \Phi = (A(0,T) - K)^+, \\
\text{Fixed Strike Asian Put} \quad & \Phi = (K - A(0,T))^+, \\
\text{Floating Strike Asian Call} \quad & \Phi = (S(T) - kA(0,T))^+, \\
\text{Floating Strike Asian Put} \quad & \Phi = (kA(0,T) - S(T))^+.
\end{aligned}$$

The weight k is usually 1, so it is often omitted.

Since (S_t, A_t) is a Markov process, it is possible to show that the price of the Option at time t, denoted by $F(t, S_t, A_t)$, on the domain $\Omega = \{(t,S,A) \in [0,T] \times [0,\infty)^2\}$, satisfies:

$$\begin{cases} \dfrac{\partial F}{\partial t} + \dfrac{\sigma^2 S^2}{2}\dfrac{\partial^2 F}{\partial S^2} + rS\dfrac{\partial F}{\partial S} + \dfrac{1}{t}(S-A)\dfrac{\partial F}{\partial A} - rF = 0, \\ F(T,S,A) = \Phi(S,A). \end{cases} \tag{1.28}$$

As discussed in Rogers and Shi's paper (1995) [29], in some cases, it is possible to reduce this PDE to a two-dimensional problem (one in time and one in space). For example, if we consider a fixed strike Call with $\Phi(S,A) = (A-K)^+$, using the change of variable $F(t,S,A) = S\omega(t, \frac{K-tA/T}{S})$, we can prove that ω satisfies:

$$\begin{cases} \dfrac{\partial \omega}{\partial t} + \dfrac{\sigma^2 \xi^2}{2}\dfrac{\partial^2 \omega}{\partial \xi^2} - \left(\dfrac{1}{T} + r\xi\right)\dfrac{\partial \omega}{\partial \xi} = 0, \\ \omega(T,\xi) = \tilde{\Phi}(\xi), \end{cases} \tag{1.29}$$

where $\tilde{\Phi}(\xi) = (\xi - K)^+$. The same result holds true for a floating strike Put with $\Phi(S,A) = (K-A)^+$ using the same change of variables and $\tilde{\Phi}(\xi) = (\xi)^+$.

If we focus our attention on the floating strike Call with $\Phi(S,A) = (S-A)^+$ (and similarly for the floating strike Put $\Phi(S,A) = (A-S)^+$), using the change of variable $F(t,S,A) = S\omega(t, -\frac{tA}{Ts})$ and $\tilde{\Phi}(\xi) = (1+\xi)^+$ (resp. $\tilde{\Phi}(\xi) = (1+\xi)^-$), we obtain the same PDE 1.29. However, it is important to note that these reductions are not possible for general interest rates and volatilities that depend on (t,S,A).

As mentioned earlier, for geometric average Asian Options, it is possible to derive closed-form formulas since we know the distribution of the average. In Kemna and Vorst (1990) [23], more formulas are provided in detail. Here, we present one example:

Proposition 1.34 (Geometric Average Pricing Formula for Fixed Strike Call). In the case of a Geometric Average Asian Call Option with a fixed strike, the pricing formula is:

$$e^{-rT}\left(S_0 e^{\mu_n T} N\left[d_n\right] - KN\left[d_n - \sigma_n\sqrt{T}\right]\right),$$

where N is the standard normal distribution and:

$$\mu_n = \left(r - \frac{1}{2}\sigma^2\right)\frac{n+1}{2n} + \frac{1}{2}\sigma_n^2,$$

$$\sigma_n^2 = \frac{\sigma^2(n+1)(2n+1)}{6n^2},$$
$$d_n = \frac{\ln\left(\frac{S_0}{K}\right) + \left(\mu_n + \frac{1}{2}\sigma_n^2\right)T}{\sigma_n\sqrt{T}}.$$

1.3.4 American Options

So far, we have focused on European Options. We now turn our attention to American Options and study their pricing problem. An American Option is a contract that can be exercised at any moment before its maturity T. Unlike a European Option, where the payoff Φ is a function $\Phi(S_T)$ achievable only at T, the payoff of an American Option is a function $\Phi(t, S_t)$ that evolves over time, since it can be exercised at any moment. As a result, an American Option provides more rights to its owner than the equivalent European Option, and thus the price of the former will be greater than or equal to the latter.

If we assume a risk-neutral probability space, the price F of an American Option will be the maximum of all possible expectations of gain:

$$F(t,x) = \sup_{\tau \in \mathcal{T}_{[t,T]}} \mathbb{E}\left(e^{-\int_t^\tau r\,ds}\Phi(\tau, S_\tau^{t,x})\right),$$

where $\mathcal{T}_{[t,T]}$ denotes the set of stopping times τ of the filtration $\mathcal{F}_t$ with values in $[t,T]$.

It is possible to prove (see Bensoussan and Lions (1984) [4] and Jaillet, Lamberton, and Lapeyre (1990) [20]) that F is the solution of the PDE:

$$\begin{cases} \min\left(-\dfrac{\partial F}{\partial t} - rS\dfrac{\partial F}{\partial S} - \dfrac{\sigma^2 S^2}{2}\dfrac{\partial^2 F}{\partial S^2} + rF;\, F - \Phi\right) = 0 \quad t \in [0,T],\, S \geq 0, \\ F(T, S_T) = \Phi(S_T). \end{cases}$$

This is equivalent to the variational inequality:

$$\begin{cases} \dfrac{\partial F}{\partial t} + rS\dfrac{\partial F}{\partial S} + \dfrac{\sigma^2 S^2}{2}\dfrac{\partial^2 F}{\partial S^2} - rF \leq 0 \quad t \in [0,T],\, S \geq 0, \\ F \geq \Phi(t, S_t) \quad t \in [0,T],\, S \geq 0, \\ \left(\dfrac{\partial F}{\partial t} + rS\dfrac{\partial F}{\partial S} + \dfrac{\sigma^2 S^2}{2}\dfrac{\partial^2 F}{\partial S^2} - rF\right)(F - \Phi) = 0 \quad t \in [0,T],\, S \geq 0, \\ F(0, S_0) = \Phi(0, S_0) \quad S \geq 0. \end{cases}$$

When we compare the prices of European Calls and Puts with their American counterparts, some important and counterintuitive results emerge. It can be proved that the price of the two Calls will always be the same since the maximum value of the expected payoff is always at the last available moment T. Therefore, exercising the American Call before maturity is not advantageous. On the other hand, for the Put Option, there exists a region where the price of the American Option coincides with the payoff at that moment $\Phi(t, S_t) = (K - S_t)^+$. In this region, the Option should be exercised since it is worth selling the underlying asset to buy a risk-free asset. This region, where $F(t, S_t) = \Phi(t, S_t)$, is called the **region of exercise**.

We will not focus on this type of Option since it is not the main topic of this discussion. Curious readers can find more details in the summary by Pascucci

(2009) [25], which references the works of Bensoussan (1984) [3] and Karatzas (1988, 1989) [21] [22] for a probabilistic approach based on the notion of the Snell envelope in continuous time, Jaillet, Lamberton, and Lapeyre (1990) [20] who employed variational techniques and provided a rigorous justification of the numerical solution, Oksendal and Reikvam [28], Gatarek and Świech [15] who employed the theory of viscosity solutions, Zhang [31] for a complete proof of strong convergence of the schemes and an extension to models with jumps, and of course, the book by Achdou and Pironneau (2005) [2], which devotes an entire chapter to this topic. A final reference is the article by Brennan and Schwartz (1977) [7].

Chapter 2

Numerical Methods: Finite Differences and FEM

2.1 Methods for Systems of ODEs

In the following sections, we will see that solving a PDE often involves converting the problem into solving a specific system of ODEs, defined using matrices that depend on the PDE. Therefore, we will need to solve such systems numerically, a problem that has been widely discussed (see, for example, [9] [17]).

Suppose now that we want to solve an ODE (both in the scalar and vectorial cases) of the form:

$$\begin{cases} y(t) = f(t, y(t)) \\ y(0) = y_0 \end{cases}$$

If we cannot compute an analytical solution, we can use numerical methods to estimate the function $y(t)$. Here we present some methods among the countless ones that we implemented in our code. In the following subsections, we will always consider h as the discretization step size.

Example 2.1 (*Forward Euler Method*)**.** The Forward Euler method is the simplest method that can be applied to solve an ODE. The idea of this method is to approximate the function $y(t)$ through its tangent line at any point. This leads to the approximating formula:

$$\begin{cases} y_{n+1} = y_n + hf(t_n, y_n) \\ y_{n=0} = y_0 \end{cases}$$

This is an explicit method of order 1, and it is also known for not being A-stable. We will see later that this unstable behavior will be observable in our implementation. Since we aim to avoid such instabilities, we will primarily use implicit methods, which are generally more stable than explicit ones.

Example 2.2 (*Backward Euler Method*)**.** The Backward Euler method is an order 1 method and is the simplest implicit method. It uses the same idea as the Forward Euler method, which is to approximate the function $y(t)$ with the tangent line, but instead of considering the tangent at the point (t_n, y_n), it uses

the point (t_{n+1}, y_{n+1}). This leads to the approximating formula:

$$\begin{cases} y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \\ y_{n=0} = y_0 \end{cases}$$

This is clearly an implicit formula because both terms depend on y_{n+1}. In this case, it is necessary to solve a linear system, a usually expensive task that generally ensures more stable behavior of the solver.

Example 2.3 (*Crank-Nicolson Method*)**.** The Crank-Nicolson method is an implicit method of order 2 that uses the approximating formula:

$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right) \\ y_{n=0} = y_0 \end{cases}$$

The Runge-Kutta methods are a family of methods where the approximation between two consecutive points (t_n, y_n) and (t_{n+1}, y_{n+1}) is computed using several stages $\xi_1, \xi_2, \ldots, \xi_n$ that act as intermediate points between them. The general formula for a Runge-Kutta method is:

$$\begin{cases} y(t_n) \approx y_n = \xi_1 \quad (\Rightarrow c_1 = 0)\,, \\ y\left(t_n + hc_2\right) \approx y_n + hc_2 f\left(t_n, y_n\right) = y_n + a_{2,1} hf\left(t_n, \xi_1\right) = \xi_2, \\ \vdots \\ y\left(t_n + hc_i\right) \approx y_n + h\displaystyle\sum_{j=1}^{i-1} a_{i,j} f\left(t_n + hc_j, \xi_j\right) = \xi_i, \\ \vdots \\ y\left(t_n + hc_\nu\right) \approx y_n + h\displaystyle\sum_{j=1}^{\nu-1} a_{\nu,j} f\left(t_n + hc_j, \xi_j\right) = \xi_\nu, \\ y_{n+1} = y_n + h\displaystyle\sum_{j=1}^{\nu} b_j f\left(t_n + hc_j, \xi_j\right). \end{cases}$$

The parameters $a_{i,j}, b_i, c_i$ are defined in a tableau specific to each method in the family. Usually, a tableau has the shape:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

Among this family, it is possible to distinguish different models: explicit, semi-implicit, and implicit, depending on where the non-zero elements $a_{i,j}$ are located in the tableau. Moreover, another important feature of this family is that methods with various orders of convergence are known, and it is possible to obtain embedded methods that control the computed error with only a little extra effort compared to estimating the solution.

Example 2.4 (*Explicit Embedded Runge-Kutta 2*)**.** Let's consider the method with the following tableau:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 1 & 0 \\ & 0 & 1 \end{array}$$

This is known as an explicit method of order 2 called the Heun method, embedded with another approximation also computed using a Runge-Kutta method to estimate the error during the resolution and adapt the step size to match a specified tolerance given as input.

Example 2.5 (*Implicit Runge-Kutta Gauss-Legendre 3*)**.** A method of order 3 is defined by the following tableau:

$$\begin{array}{c|cc} \frac{3+\sqrt{3}}{6} & \frac{3+\sqrt{3}}{6} & 0 \\ \frac{3-\sqrt{3}}{6} & -\frac{\sqrt{3}}{3} & \frac{3+\sqrt{3}}{6} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

This method is known as the Runge-Kutta Gauss-Legendre method, and it is famous for being stable even on stiff problems.

Example 2.6 (*Embedded Runge Kutta Lobatto 3C and 3B*)**.** These two methods are defined respectively by the following tableaux:

$$\begin{array}{c|ccc} 0 & \frac{1}{6} & -\frac{1}{3} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{6} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ & -\frac{1}{2} & 2 & -\frac{1}{2} \end{array}$$

and

$$\begin{array}{c|ccc} 0 & \frac{1}{6} & -\frac{1}{6} & 0 \\ \frac{1}{2} & \frac{1}{6} & \frac{1}{3} & 0 \\ 1 & \frac{1}{6} & \frac{5}{6} & 0 \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ & -\frac{1}{2} & 2 & -\frac{1}{2} \end{array}$$

These are known to be implicit embedded methods, both with an order of convergence of 2. They are known for their stability and the ability to control the computed error at each step.

Looking at the structure of the two tableaux, it is possible to notice an important characteristic that makes the Runge-Kutta Lobatto 3B preferable: its third column in the tableau is full of zeros. This is a remarkable property because it makes the linear systems related to each step much easier to solve without implying (at least in our case) a loss of stability. We will see this fact in the results presented in Chapter 4, for example in Table 4.2 and Table 4.3.

Let's move on to solving PDE problems. Consider the boundary value problem, such as the Cauchy problem expressed by the general Feynman-Kač formula (1.7), also reported here:

$$\begin{cases} \dfrac{\partial F(t,x)}{\partial t} + \mu(t,x)\dfrac{\partial F(t,x)}{\partial x} + \dfrac{1}{2}\sigma^2(t,x)\dfrac{\partial^2 F(t,x)}{\partial x^2} - rF(t,x) = 0 \\ F(T,x) = \Phi(x) \end{cases} . \quad (2.1)$$

There are at least two main ways to solve this numerically: using finite differences and using the finite element method (FEM). The first method is simple to describe in low dimensions but difficult to generalize to higher dimensions, where FEM is often preferred.

2.2 Finite Differences

The finite difference method aims to solve PDEs by first discretizing the spatial variable x and then the time variable t. This approach reduces the problem to a system of ODEs that can be solved using methods such as Euler, Backward-Euler, Crank-Nicolson, Runge-Kutta, etc. To understand the main idea of the finite difference method, let's consider a simple problem: estimating the first and second derivatives of a function $u(x) \in C^3([a,b])$. Consider the discretization given by $x_i = a + (i-1)h$ for $i = 1, \ldots, m$ and $h = (b-a)/(m-1)$. The Taylor series (in the Lagrange form) gives us:

$$u(x_{i+1}) = u(x_i) + hu'(x_i) + \frac{h^2}{2}u''(x_i) + \frac{h^3}{6}u^{(3)}(\hat{x}_i)$$

$$u(x_{i-1}) = u(x_i) - hu'(x_i) + \frac{h^2}{2}u''(x_i) - \frac{h^3}{6}u^{(3)}(\tilde{x}_i)$$

from which

$$u'(x_i) = \Delta u(x_i) - \tau_i^{(1)} = \frac{u(x_{i+1}) - u(x_{i-1})}{2h} - \tau_i^{(1)}$$

where $\tau_i^{(1)} = \frac{h^2}{6}u^{(3)}(\bar{x}_i)$ is known as the **local error** for some $\bar{x}_i$. We have thus obtained the **second-order centered finite difference** formula for the first derivative of u. Similarly, if we consider a function $u(x) \in C^4([a,b])$, we have:

$$u(x_{i+1}) = u(x_i) + hu'(x_i) + \frac{h^2}{2}u''(x_i) + \frac{h^3}{6}u^{(3)}(x_i) + \frac{h^4}{24}u^{(4)}(\hat{x}_i)$$

$$u(x_{i-1}) = u(x_i) - hu'(x_i) + \frac{h^2}{2}u''(x_i) - \frac{h^3}{6}u^{(3)}(x_i) + \frac{h^4}{24}u^{(4)}(\tilde{x}_i)$$

from which

$$u''(x_i) = \Delta^2 u(x_i) - \tau_i^{(2)} = \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2} - \tau_i^{(2)}$$

where $\tau_i^{(2)} = \frac{h^2}{12}u^{(4)}(\bar{x}_i)$, and we have obtained the second-order centered finite difference formula for the second derivative of u.

Using similar techniques, it's possible to derive other finite differences for derivatives of any degree and for any order. Moreover, it's also possible to obtain **uncentered finite differences** (also known as **forward** and **backward** finite differences), which are useful for estimating the derivative at a boundary where a derivative at x_i is estimated using only points to the left or right of x_i. An example of an uncentered finite difference is:

$$u''(x_i) = \frac{2u(x_i) - 5u(x_{i+1}) + 4u(x_{i+2}) - u(x_{i+3})}{h^2}$$

which is the second-order backward finite difference for a function u that is sufficiently smooth.

Observation 2.7. The smoothness of the function u is key for the convergence order of every finite difference. It's important to note that the order of convergence of a finite difference can be proved only if the function is smooth enough to have a well-defined Taylor expansion.

Returning to the discretization of u' on the interval $[a, b]$, ignoring for the moment the conditions at x_1 and x_m, we have the following system of equations:

$$\begin{cases} \Delta u_2 = \dfrac{u(x_1) - u(x_3)}{2h} \\ \quad \vdots \\ \Delta u_i = \dfrac{u(x_{i-1}) - u(x_{i+1})}{2h} \\ \quad \vdots \\ \Delta u_m = \dfrac{u(x_{m-2}) - u(x_m)}{2h} \end{cases}$$

which is a linear system. Considering $\boldsymbol{u} = (u_1, \dots, u_m)^T$ and $\boldsymbol{\Delta u} = (\Delta u_1, \dots, \Delta u_m)$, we can write it as:

$$\boldsymbol{\Delta u} = \frac{1}{2h} \begin{bmatrix} * & * & * & * & * & * \\ -1 & 0 & 1 & 0 & \dots & 0 \\ 0 & -1 & 0 & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -1 & 0 & 1 \\ * & * & * & * & * & * \end{bmatrix} \boldsymbol{u}.$$

We can repeat the procedure for the second-order finite difference, obtaining:

$$\boldsymbol{\Delta^2 u} = \frac{1}{h^2} \begin{bmatrix} * & * & * & * & * & * \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -1 & 2 & -1 \\ * & * & * & * & * & * \end{bmatrix} \boldsymbol{u}.$$

Observation 2.8. These two matrices usually have some remarkable properties. They are almost always sparse, with non-zero elements on the diagonals, and sometimes they are symmetric or skew-symmetric and diagonally dominant.

To fill the first and last rows of these two matrices, we generally need to consider the boundary conditions of our equation, especially since we may have a PDE defined not only with initial/final conditions but also on a specific domain with fixed values on the boundary. In our case, given by the Cauchy problem 2.1, we only have the final condition $F(T, x) = \Phi(x)$ with F defined on the whole real line. Therefore, we don't have any boundary conditions, and we can simply fill the first and last rows with some forward or backward finite differences, preserving the general order of convergence of the matrices (like the example given earlier). We end up with two matrices A, B that we can

use to approximate the first and second derivatives of a generic function with second-order accuracy.

Now we are almost ready to discretize the Feynman-Kač formula with respect to the spatial variable x, with $x_1, \dots, x_m$ on the interval $[a, b]$. We only need to define slightly different versions of the previous matrices since μ and σ are both functions of (t, x). We will consider A', B' given by the element-wise product between $\boldsymbol{\mu} = (\mu(t, x_1), \dots, \mu(t, x_m))$ and A, and between $\boldsymbol{\sigma^2} = (\sigma^2(t, x_1), \dots, \sigma^2(t, x_m))$ and B:

$$A' = [\mu(t, x_i)a_{i,j}]_{i,j}, \quad B' = [\sigma^2(t, x_i)b_{i,j}]_{i,j},$$

Finally, we can discretize the spatial variable x using the vector $\boldsymbol{F}(t) = (F(t, x_1), \dots, F(t, x_m))$, obtaining:

$$\frac{\partial \boldsymbol{F}(t)}{\partial t} + A'\boldsymbol{F}(t) + \frac{1}{2}B'\boldsymbol{F}(t) - r\boldsymbol{F}(t) = 0$$

This is now a system of ODEs where the only remaining variable is t. Letting $\boldsymbol{\Phi} = (\Phi(x_1), \dots, \Phi(x_m))$, we can rewrite it as:

$$\begin{cases} \dfrac{\partial \boldsymbol{F}(t)}{\partial t} = (Ir - A' - \dfrac{1}{2}B')\boldsymbol{F}(t) \\ \boldsymbol{F}(T) = \boldsymbol{\Phi} \end{cases}$$

This is the classical form of a system of ODEs that can be solved using one of the known methods (e.g., Euler method, θ-methods, or the Runge-Kutta family).

Observation 2.9. It's important to remember that using an explicit method to solve the ODE may lead to some unstable behavior, which may require a very small step size to achieve convergence to the solution. In this case, it may be preferable to use implicit methods, which are usually A-stable or more stable than their explicit counterparts, even if the computational effort for each step will increase.

Observation 2.10. If we consider the Black-Scholes equation 1.5 (which is a particular case of the Feynman-Kač formula) for a European Call with strike K and maturity date T, we obtain:

$$\begin{cases} \dfrac{\partial \boldsymbol{F}(t)}{\partial t} = (Ir - A'' - \dfrac{1}{2}B'')\boldsymbol{F}(t) \\ \boldsymbol{F}(T) = (K - s_i)^+ \end{cases}$$

where

$$A'' = [rs_i a_{i,j}]_{i,j}, \quad B'' = [s_i^2\sigma^2(t, s_i)b_{i,j}]_{i,j}$$

This technique, despite being simple to understand and implement in the 1-dimensional case, is difficult to generalize to higher dimensions without breaking some of the properties of the matrices (like those mentioned in Observation 2.8) that can be exploited while solving the system of ODEs. Most importantly, it finds its natural application when we are dealing with a smooth function F (Observation 2.7).

This last condition is generally not satisfied by our function F at maturity (consider, for example, the payoff of a Call Option, which is not differentiable at K). Therefore, in our implementation, we will not be interested in finite differences with high orders of convergence since their gain is lost due to the irregularity of our solution. Instead, we will focus on computing the solution with smaller steps to achieve a better approximation.

2.3 Finite Elements Method (FEM)

Let's move on to a method that is easily generalizable to higher dimensions and requires fewer hypotheses than the finite difference method. Suppose, for example, that we are trying to solve an ill-posed Cauchy problem where the *solution* is given by a function f that has some discontinuities or is not differentiable at some points. In this case, we refer to f as a **weak solution** (in a variational sense) rather than a **strong solution**. We will discuss the differences between these two concepts shortly.

Let's consider now the Black-Scholes equation 1.5 for a European Option with a local volatility (i.e., σ is a function of (t, S_t)) and an interest rate depending on t. For simplicity, we replace the variable t with the time to maturity $T - t$, leading to a forward parabolic equation:

$$\begin{cases} \dfrac{\partial F(t,S)}{\partial t} - rS\dfrac{\partial F(t,S)}{\partial S} - \dfrac{1}{2}S^2\sigma^2(t,S)\dfrac{\partial^2 F(t,S)}{\partial S^2} + rF(t,S) = 0 \\ F(0,S) = \Phi(S) \end{cases} \tag{2.2}$$

Before discussing the concept of a weak solution, we need to introduce some function spaces.

Let $L^2(\mathbb{R}^+)$ be the classical Hilbert space of square-integrable functions on $\mathbb{R}^+$ endowed with the norm $||v||_{L^2(\mathbb{R}^+)} = \left(\int_{\mathbb{R}^+} v(x)^2\,dx\right)^{\frac{1}{2}}$ and the inner product $(v,w)_{L^2(\mathbb{R}^+)} = \int_{\mathbb{R}^+} v(x)w(x)\,dx$. It is a known result that the space $\mathcal{D}(\mathbb{R}^+)$ of smooth functions with compact support in $\mathbb{R}^+$ is dense in $L^2(\mathbb{R}^+)$.

Let us introduce the space

$$W = \left\{ w \text{ continuous on } [0,+\infty) : w(x) = \int_0^x \phi(s)\,ds, \text{ with } \phi \in L^2(\mathbb{R}^+) \right\}$$

which clearly consists of functions that vanish at $x = 0$. The space W, endowed with the norm $||w||_W = ||\frac{dw}{dx}||_{L^2(\mathbb{R}^+)}$, is a Hilbert space that is topologically isomorphic to $L^2(\mathbb{R}^+)$. Thus, W is separable.

It is possible to prove that the space $\mathcal{D}(\mathbb{R}^+)$ is dense in W, and consequently, Hardy's inequality holds.

Proposition 2.11 (**Hardy's inequality**). If $w \in W$, then $\frac{w}{x} \in L^2(\mathbb{R}^+)$ and

$$\left|\left|\frac{w}{x}\right|\right|_{L^2(\mathbb{R}^+)} \leq 2||w||_W \tag{2.3}$$

Proof. Theoretically, it is sufficient to recall that the space $\mathcal{D}(\mathbb{R}^+)$ is dense in W to ensure (2.3). We can also prove this result straightforwardly:

$$\begin{aligned} \left|\left|\frac{w}{x}\right|\right|^2_{L^2(\mathbb{R}^+)} &= \int_{\mathbb{R}^+} \frac{1}{x^2}\left(\int_0^x w'(s)\,ds\right)^2 dx \\ &= \int_{\mathbb{R}^+} \frac{2}{x}w'(x)\left(\int_0^x w'(s)\,ds\right) dx \\ &= \int_{\mathbb{R}^+} \frac{2}{x}w'(x)w(x)\,dx \\ &\leq 2\left|\left|\frac{w}{x}\right|\right|_{L^2(\mathbb{R}^+)} ||w||_W \end{aligned}$$

where we integrated by parts and used the Cauchy-Schwarz inequality to conclude. □

With Hardy's inequality, we can define the space V:

$$V = \left\{ \frac{w}{x} : w \in W \right\} = \left\{ v \in L^2(\mathbb{R}^+) : v = \frac{w}{x} \text{ with } w \in W \right\}$$

where the function $v \in V$ is clearly continuous on $\mathbb{R}^+$ if vx is continuous on $[0, +\infty]$. Thus, V can be rewritten as

$$V = \left\{ v \in L^2(\mathbb{R}^+) : x\frac{dv}{dx} \in L^2(\mathbb{R}^+) \right\}$$

where the derivative must be understood in the sense of the distribution on $\mathbb{R}^+$.

We can now endow V with the inner product $(v, w)_V = (v, w) + (x\frac{dv}{dx}, x\frac{dw}{dx})$ and with the Euclidean norm $||v||_V = \sqrt{(v, v)_V}$ to complete V and obtain a Hilbert space. Moreover, with these choices, we have that the space $\mathcal{D}(\mathbb{R}^+)$ is dense in V.

Proposition 2.12 (**Poincaré's inequality**). If $v \in V$, then

$$||v||_{L^2(\mathbb{R}^+)} \leq 2 \left|\left| x\frac{dv}{dx} \right|\right|_{L^2(\mathbb{R}^+)}$$

Proof. Since $\mathcal{D}(\mathbb{R}^+)$ is dense in V, it is sufficient to prove the inequality for $v \in \mathcal{D}(\mathbb{R}^+)$. We have

$$2\int_{\mathbb{R}^+} xv(x)\frac{dv}{dx}(x)\,dx = -\int_{\mathbb{R}^+} v^2(x)\,dx$$

using integration by parts. Now, applying the Cauchy-Schwarz inequality to the left-hand side, we can conclude that:

$$||v||^2_{L^2(\mathbb{R}^+)} \leq 2||v||_{L^2(\mathbb{R}^+)} \left|\left| x\frac{dv}{dx} \right|\right|_{L^2(\mathbb{R}^+)}$$

which yields the desired estimate. □

With Poincaré's inequality, we can prove that the seminorm $|v|_V = ||x\frac{v}{x}||_{L^2(\mathbb{R}^+)}$ is indeed a norm on V and that it is equivalent to $||\cdot||_V$. Similarly, using the density of $\mathcal{D}(\mathbb{R}^+)$ in $L^2(\mathbb{R}^+)$, one can prove the following result:

Proposition 2.13. If $w \in L^2(\mathbb{R}^+)$, then the function v, defined as $v(x) = \frac{1}{x}\int_0^x w(s)\,ds$, belongs to V, and there exists a positive constant C independent of w such that $||v||_V \leq ||w||_{L^2(\mathbb{R}^+)}$.

We denote by V' the topological dual space of V. We also denote by $(\cdot, \cdot)$ the duality pairing between V and V' and define $||\cdot||_{V'}$ by:

$$||w||_{V'} = \sup_{v \in V \setminus \{0\}} \frac{(w, v)}{|v|_V}.$$

Now that we have introduced some function spaces useful for this discussion, we can return to the Black-Scholes equation (2.2) and introduce what the **weak formulation** of this problem means.

Let us multiply the PDE by a smooth real-valued function $\xi(S)$ on $\mathbb{R}^+$ and integrate in S on $\mathbb{R}^+$. Assuming that integration by parts is permitted, we obtain:

$$\begin{aligned}0 =& \frac{d}{dt}\int_{\mathbb{R}^+} F(t,S)\xi(S)\,dS \\ &+ \int_{\mathbb{R}^+} \frac{S^2\sigma^2(t,S)}{2}\frac{\partial F}{\partial S}(t,S)\frac{\partial \xi}{\partial S}(S)\,dS \\ &+ \int_{\mathbb{R}^+} \left(-r(t) + \sigma^2(t,S) + S\sigma(t,S)\frac{\partial \sigma}{\partial S}(t,S)\right) S\frac{\partial F}{\partial S}(t,S)\xi(S)\,dS \\ &+ r(t)\int_{\mathbb{R}^+} F(t,S)\xi(S)\,dS.\end{aligned}$$

This leads us to introduce the bilinear form a_t:

$$\begin{aligned}0 =& + \int_{\mathbb{R}^+} \frac{S^2\sigma^2(t,S)}{2}\frac{\partial v}{\partial S}\frac{\partial w}{\partial S}\,dS \\ &+ \int_{\mathbb{R}^+} \left(-r(t) + \sigma^2(t,S) + S\sigma(t,S)\frac{\partial \sigma}{\partial S}(t,S)\right) S\frac{\partial v}{\partial S} w\,dS \\ &+ r(t)\int_{\mathbb{R}^+} vw\,dS.\end{aligned} \tag{2.4}$$

To proceed with the discussion, we need some assumptions:

Hypothesis 2.14. We make some assumptions on σ and r: we assume that the coefficients σ and r are non-negative, continuous (this is only for simplicity), and that σ is sufficiently regular so that the following conditions make sense:

- There exist two positive constants, σ_1 and σ_2, such that for all $t \in [0,T]$ and all $S \in \mathbb{R}^+$,
$$0 < \sigma_1 \leq \sigma(t,S) \leq \sigma_2.$$
- There exists a positive constant C_σ such that for all $t \in [0,T]$ and all $S \in \mathbb{R}^+$,
$$\left|S\frac{\partial \sigma}{\partial S}(t,S)\right| \leq C_\sigma.$$

Under these assumptions, we find that the bilinear form a_t is continuous on V, i.e., there exists a positive constant μ such that for all $v, w \in V$:

$$|a_t(v,w)| \leq \mu |v|_V |w|_V.$$

The proof of this result is quite tedious and involves controlling each integral with an estimate.

We recall the standard notation $C^0([0,T];L^2(\mathbb{R}^+))$ as the space of continuous functions on $[0,T]$ with values in $L^2(\mathbb{R}^+)$, and $L^2((0,T);V)$ as the space of square-integrable functions on $(0,T)$ with values in V. Assuming that $\Phi \in L^2(\mathbb{R}^+)$, we can prove the weak formulation for the Black-Scholes PDE [24].

Theorem 2.15 (**Weak Formulation of (2.2)**). Solving the Black-Scholes PDE (2.2) is equivalent (in the weak sense) to finding a function $F \in C^0([0,T]; L^2(\mathbb{R}^+)) \cap L^2((0,T); V)$ such that $\frac{\partial F}{t} \in L^2((0,T); V')$ satisfying:

$$\forall v \in V, \quad \left(\frac{\partial F}{t}(t), v\right) + a_t(F(t), v) = 0, \tag{2.5}$$

$$F(0) = \Phi \quad \text{in } \mathbb{R}^+ \text{ and for a.e. } t \in (0,T). \tag{2.6}$$

For the next discussion, we need the following estimate:

Proposition 2.16 (**Gårding's inequality**). Under the assumptions 2.14, there exists a non-negative constant λ such that for all $v \in V$:

$$a_t(v,v) \geq \frac{\sigma^2}{4}|v|_V^2 - \lambda ||v||^2_{L^2(\mathbb{R}^+)}$$

Theorem 2.17 (**Existence of a Solution for the Weak Formulation**). If $\Phi \in L^2(\mathbb{R}^+)$ and under Hypothesis (2.14), the weak formulation (2.5) has a unique solution.

The proof of this existence is given in [24]. Note that this result applies to any European Option with a payoff function in $L^2(\mathbb{R}^+)$, particularly to vanilla Puts. It does not apply to vanilla Calls, and we will revisit this later.

If the interest rate, the volatility, and the payoff are smooth enough, then it is possible to prove additional regularity for the solution of the Weak Formulation (2.5) by calling A_t the unbounded operator in $L^2(\mathbb{R}^+)$:

$$A_t v = -\frac{\sigma^2(t,S)S^2}{2}\frac{\partial^2 v}{S^2} - r(t)S\frac{\partial v}{S} + r(t)v,$$

it can be checked that for all $t \in [0,T]$ and for λ given by Gårding's inequality 2.16, the domain of $A_t + \lambda$ is:

$$D = \left\{ v \in V : S^2 \frac{\partial^2 v}{S^2} \in L^2(\mathbb{R}^+) \right\}.$$

Hypothesis 2.18. There exist a positive constant C and $0 < \alpha < 1$ such that for all $t_1, t_2 \in [0,T]$ and $S \in \mathbb{R}^+$:

$$|r(t_1) - r(t_2)| + |\sigma(t_1,S) - \sigma(t_2,S)| + \left|\frac{\partial \sigma}{\partial S}(t_1,S) - \frac{\partial \sigma}{\partial S}(t_2,S)\right| \leq C|t_1 - t_2|^\alpha.$$

With the assumptions 2.14 and 2.18, it is possible to prove the so-called smoothing effect: the solution of the weak formulation 2.5 belongs to D at any time $t > 0$, for any Cauchy data $\Phi \in L^2(\mathbb{R}^+)$. More precisely, we have that:

- The domain of A_t is D, which is dense in $L^2(\mathbb{R}^+)$ and independent of t;
- Gårding's inequality holds;
- If $\tilde{A}_t = A_t + \lambda I$, there exists a constant L such that:

$$||(\tilde{A}_t - \tilde{A}_s)\tilde{A}_\tau^{-1}||_{\mathcal{L}(L^2(\mathbb{R}^+))} \leq L|t-s|^\alpha$$

With these three facts, we can apply the general results of Kato on parabolic evolution equations [26], [10] and obtain the following results:

Theorem 2.19. Under Assumptions 2.14 and 2.18, for all $S \in \mathbb{R}^+$, $t \in [0,T]$, the solution F of the Weak Formulation 2.5 satisfies $F \in C^0([t,T];D)$ and $\frac{\partial F}{t} \in C^0([t,T];L^2(\mathbb{R}^+))$, and there exists a constant C such that for all $t \in [0,T]$:

$$||A_t F(t)||_{L^2(\mathbb{R}^+)} \leq \frac{C}{t}.$$

On the other hand, if $\Phi \in D$, then the solution F of 2.5 belongs to $C^0([t,T];D)$ and $\frac{\partial F}{t} \in C^0([t,T];L^2(\mathbb{R}^+))$.

This gives us the regularity result for $\Phi \in V$.

Proposition 2.20. If Assumption 2.14 is satisfied and $\Phi \in V$, then the solution of 2.5 belongs to $C^0([0,T];V) \cap L^2(0,T;D)$, $\frac{\partial F}{t} \in L^2(0,T;L^2(\mathbb{R}^+))$, and there exists a non-negative constant $\tilde{\lambda}$ such that the estimate:

$$e^{-2\tilde{\lambda}t}\|S\frac{\partial F}{S}(t)\|^2_{L^2(\mathbb{R}^+)} + \frac{\sigma^2}{2}\int_0^t e^{-2\tilde{\lambda}\tau}|S\frac{\partial F}{S}(\tau)|^2_V d\tau \leq \|S\frac{\partial \Phi}{S}\|^2_{L^2(\mathbb{R}^+)}$$

Observation 2.21. As a consequence of this last proposition, the solution to the weak formulation is continuous if Assumption 2.14 is satisfied and $\Phi \in V$.

It is easy to notice that these results apply to a European vanilla Put because $\Phi = (K-S)^+ \in V$ but they don't apply to a European vanilla Call.

Now that we have some results that ensure the regularity of the solution given certain assumptions, we want to study other properties of the solution given by the weak formulation. In particular, we want to prove that some results already known in finance are ensured by this formulation. For example, we will see that the positivity of the pricing function is guaranteed, that the Put-Call Parity is ensured, and that every pricing function is controlled by other pricing functions if its payoff is controlled by their payoffs.

To achieve these results, we have to state the Maximum Principle for a class of functions much larger than V and understand its consequences. For a deep dive into the maximum principle, the reader can refer to Protter, Murray, and Weinberger [27]. We define the function space:

$$\mathcal{V} = \left\{ v : \forall\, \varepsilon > 0,\ v(S)e^{-\varepsilon \log^2(S+2)} \in V \right\}$$

which also contains polynomial functions. Using Stampacchia's truncation method, we have the Weak Maximum Principle:

Theorem 2.22 (Weak Maximum Principle). Let $u(t,S)$ be such that for all positive numbers ε:

1. $ue^{-\epsilon \log^2(S+2)} \in C^0([0,T];L^2(\mathbb{R}^+)) \cap L^2(0,T;V)$,
2. $\frac{\partial u}{t}e^{-\epsilon \log^2(S+2)} \in L^2(\mathbb{R}^+ \times (0,T))$,
3. $u|_{t=0} \geq 0$ a.e.,
4. $\frac{\partial u}{t} + A_t u \geq 0$ (in the sense of distributions);

then $u \geq 0$ a.e.

This theorem may be generalized using the previous assumptions and using a function that satisfies the weak formulation 2.5, but for our purposes, this result is sufficient for deriving the consequences we are looking for. That's because the maximum principle is an extremely powerful tool for proving estimates on the solution of elliptic and parabolic PDEs.

Proposition 2.23. Under Assumption 2.14, let F be the weak solution to 2.5, with $\Phi \in L^2(\mathbb{R}^+)$ a bounded function, i.e., $0 \leq \underline{\Phi} \leq \Phi(S) \leq \overline{\Phi}$. Then, a.e.:

$$\underline{\Phi} e^{-\int_0^t r(\tau)d\tau} \leq F(t,S) \leq \overline{\Phi} e^{-\int_0^t r(\tau)d\tau}$$

An example of the application of this proposition for a vanilla Put Option $\Phi(S) = (K-S)^+$ gives the (not surprising but comforting) result $0 \leq F(t,S) \leq Ke^{-\int_0^t r(\tau)d\tau}$.

For vanilla Puts, we also have other results, such as the following:

Proposition 2.24. Under Assumption 2.14, let F be the weak solution to 2.5, with $\Phi(S) = (K-S)^+$, then we have that:

$$(Ke^{-\int_0^t r(\tau)d\tau} - S)^+ \leq F(t,S) \leq Ke^{-\int_0^t r(\tau)d\tau} \tag{2.7}$$

and:

$$P(t,0) = Ke^{-\int_0^t r(\tau)d\tau}.$$

This last result is important because we didn't impose the boundary condition 2.7 a priori. Indeed, it arises naturally both in finite differences and finite elements.

Lemma 2.25 (The Super-Replication Principle). Take two European Put Options with the same maturity and two different payoff functions Φ_1 and Φ_2. Call $F_1(t,S)$ and $F_2(t,S)$ their respective prices, which both satisfy 2.2. One easy consequence of the maximum principle is that if for all S, $\Phi_1(S) \leq \Phi_2(S)$, then for all t and S, $F_1(t,S) \leq F_2(t,S)$. This is called the **super-replication principle** in the finance community.

Lemma 2.26 (The Put-Call Parity). Take a vanilla Put Option and consider the function $C(t,S)$ given by:

$$C(t,S) = S - Ke^{-\int_0^t r(\tau)d\tau} + P(t,S).$$

From the fact that P and $S - Ke^{-\int_0^t r(\tau)d\tau}$ satisfy 2.2, it is clear that C is a solution of the same PDE 2.2 with the Cauchy condition $C(0,S) = (S-K)^+$. This is precisely the boundary value problem for the European vanilla Call Option. On the other hand, from the Maximum Principle, we know that a well-behaved solution (in the sense of the weak maximum principle hypothesis 2.22) to this boundary value problem is unique. Therefore, we can deduce the price of the Call Option from that of the Put Option: we recover the Put-Call parity seen in Chapter 1 and well known to financial players.

If we now consider some results regarding the desired convexity of the pricing function F in the variable S, we need a new assumption.

Hypothesis 2.27. There exists a positive constant C such that:

$$|S^2\frac{\partial^2\sigma}{S^2}(t,S)| \leq C \text{ a.e.}$$

Proposition 2.28. Under Assumptions 2.14 and 2.27, let F be the weak solution to 2.5, where $\Phi \in V$ is a convex function such that $\frac{\partial^2\Phi}{S^2}$ has a compact support. Then, for all $t > 0$, $F(t,S)$ is a convex function of S.

As a consequence of this result, we see that under these assumptions, the price of a vanilla European Put Option is convex with respect to S, and thanks to the Call-Put parity, this is also true for the vanilla European Call.

For vanilla European Puts, we also have that the solution of a weak formulation with local volatility can be controlled by the price of two vanilla Puts with constant volatilities.

Proposition 2.29. Under Assumption 2.14, we have, for all $t \in [0,T]$ and for all $S > 0$:

$$\underline{F}(t,S) \leq F(t,S) \leq \bar{F}(t,S)$$

where $\underline{F}$ (resp., $\bar{F}$) is the solution to 2.5 with $\sigma = \underline{\sigma}$ (resp., $\bar{\sigma}$).

Returning to the numerical computation of the price function F for a vanilla Put, we see that S is defined on the entire half-line $(0,+\infty)$, so we will be forced to restrict the domain to $S \in (0,\bar{S})$ for $\bar{S}$ large enough and impose some boundary condition at $S = \bar{S}$. If we assume that the function vanishes on the artificial boundary, we obtain the new BVP:

$$\begin{cases} \frac{\partial\tilde{F}}{t} - \frac{\sigma^2S^2}{2}\frac{\partial^2\tilde{F}}{S^2} - rS\frac{\partial\tilde{F}}{S} + r\tilde{F} = 0, \quad t \in (0,T], \quad S \in (0,\bar{S}) \\ \tilde{F}(\bar{S},t) = 0, \quad t \in (0,T] \end{cases}$$

with the Cauchy data $\tilde{F}(S,0) = (K-S)^+$ in $(0,\bar{S})$. The theory of weak solutions applies to this new boundary value problem, but one has to work in the new Sobolev space:

$$\tilde{V} = \{v, S\frac{\partial v}{S} \in L^2((0,\bar{S})), v(\bar{S}) = 0\}.$$

The results of previous theorems can be transferred to this new BVP. The question is how to estimate the error between F and $\tilde{F}$. For that, we use a version of the maximum principle adapted to 2.3, the proof of which is omitted for brevity. We ultimately obtain the following result:

Proposition 2.30. Under Assumption 2.14, the error $\max_{t\in[0,T],S\in[0,\bar{S}]} |F(t,S) - \tilde{F}(t,S)|$ decays faster than any exponential $\exp(-\eta\bar{S})(\eta > 0)$ as $\bar{S} \to \infty$.

2.3.1 Variational Formulation: The Galerkin Method

In the previous sections, we introduced the weak formulation of the Black-Scholes equation, which is defined on an infinite-dimensional function space V. To solve it using the Finite Element Method (FEM), we need to approximate the problem by projecting it onto a finite-dimensional subspace V_h of V using the Galerkin method.

The Galerkin method is widely applicable, regardless of the dimensional space. When posed in an infinite-dimensional space V (e.g., $V = H^1(\Omega)$, which is the space of functions with square-integrable derivatives), the method involves selecting a finite-dimensional subspace V_h of V. This subspace might consist, for instance, of continuous piecewise linear functions over a triangulation of the domain Ω.

Construction of the Finite-Dimensional Subspace V_h

- **Domain Discretization**: The domain Ω is partitioned into non-overlapping cells (elements), such as intervals in one dimension, triangles or quadrilaterals in two dimensions, and tetrahedra or hexahedra in three dimensions. This set of elements forms a mesh, which can be structured or unstructured. In practice, this discretization might lead to a computational domain $\bar{\Omega}$ that is an approximation of the real domain Ω (Figure 2.1).
- **Basis Function Construction**: The degree k of the polynomial approximation within each element is chosen. For simplicity, degree $k = 1$ is often used, leading to piecewise linear basis functions. The basis functions $(\phi_j)_{j=1}^h$ are constructed such that each basis function ϕ_j is 1 at node j and 0 at all other nodes, with linear interpolation between nodes (see Figures 2.2 and 2.3).
- **Function Approximation**: A function $v \in V$ can then be approximated by its projection $\hat{v}$ onto V_h as

$$\hat{v} = \sum_{j=1}^{h} \hat{v}_j \phi_j(x).$$

Observation 2.31. The mesh does not need to be regular. In fact, it is generally advantageous to use an adaptive mesh to focus computational effort on regions where the solution is less regular.

It is possible to distinguish a priori and a posteriori error estimates for the computed solution: in a priori estimates, the error is bounded by some quantity depending on the solution of the continuous problem (which is unknown, but for which estimates are available), whereas, in a posteriori estimates, the error is bounded by some quantity depending on the solution of the discrete problem which is available. For a priori error estimates, one can see the books of Strang and Fix [8], Brenner and Scott [8] on parabolic problems. For a posteriori estimate, one can look at Chapter 5 of the book by Achdou & Pironneau [2] for posteriori estimates on the Black-Scholes equation.

Variational Framework

Let us define the framework for the FEM approach:

- Let Ω be a polygonal domain in $\mathbb{R}^2$ (i.e., Ω is open and bounded);
- Let Γ be the boundary of Ω;

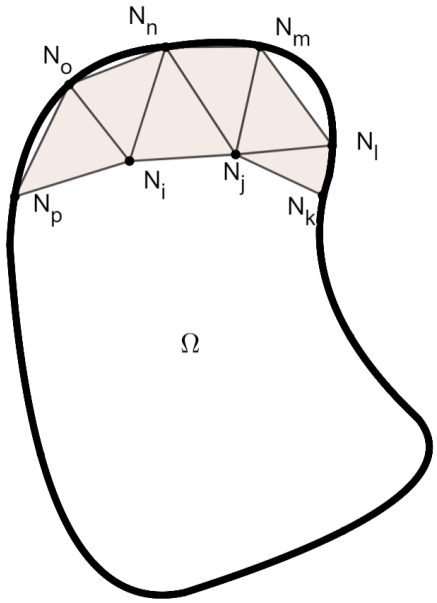
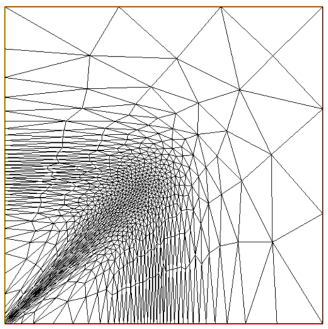
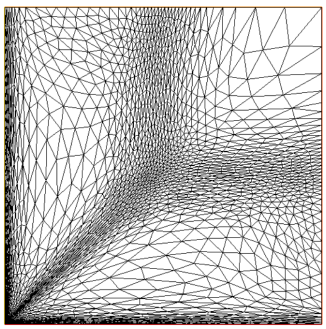


Figure 2.1: Mismatch between the real domain Ω and the computational domain $\bar{\Omega}$.



(a) Mesh for Put Option on the max of two stocks



(b) Mesh for Put Option on the min of two stocks

- We assume that $\Gamma = \Gamma_d \cup \Gamma_n$, where the one-dimensional measure of $\Gamma_d \cap \Gamma_n$ is 0; for $x \in \Gamma$, Γ_d and Γ_n define a partition of Γ except for a null set;
- Denote by n the unit normal vector to Γ at x, pointing outward;
- Consider smooth enough functions:

$$\kappa : \Omega \mapsto \mathbb{R}^{2\times 2}, \quad \alpha : \Omega \mapsto \mathbb{R}^2, \quad \beta : \Omega \mapsto \mathbb{R}, \quad b : \Gamma_n \mapsto \mathbb{R}.$$

For suitable functions $u_0 : \Omega \mapsto \mathbb{R}$, $\phi : \Omega \times (0,T) \mapsto \mathbb{R}$, $g : \Gamma_d \times (0,T] \mapsto \mathbb{R}$, $f : \Gamma_n \times (0,T] \mapsto \mathbb{R}$, we seek $u(x,t)$ solving the parabolic boundary value problem:

$$\begin{cases} \dfrac{\partial u}{\partial t} - \nabla \cdot (\kappa \nabla u) - \nabla \cdot (\alpha u) + \beta u = \psi & \text{in } \Omega \times (0,T), \\ u|_{t=0} = u_0(x) & \text{in } \Omega, \\ u = g & \text{on } \Gamma_d \times (0,T), \\ -bu - (\kappa \nabla u) \cdot n = f & \text{on } \Gamma_n \times (0,T). \end{cases} \tag{2.8}$$

Multiplying both sides by a smooth function $w \in W$ and assuming integration by parts is allowed, we introduce the bilinear form on W:

$$a(w,v) = \int_\Omega ((\kappa \nabla w) \cdot \nabla v - \nabla \cdot (\alpha w) v + \beta w v) + \int_{\Gamma_n} bwv.$$

Assuming that:

- For almost every $x \in \Omega$, $\kappa(x)$ is a symmetric tensor and there exist two positive constants $0 < \underline{\kappa} \leq \bar{\kappa}$ such that, for all $\xi \in \mathbb{R}^2$,

$$\underline{\kappa}|\xi|^2 \leq \kappa(x)\xi \cdot \xi \leq \bar{\kappa}|\xi|^2 \quad \text{for almost every } x \in \Omega;$$

- $\alpha \in (L^\infty(\Omega))^2$, $\beta \in L^\infty(\Omega)$, $b \in L^\infty(\Gamma_n)$;

it can be shown that there exist two positive constants $\underline{c} \leq \bar{c}$ and a non-negative constant λ such that for all $v, w \in W$, we obtain the upper and lower bounds:

$$a(v,w) \leq \bar{c}|v|_W |w|_W$$

and (the Gårding's inequality):

$$a(v,v) \geq \underline{c}|v|_W^2 - \lambda \|v\|_{L^2(\Omega)}^2.$$

The variational formulation of 2.8 can be rephrased as:
Find $u : u - u_g \in L^2((0,T);V)$, $u \in C^0([0,T];L^2(\Omega))$, *and* $\frac{\partial u}{\partial t} \in L^2((0,T);V')$, *with* $u|_{t=0} = u_0$, *and, for almost every* $t \in (0,T)$*:*

$$\forall v \in V, \quad \left(\frac{\partial u}{\partial t}(t), v\right)_V + a(u(t), v) = \int_\Omega \psi(t) v + \int_{\Gamma_n} f(t) v. \tag{2.9}$$

Thanks to the bounds on $a(v,w)$, it can be shown that if u_g satisfying the above condition exists, then the variational formulation has a unique solution, which satisfies the first line of 2.8 in the sense of distributions.

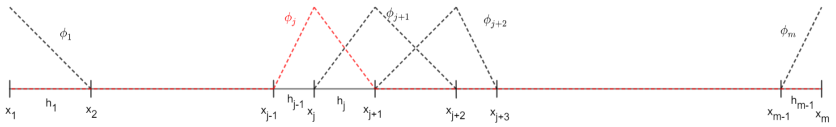


Figure 2.2: Hat function.

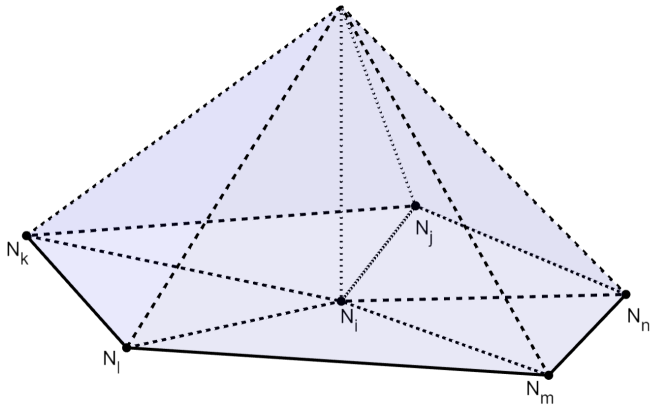


Figure 2.3: Basis function ϕ_i.

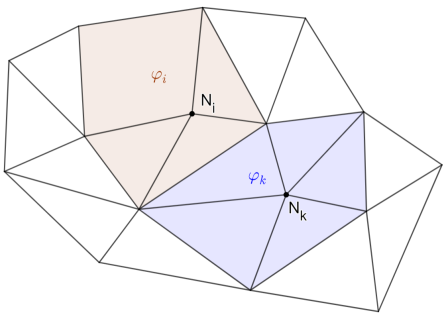


Figure 2.4: Representation of two basis functions ϕ_i and ϕ_k where nodes N_i and N_k are not adjacent. The red and blue areas show where the functions are non-zero.

Computation of the Finite Element Solution

Suppose now that we have the triangulation of our computational domain $\bar{\Omega}$ with n triangles and h nodes. Suppose also that we chose $k = 1$, meaning that the functions used to approximate V have a maximum degree of 1. The simplest basis functions of the space V_h are the set of functions $(\phi_j)_{j=1}^h$, where each ϕ_j is equal to 1 at node j and zero at all the remaining nodes, with linear interpolation between nodes. Figures 2.2 and 2.3 show representations of these functions in 1D and 2D cases, respectively.

In this space V_h, the orthogonal decomposition $\hat{v}$ of v is:

$$\hat{v} = \sum_{j=1}^{h} \hat{v}_j \phi_j(x).$$

This decomposition allows us to easily obtain the value of v at node j as $\hat{v}_j$ and simplifies our calculations by recognizing that the basis functions are non-zero only in a limited number of elements. Indeed, when we consider the variational formulation 2.9 on the function $\hat{u} \in V_h$, it suffices to verify the equation for the basis functions, i.e.:

$$\forall \phi_i \in (\phi_i)_i, \quad \left(\frac{\partial \hat{u}}{\partial t}(t), \phi_i\right)_V + a(\hat{u}(t), \phi_i) = \int_\Omega \psi(t)\phi_i + \int_{\Gamma_n} f(t)\phi_i. \tag{2.10}$$

Taking the orthogonal decomposition $\hat{u} = \sum_i \hat{u}_i \phi_i$ and recalling the linearity of the bilinear form a and the linearity of the integral operator $(\cdot, \cdot)_V$, we get:

$$\sum_{j=1}^{h} \hat{u}_j \left(\frac{\partial \phi_j}{\partial t}, \phi_i\right)_V + \sum_{j=1}^{h} \hat{u}_j a(\phi_j(t), \phi_i) = \int_\Omega \psi(t)\phi_i + \int_{\Gamma_n} f(t)\phi_i.$$

This set of equations can be manipulated to recognize it as the linear system $A\hat{u} = b$, where:

$$A_{i,j} = \left(\frac{\partial \phi_j}{\partial t}, \phi_i\right)_V + a(\phi_j(t), \phi_i) \quad \text{and} \quad b_i = \int_\Omega \psi(t)\phi_i + \int_{\Gamma_n} f(t)\phi_i.$$

Thanks to the basis functions used, most of the products between two basis functions are zero. Indeed, this product is non-zero only if the nodes i and j of ϕ_i and ϕ_j are adjacent (Figure 2.4). Therefore, the matrix A is sparse and can be computed using a local approach: building the matrix A element by element is simpler than doing it node by node. For example, if we have an element e with vertices N_1, N_{10}, N_{23}, we obtain:

$$A^{(e)} = \begin{bmatrix} a_{1,1}^{(e)} & a_{1,10}^{(e)} & a_{1,23}^{(e)} \\ a_{10,1}^{(e)} & a_{10,10}^{(e)} & a_{10,23}^{(e)} \\ a_{23,1}^{(e)} & a_{23,10}^{(e)} & a_{23,23}^{(e)} \end{bmatrix}$$

which we sum to the related entries of A:

$$A = \begin{bmatrix} a_{1,1} & \dots & a_{1,10} & \dots & a_{1,23} & \dots \\ \vdots & \ddots & \vdots & \ddots & \vdots & \dots \\ a_{10,1} & \dots & a_{10,10} & \dots & a_{10,23} & \dots \\ \vdots & \ddots & \vdots & \ddots & \vdots & \dots \\ a_{23,1} & \dots & a_{23,10} & \dots & a_{23,23} & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}.$$

(Of course, we can sum above the main diagonal and use the symmetry of A to fill in the values below.)

We are now interested in computing the local products for the local matrix, which depends on the specific problem being solved.

If we focus on the classical Black-Scholes equation, using the bilinear form defined in 2.4, we can easily compute the matrix A and have everything needed to solve our problem numerically.

2.3.2 Barrier Option

Let's consider the case of a knockout boundary option on a single asset, where the boundary does not depend on time. In this scenario, the option becomes worthless if the price of the underlying asset exits the interval $[b, B]$. Consequently, the price of the option satisfies the usual Black-Scholes equation with a new boundary condition on the barriers:

$$F(t, b) = F(t, B) = 0.$$

The domain Ω is then restricted to the interval $[b, B]$, and the previous arguments still apply by using the space:

$$V = \left\{ v \in L^2(\Omega) : \frac{\partial v}{\partial S} \in L^2(\Omega); v(b) = v(B) = 0 \right\}$$

when there is a lower constraint, and:

$$V = \left\{ v \in L^2(\Omega); S\frac{\partial v}{\partial S} \in L^2(\Omega); v(B) = 0 \right\}$$

when there isn't a lower constraint.

The same argument holds true when we consider basket options. For example, if we have two assets S_1 and S_2 with barriers b_1, B_1 and b_2, B_2, respectively, the new domain becomes $\Omega = [b_1, B_1] \times [b_2, B_2]$ with the boundary condition:

$$F(t, s_1, s_2) = 0 \quad \text{on } \partial\Omega.$$

2.3.3 Basket Option

Let's consider, for simplicity, the case of an option on two assets. We assume, as before, that these assets follow the dynamics described in equation 1.10. In this case, we define the function space as:

$$V = \left\{ v : v, S_1\frac{\partial v}{\partial S_1}, S_2\frac{\partial v}{\partial S_2} \in L^2(\mathbb{R}^2_+) \right\}.$$

Among the payoffs mentioned in Section 1.2, we observe that the payoffs given by 1.12 and 1.13 belong to $L^2(\mathbb{R}^2_+)$, while the payoff given by 1.14 does not. For the following discussion, we assume that the payoff belongs to $L^2(\mathbb{R}^2_+)$.

We introduce the bilinear form:

$$\begin{aligned} a(v,w) = &\int_{\mathbb{R}^2_+} \frac{1}{2} \sum_{k,l=1}^{2} M_{k,l} S_k S_l \frac{\partial v}{\partial S_k} \frac{\partial w}{\partial S_l}\, dS \\ &+ \int_{\mathbb{R}^2_+} \sum_{k=1}^{2} \left(-r + \sum_{l=1}^{2} \frac{1}{2} M_{k,l} \right) S_k \frac{\partial v}{\partial S_k} w\, dS \\ &+ r \int_{\mathbb{R}^2_+} vw\, dS. \end{aligned}$$

Then, the weak formulation of equation 1.11 consists of finding a function F such that:

$$\begin{aligned} F \in C^0\left([0,T]; L^2(\mathbb{R}^2_+)\right) \cap L^2(0,T;V), \quad &\text{such that} \\ \frac{\partial F}{\partial t} \in L^2(0,T;V'), \quad &\text{and} \\ F|_{t=0} = \Phi \quad &\text{in } \mathbb{R}^2_+, \end{aligned} \tag{2.11}$$

and for almost every $t \in (0,T)$:

$$\forall v \in V, \quad \left(\frac{\partial F}{\partial t}(t), v \right) + a(F(t), v) = 0. \tag{2.12}$$

Similarly to the previous cases, it is possible to prove that if $\Phi \in L^2(\mathbb{R}^2_+)$, then the problem defined by equations 2.11 and 2.12 has a unique solution F. Moreover, we have the following estimate, for all $t \in (0,T)$:

$$e^{-2\lambda t} \|F(t)\|^2_{L^2(\mathbb{R}^2_+)} + 2\underline{\sigma}^2 \int_0^t e^{-2\lambda\tau} |F(\tau)|^2_V \, d\tau \leq \|\Phi\|^2_{L^2(\mathbb{R}^2_+)}.$$

Additionally, it is possible to prove a maximum principle analogous to Theorem 2.22. This is the main tool for establishing that if Φ is given by 1.12 or 1.13, then $F(t,S_1,0) = (Ke^{-rt} - S_1)^+$ and $F(t,0,S_2) = (Ke^{-rt} - S_2)^+$.

Naturally, there may be barrier options on baskets of several assets. For two assets and a barrier independent of time, pricing the option then amounts to solving the boundary value problem:

$$\begin{cases} \dfrac{\partial F}{\partial t} - \dfrac{1}{2} \displaystyle\sum_{k,l=1}^{2} M_{k,l} S_k S_l \frac{\partial^2 F}{\partial S_k \partial S_l} - \sum_{k=1}^{2} r S_k \frac{\partial F}{\partial S_k} + rF = 0, \quad t \in (0,T], \\ \qquad\qquad F(0,S_1,S_2) = \Phi(S_1,S_2), \quad (S_1,S_2) \in \Omega, \\ \qquad\qquad F(t,S_1,S_2) = 0 \quad \text{on} \quad \partial\Omega, \end{cases}$$

for a domain $\Omega \subset \mathbb{R}^2_+$.

Chapter 3

Program implementation

We decided to develop our OOP (Object-Oriented Programming) implementation using the Python language and the famous NumPy library [18], which is well-known for numerical computation in Python. Python natively lacks a built-in array and matrix data structure (it uses lists by default). However, since 2005, NumPy has provided multidimensional array objects implemented in the C programming language. With this tool, we can perform faster numerical computations and have array objects with their expected behavior. Furthermore, we can use sparse matrices and sparse arrays via the Scipy library [30], which uses NumPy to interface with C.

Another important library we will use later is PyFreeFem [19], an API that allows us to run code written in the FreeFem language from within our Python program.

The basic structure of the project is shown in Figure 3.1. Before discussing each object in detail, it's important to provide an overview to understand the relationships between the main classes. Using a top-down approach, the first class to discuss is the `Discretizer`. This class is defined by several components:

- `DiscretizerType`: An Enum value that defines the type of discretizer, changing the behavior of the discretizer itself and its associated wrappers.
- `TimeMethod`: Represents the method used to solve the ODE linear system.
- `Dynamics`: Defines the model used to simulate the stock's random walk.
- `Option`: Represents the option for which we want to compute the fair value.

To complete the `Discretizer` class, it's important to define a domain in both the time and space variables where we will compute the solution, the discretization parameters for that domain, and any tolerances needed by the chosen method (e.g., the tolerances for embedded methods).

For the `Dynamics` object, we have defined three distinct objects to model the various market conditions that affect pricing: `DividendRate`, `Volatility`, and `InterestRate`. These objects implement the respective interfaces for dividend rate, volatility, and interest rate modeling. A constant (float) parameter is also used to define the correlation between two stocks when dealing with higher-dimensional problems.

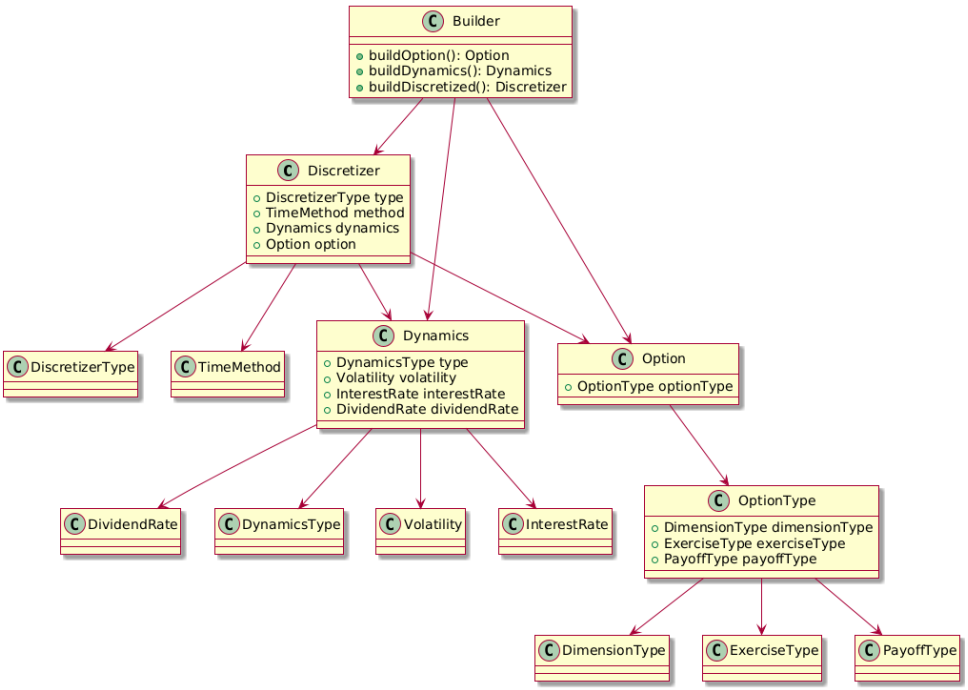


Figure 3.1: General overview of the OOP project. Some details are omitted for clarity.

Another key part of the implementation is the `OptionType` class, which maps all the options discussed in previous chapters. We achieved this by combining three defining aspects for each option using the Enum classes `DimensionType`, `ExerciseType`, and `PayoffType`, which we will discuss in detail later. To construct an option, it's mandatory to pass in all relevant market parameters, such as strike price, time to maturity, and the values of any lower or upper barriers.

Given the complexity of all these classes required to compute the fair price of an option, we also implemented a `Builder` class. This class ensures the correct construction of all objects, given an intuitive set of parameters defined by the user based on the current market conditions. The `Builder` also helps manage dependencies between objects, allowing the user to interact with just a few high-level classes while the program handles the rest.

Now that we have provided this preliminary overview, we can start analyzing each class in detail, using a bottom-up approach. This means starting with simpler classes and progressively moving towards the more complex ones, which combine multiple objects.

3.1 Enum Classes

The first classes we analyze are the Enum classes. We remind you that an Enum class is useful for defining a set of named values that represent a finite set of possible states or options. This ensures that the class is strongly typed, preventing accidental misuse of enumerator values across different Enums in

classes that utilize it. Additionally, another advantage of using these classes is the "type safety" guaranteed by the fact that numerical enumerators or strings are not used in the code to propagate information. This minimizes the risk of errors due to typos, which could potentially lead to issues with code stability. Generally speaking, the use of an Enum class is always recommended when there is a need for clearly defined, limited sets of options.

The Enum classes that we defined are:

- `DiscretizerType`: Defines the discretizer type with respect to the space variable. It can assume one of two values: *Finite Differences* or *Finite Elements*. If the chosen Enum is *Finite Differences*, it can be further specified through a sub-Enum class that allows the distinction between *Second Order* and *Fourth Order* finite difference discretization.

- `FiniteDifferencesType`: Defines the order of convergence to use when generating the matrices for the finite differences method. This class includes the method:

  - `get_macro_discretizer_type`: Returns a `FiniteDifferences` (or a `DiscretizerType` since it inherit from this class).

- `DynamicsType`: Defines the model used to simulate the evolution of stock prices over time. Currently, it can only assume the value *Black-Scholes Dynamics*. However, in a possible future development of the solver, it might also support values like *Variance Gamma Lévy Dynamics* or *Gaussian Jump Lévy Dynamics*, allowing the solution of PDEs for these jump dynamics. Although this feature wasn't the target of our discussion, we developed the code in such a way that it will be possible to add this new feature in the future with relatively little effort.

- `TimeMethodType`: Defines the method to be used for solving the system of ODEs after space discretization. It can take the following values: *Forward Euler*, *Backward Euler*, *Crank-Nicolson*, *Explicit Embedded Runge-Kutta 2*, *Diagonally Implicit Runge-Kutta 3*, *Embedded Runge-Kutta Lobatto 3C*, and *Embedded Runge-Kutta Lobatto 3B*. As the reader can observe, all these methods are implicit, and some are also embedded. In a real-world scenario, the user would be interested in computing a solution with a given tolerance in the shortest possible time. It is important to remember that financial instruments are quoted in ticks, which are never smaller than 1×10^{-3}. Therefore, it is unnecessary to compute an overly precise price for an instrument like an option. On the other hand, it is essential to be as fast as possible to react to changing market conditions within a short interval, allowing us to profit from trades with slower market participants.

- `DimensionType`: Defines the number of underlying stocks on which the payoff of an option depends, thus defining the space dimension of the problem we need to solve when computing the fair price of the option.

- `ExerciseType`: Defines the type of exercise allowed for the option. Currently, the only possible value is *European*, but other values like *American* or *Bermuda* could be introduced in future development.

- `PayoffType`: Defines the type of payoff that will be paid at maturity by the option. It can assume several values, some of which are compatible only with certain other configuration values. For example, the values *Call*, *Put*, *Asian Call*, *Asian Put*, *Digital Call*, and *Digital Put* are well-defined only for 1-dimensional options. For higher dimensions, the values *Put Min*, *Put Max*, *Put Average*, *Call Min*, *Call Max*, and *Call Average* should be used to correctly define the interaction between the different stocks to compute the final payoff. The reader can find an explanation for these in formulas 1.14, 1.13, 1.12.

3.2 Option Class

Moving on to the first class with a non-trivial implementation, let's discuss the `Option` class. Figure 3.2 presents a detailed UML diagram of the class, including all methods and object fields. To maintain a consistent code structure, we decided to unify options with and without barriers within the same class. This decision stems from the observation that an option without barriers can be considered as an option with barriers set at 0 or $+\infty$, respectively.

We also considered implementing barrier options as a wrapper around the basic option instrument, but we ultimately discarded this idea. The primary reason for this was the potential complications during the construction of the discretizer matrix (handled in the `Discretizer` class). Specifically, barrier options require a null Dirichlet boundary condition, which would have been challenging to impose elegantly using a wrapper approach. Therefore, we decided to delegate to the `Builder` the role of controlling the correct definition of the barriers, resulting in simpler code within the discretizer that is easier to understand and debug.

The `Option` class has several methods, most of which are getters for the inner `OptionType` class, and we will not discuss them in detail here. Instead, we will focus on the two methods `compute_payoff` and `convert_to_fair_price`:

- `compute_payoff`: Given a discretized domain (a NumPy array), this method computes the payoff for each point of the meshgrid. This payoff will be used as the initial condition in the resolution of the system of ODEs.

- `convert_to_fair_price`: Given a fair price (a NumPy array) computed by the discretizer and a discretized domain (also a NumPy array), this method returns the final, correct fair price estimate. This method is designed to convert the solution of the PDE problem into the corresponding real fair price of the option. This step is necessary whenever a change of variables is performed between the initial Black-Scholes PDE and the one actually used to compute the solution. For example, this occurs with Asian options in the 1-dimensional case when transitioning between the formulation in Equation 1.28 and Equation 1.29, which is easier to solve. In all other cases, this method simply returns the fair price without any transformation.

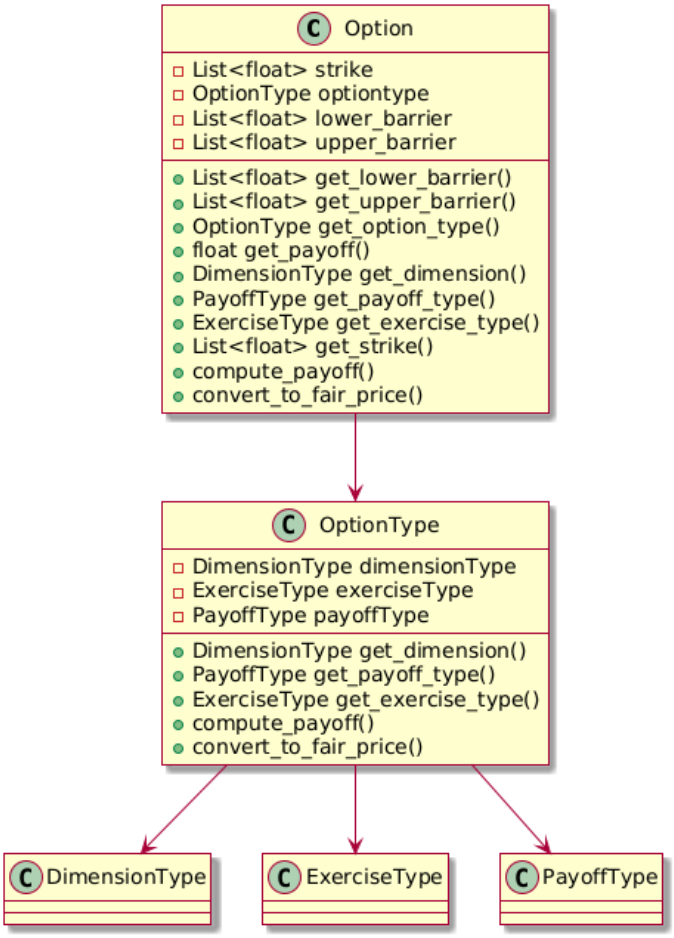


Figure 3.2: Structure of the `Option` class.

3.3 Dynamics Class

The `Dynamics` class is responsible for modeling the evolution of stock prices over time and has the structure shown in Figure 3.3. The main component for implementing this part of the code is the interface class `Dynamics`, which defines all the methods that every concrete class must implement to properly interact with the remaining classes of the code.

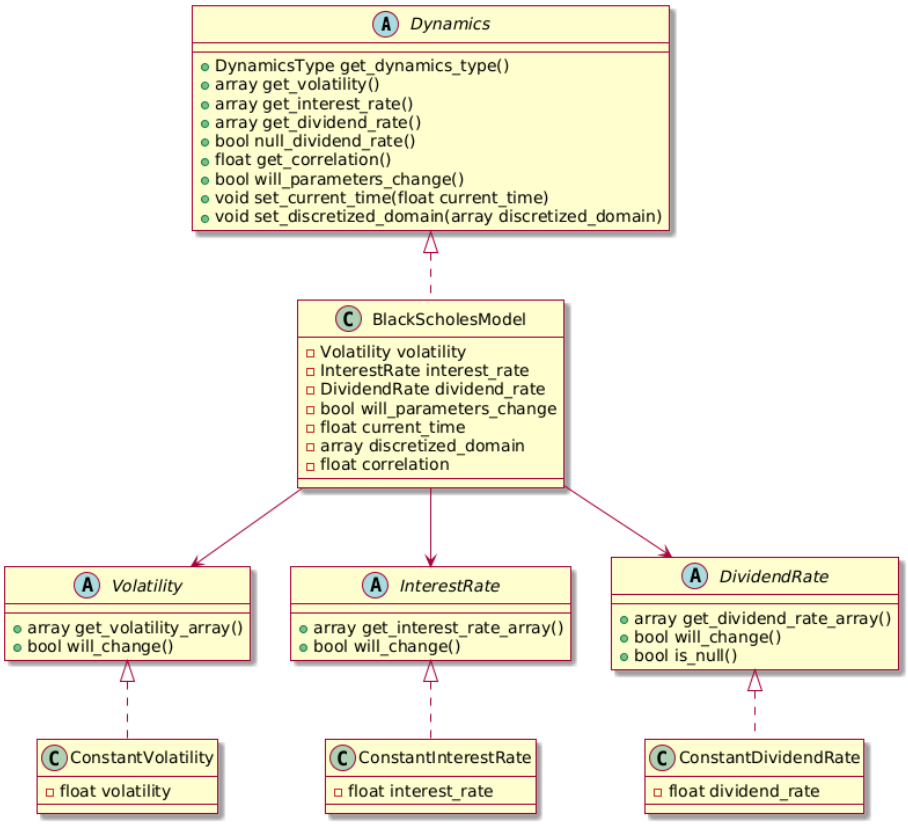


Figure 3.3: Structure of the `Dynamics` class.

Among the methods belonging to the `Dynamics` class, we want to focus our attention on the following:

- `null_dividend_rate`: This method is a getter to check if there is a null dividend rate in the dynamics configuration. This information allows the simplification of the discretization implementation within the `Discretizer` class.

- `will_parameters_change`: This method is a getter to check if the parameters may change during the evolution of time. This is important because it can help reduce the time consumed at each iteration when the `Discretizer` runs on a `Dynamics` with constant coefficients. In such cases, instead of recomputing the matrices at each time step with the same con-

stant parameters, we can reuse the previously computed matrices, thus improving the performance of our code.

- `set_current_time`: This method ensures that the `Discretizer` and the `Dynamics` always have the same current time when solving the ODEs system. This synchronization ensures that parameters are evaluated correctly when they are functions of time t. In every getter method for the market parameters, the current time is also passed to facilitate this feature.

- `set_discretized_domain`: Once the `Discretizer` has computed the discretized domain, it is passed to the `Dynamics`, which uses it to compute the values of the current market parameters for every point of the mesh grid.

The other getter methods are key for the interaction between the `Dynamics` and the `Discretizer`, but their implementation is straightforward and intuitive. They call the corresponding getter method for the inner class with the correct parameters (the discretized domain and the current time).

The first implementation of this class is the concrete class `BlackScholesModel`, which provides implementations for all the discussed methods and has the following field attributes:

- `volatility`: of type `Volatility`;

- `interest_rate`: of type `InterestRate`;

- `dividend_rate`: of type `DividendRate`;

- `current_time`: of type `float`, initialized to 0;

- `discretized_domain`: of type `NumPy array`, initialized to `None`;

- `correlation`: of type `float`.

It uses three classes: `Volatility`, `InterestRate`, and `DividendRate`. These are all interfaces with a similar structure since the way they are modeled is similar.

The `Volatility` class has the following methods:

- `get_volatility_array`: Given the discretized domain and the current time, this method computes the volatility for each node of the mesh grid and returns it as a NumPy array.

- `will_change`: Returns a boolean value indicating whether the volatility will change during the resolution of the pricing problem.

This interface is implemented by the class `ConstantVolatility`, which assumes a constant volatility for each node at any time.

A similar structure is implemented for the `InterestRate` class, which has the following methods:

- `get_interest_rate_array`: Given the discretized domain and the current time, this method computes the interest rate for each node of the mesh grid and returns it as a NumPy array.

- `will_change`: Returns a boolean value indicating whether the interest rate will change during the resolution of the pricing problem.

This interface is implemented by the class `ConstantInterestRate`, which assumes a constant interest rate for each node at any time.

The `DividendRate` interface has the following methods:

- `get_dividend_rate_array`: Given the discretized domain and the current time, this method computes the dividend rate for each node of the mesh grid and returns it as a NumPy array.
- `will_change`: Returns a boolean value indicating whether the dividend rate will change during the resolution of the pricing problem.
- `is_null`: Returns a boolean value indicating whether the dividend rate will always be null during the resolution of the pricing problem.

This interface is implemented by the class `ConstantDividendRate`, which assumes a constant dividend rate for each node at any time.

3.4 Discretizer Class

The `Discretizer` class is the core of our code, responsible for computing the fair price using the methods specified by the user. In this part of the code, all the previously discussed classes interact and combine their behaviors, ensuring the desired modularity and independence of different parts of the code.

The main class in this section of the project is the abstract `Discretizer` class, which is concretized by the `DiscretizerMeta` class. The `DiscretizerMeta` class defines all the methods that do not depend on the specific discretizer chosen. This class serves as the base class from which every other discretizer will inherit, and it has its own builder with all the routines common to every discretizer class. It defines the following methods:

- `get_dynamics_type`: Returns the value of `DynamicsType` used during the construction of the Dynamics.
- `get_time_step_size`: Returns the current size of the time step used to compute the next step of the system of ODEs. Keep in mind that this value can change during the run because methods like the embedded ones will adjust it to satisfy the user-imposed tolerances.
- `get_dynamics`: Returns the `Dynamics` object containing all the market parameters.
- `get_domain`: Returns the initial space domain used to compute the solution. This domain is expressed as a list of 2-dimensional lists, where each element of the outer list represents the interval on the respective dimension (e.g., the list `[[0,100], [20, 200]]` corresponds to the Cartesian product in $[0, 100] \times [20, 200]$ in $\mathbb{R}^2$). Note that the domain specified by the user can be truncated during the construction of the discretized domain for options like Barrier Options, because for them, the PDE is defined only in a specific region of the domain and they have boundary conditions

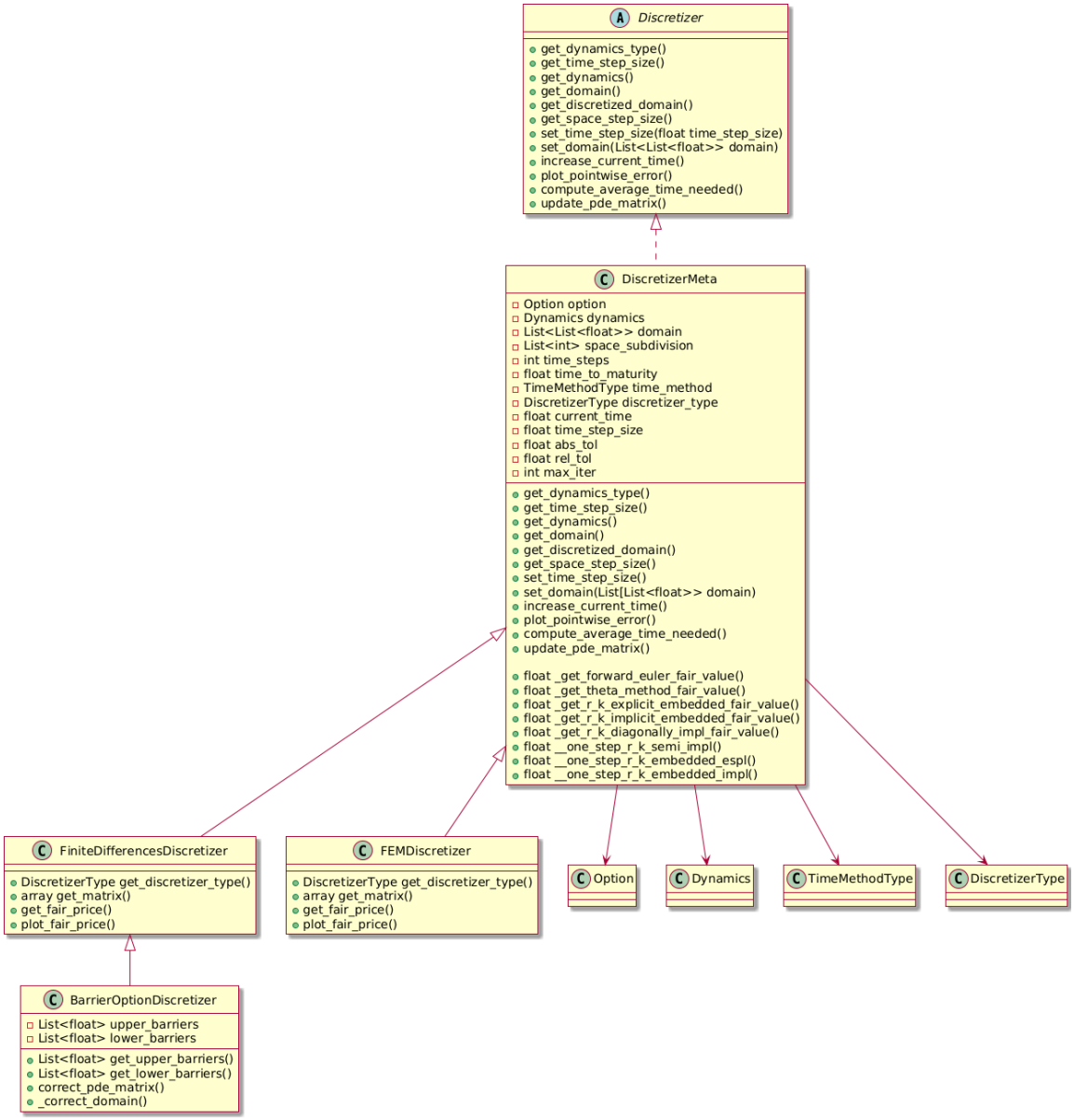


Figure 3.4: Structure of the `Discretizer` class.

to be imposed. For example, a domain of `[1, 200]` with a down-and-out barrier at 50 will generate the discretized domain on `[50,200]`. Also, note that 0 cannot be used as the left boundary of a domain because, at that point, the discretization of the PDE is poorly defined; instead, use small values that approach 0.

- `get_discretized_domain`: Returns the discretized domain, a NumPy array built from the original space domain with a discretization specified by the number of space steps.
- `get_space_step_size`: Returns the space step size of the discretized domain.
- `set_time_step_size`: This method is used by the embedded methods to properly update the value of the time step size.
- `set_domain`: This method is used by the builder method of the class to compute the discretized domain from the given domain. It also checks the consistency of the given input with the `DimensionType` inside the `Option`.
- `increase_current_time`: This method is used by the embedded methods to properly update the value of the current time according to the current step size.
- `plot_pointwise_error`: This method returns a plot of the computed option fair value alongside the corresponding analytical value computed through known formulas. The user can choose to plot the option payoff for comparison. It's also possible to generate the graph with a reference solution computed with higher accuracy. The figures presented in Chapter 4 were generated with this function.
- `compute_average_time_needed`: This method returns an average of the time needed to compute a fair price, given a number of runs on which to compute the average. This method simply runs the `get_fair_price` method several times, which we will discuss shortly.
- `update_pde_matrix`: This method updates the matrix associated with the system at each iteration of the ODE solver. Since this matrix depends on the market values of volatility, interest rate, and dividend rate, it's important to update the matrix correctly at each iteration when these parameters aren't constant. This method involves the `will_parameters_change` method of the `Dynamics` class to speed up the solver in cases of constant coefficients.

We also developed the following methods to generalize the resolution of linear systems. They are used in the `get_fair_price` method of each discretizer to solve the ODE with the correct time method:

- `_get_forward_euler_fair_value`: Solves a linear system of ODEs using the classic Forward Euler method.
- `_get_theta_method_fair_value`: Solves a linear system of ODEs using a method from the theta method family. This family includes the Backward

Euler and Crank-Nicolson methods, and these are the only two methods applied in our scope. This method requires the value of theta corresponding to the desired method.

$$y_{n+1} = y_n + k\left[(1-\theta)f(t_n, y_n) + \theta f(t_{n+1}, y_{n+1})\right]$$

- `_get_r_k_explicit_embedded_fair_value`: Solves a linear system of ODEs using the explicit embedded Runge-Kutta method described in Example 2.4. Since this is an embedded method, it's important to declare how the method behaves when the rejection rule is verified. In that case, we reject the time step size due to a high error (computed in absolute norm) between the two new estimates given by the Runge-Kutta tableau. We decided that, in these cases, the new step size would be given by

$$k_{n+1} = \min\left(2, \max\left(0.5, 0.7 \cdot \left(\frac{\text{tol}_a + \left\|\boldsymbol{y}_n^{(p-1)}\right\| \cdot \text{tol}_r}{\left\|\boldsymbol{y}_n^{(p)} - \boldsymbol{y}_n^{(p-1)}\right\|}\right)^{1/p}\right)\right) \cdot k_n \tag{3.1}$$

  where $\boldsymbol{y}_n^{(p-1)}, \boldsymbol{y}_n^{(p)}$ are the two different estimates at step n with orders $p-1$ and p. The norm used is the infinity norm, and k_n is the step size at step n. Using this formula, changes in step size are proportional to the computed error and the given tolerances but without abrupt changes in step size.

- `_get_r_k_implicit_embedded_fair_value`: Solves a linear system of ODEs using one of the implicit embedded Runge-Kutta methods described in Example 2.6, depending on the input value chosen by the user. This method uses the same update formula 3.1 as the previous function.

- `_get_r_k_diagonally_impl_fair_value`: Solves a linear system of ODEs using the semi-implicit Runge-Kutta Gauss-Legendre method described in Example 2.5.

- `__one_step_r_k_semi_impl`: This utility method is used by the solver `_get_r_k_diagonally_impl_fair_value` method, computing a single step for a generic semi-implicit Runge-Kutta method.

- `__one_step_r_k_embedded_espl`: This utility method is used by the solver `_get_r_k_explicit_embedded_fair_value` method, computing a single step for a generic explicit embedded Runge-Kutta method.

- `__one_step_r_k_embedded_impl`: This utility method is used by the solver `_get_r_k_implicit_embedded_fair_value` method, computing a single step for a generic implicit embedded Runge-Kutta method.

From the `DiscretizerMeta` class, the `FiniteDifferencesDiscretizer` and `FEMDiscretizer` classes inherit. These two classes implement the abstract methods that were left unimplemented in the base class. These are methods that find their natural implementation only after a discretizer method is chosen, as they are all related to the construction of the mesh grid, the discretizer matrix, or the computation of the fair price. We have the following methods:

- `get_discretizer_type`: Getter for the Enum `DiscretizerType` that identifies the type of discretizer.

- `get_matrix`: One of the most important methods, it performs the discretization of the PDE given by the dynamics (recall the `DynamicsType` Enum) using the method specified by the discretizer. Clearly, this method has two different implementations in the two classes. In the Finite Difference implementation, the method first computes the two matrices approximating the space derivatives using centered differences on the inner nodes and uncentered finite differences on the outer nodes (to maintain the global order of convergence of the matrix), both with order 2 or order 4 depending on the user request. Then it aggregates all the matrices into a unique matrix that defines the ODE linear system to be solved with the `get_fair_value` method. This method also manages the specific PDEs related only to a family of options, such as the 1-dimensional Asian Option, where it's possible to simplify the problem using the PDE 1.29.

  In this implementation, we decided to use the CSR sparse matrix structure provided by the SciPy package. This ensures a fast resolution of the linear system, as we will see in the next chapter. We did not implement a full matrix representation of the same matrix because the advantages of sparse matrices are well known in the numerical field.

  On the other hand, the FEM implementation of the same method uses the PyFreeFem package to execute code with the FreeFem compiler. This choice allows us to access a well-developed set of methods native to this language, such as the ability to easily perform mesh adaptation and the automatic translation from the variational formulation of the FEM problem to its solution. For our purposes, we implemented a mesh adaptation with the function `adaptmesh` on a `fespace` with polynomials of degree 1, then solved the variational problem by imposing the boundary condition and using the `convect` formula for the convective term. Between these steps, it's possible to retrieve the discretized matrix for debugging the code or performing some analysis, though it is more efficient to abandon OOP habits for a more efficient FreeFem single-run calculation.

- `get_fair_price`: Returns the fair price of the option after sequentially running all the other methods previously mentioned. It is clear that to perform this computation, the payoff and the discretizer matrix must first be computed, after which one of the methods capable of solving the ODE system must be applied.

- `plot_fair_price`: This method generates the plot for the computed fair value and compares it with the analytical one (when available) and with the payoff when requested by the user. This method was used to generate the plots in Chapter 4.

One last class we would like to mention is the `BarrierOptionDiscretizer` class, which inherits from the `FiniteDifferencesDiscretizer`. This is a necessary step to correctly price a barrier option since, unlike the general option case, the PDE presents a homogeneous boundary condition on every barrier, and the domain itself is reduced by the presence of the barrier. This behavior was

possible only by implementing the `correct_pde_matrix` and `_correct_domain` methods, which were abstract in the base class. With this class, it's possible to build all the Knock-Out Options in the 1-dimensional case with one or two barriers. This class also defines two new fields for the Discretizer: `upper_barrier` and `lower_barrier`.

3.5 Builder class

To conclude this discussion, we introduce the `Builder` class, which allows the user to easily (and correctly!) build all the necessary objects in the correct order. As seen in Figure 3.5, this single class contains all the methods required to build the three primary objects: the `Discretizer`, `Option`, and `Dynamics`. It automatically decides which object to build for each interface based on the given parameters and performs consistency checks to ensure that these parameters are reliable and manageable by the currently developed code.

To ensure correct and logically consistent use of the solver, we recommend that the user builds the objects starting with the `Dynamics`, followed by the `Option`, and finally the `Discretizer`. Once these steps are completed, the user can simply call the `get_fair_price` function of the `Discretizer` to compute the price effortlessly. Examples of how to use all the possible parameters can be found in the Solver package under the `Test` directory, which contains all the scripts used to test the library and generate the results presented in the following chapter.

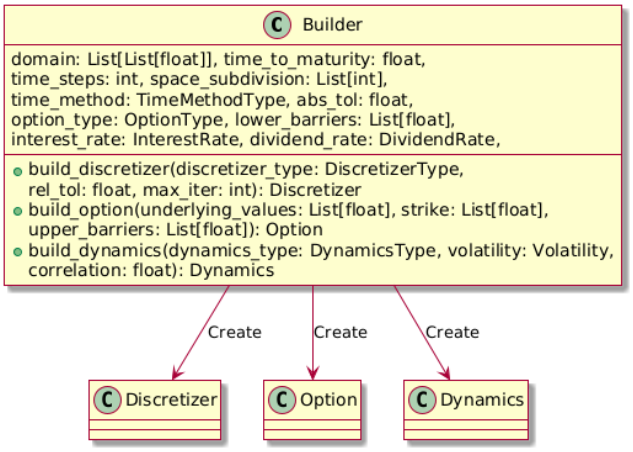


Figure 3.5: Structure of the `Builder` class.

Chapter 4

Numerical Results

4.1 European Options in 1 Dimension

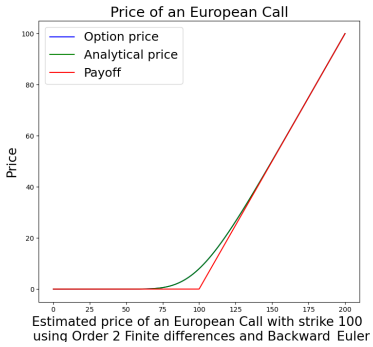
In this chapter, we present the results of our implementation discussed in the previous chapter. We report on the accuracy of the developed methods by testing their convergence and, when possible, by comparing the error with analytical solutions.

Since we are dealing with an OOP implementation, it is unnecessary to test every possible combination of the objects we developed. Proving that the interactions between classes are correct ensures that any new object (if correctly developed) will benefit from the established implementation. For example, we will not present the tests for all combinations of payoffs, time methods, and space methods performed during development. Instead, we will present individual results that demonstrate the correctness of payoff implementation or method implementation. In a few cases, we will present the complete results that combine these individual results to confirm the overall correctness of our program.

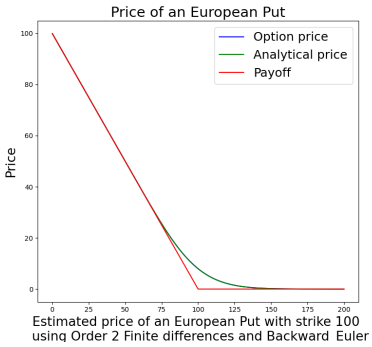
Let's begin with some generic examples to qualitatively verify that our program computes results in accordance with theoretical expectations. Consider a framework with a single stock S_t and price European Calls, Puts, Digital Calls, and Digital Puts on it, with different volatility σ, interest rate r, and dividend rate q. All options have a strike $K = 100$, time to maturity $T = 1$, on the domain $[0.1, 200]$. We computed the prices of these options using the Backward Euler method for time discretization and the second-order Finite Differences method for space discretization on the given domain using 1000 time steps and 1000 space discretizations.

In Figures 4.1, 4.2, 4.3, 4.4, and 4.5, we present all the computed prices. For each option, we also plot the corresponding analytical price (which overlaps with our computed price) and the option payoff.

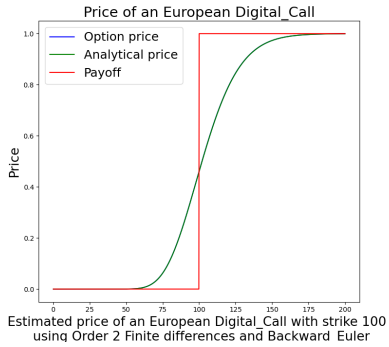
From these results, it is evident that the computed price consistently overlaps with the analytical price. Our program correctly predicts the analytical price under various market conditions and manages the presence of a constant dividend rate. In Table 4.1, the error with respect to the analytical solution and the computation time are reported. For each run, the error is lower than 0.01, and in some cases, it is even lower than 0.001. This observation is rele-



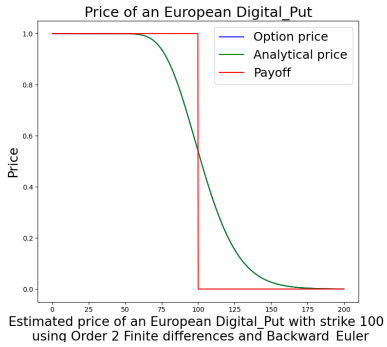
(a) *Call Option*



(b) *Put Option*

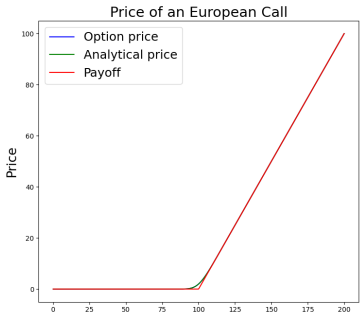


(c) *Digital Call Option*

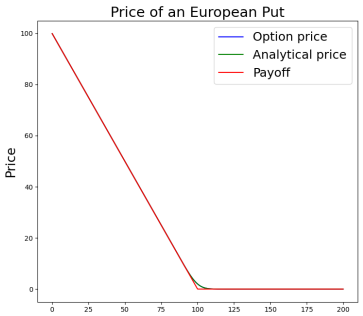


(d) *Digital Put Option*

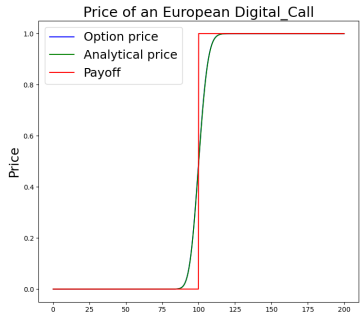
Figure 4.1: Computed and analytical options' prices with $\sigma = 0.2, r = 0, q = 0$.



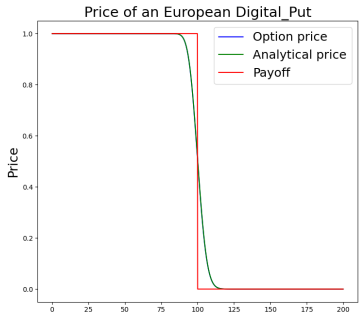
(a) *Call Option*



(b) *Put Option*

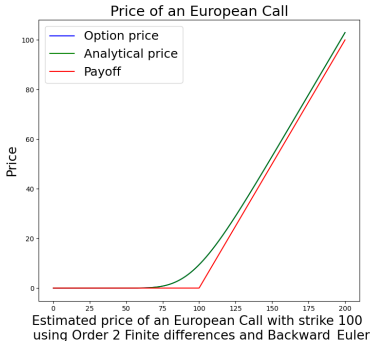


(c) *Digital Call Option*

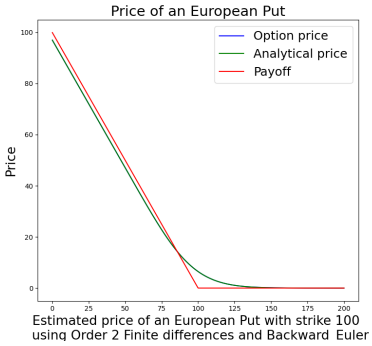


(d) *Digital Put Option*

Figure 4.2: Computed and analytical options' prices with $\sigma = 0.05, r = 0, q = 0$.

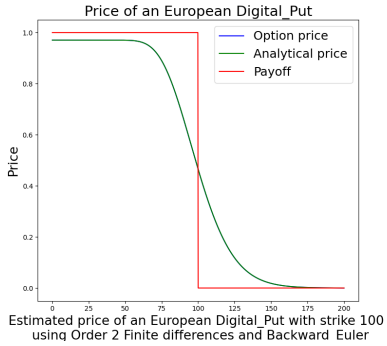


(a) *Call Option*



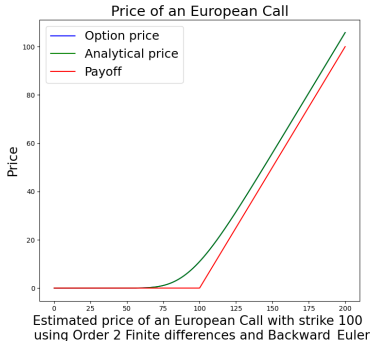
(b) *Put Option*

(c) *Digital Call Option*

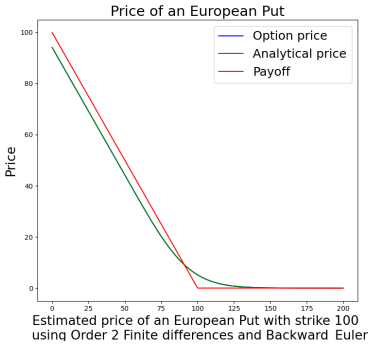


(d) *Digital Put Option*

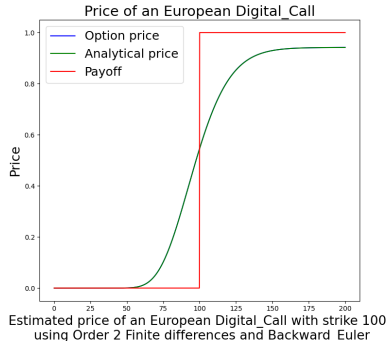
Figure 4.3: Computed and analytical options' prices with $\sigma = 0.2, r = 0.03, q = 0$.



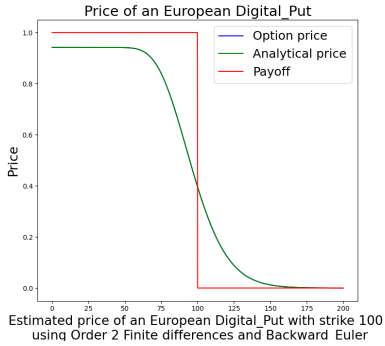
(a) *Call Option*



(b) *Put Option*

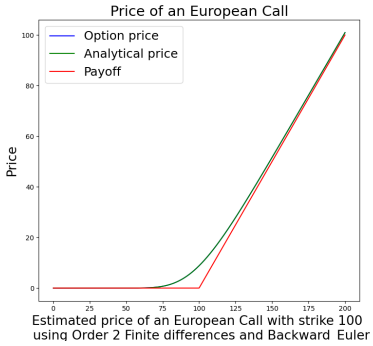


(c) *Digital Call Option*

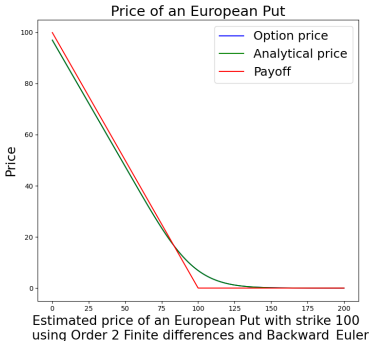


(d) *Digital Put Option*

Figure 4.4: Computed and analytical options' prices with $\sigma = 0.2, r = 0.06, q = 0$.

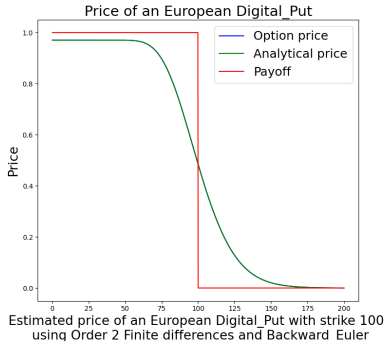


(a) *Call Option*



(b) *Put Option*

(c) *Digital Call Option*



(d) *Digital Put Option*

Figure 4.5: Computed and analytical options' prices with $\sigma = 0.2, r = 0.03, q = 0.01$.

vant because, in the financial market, most bid and ask prices are expressed in cents, with only a few cases using thousandths. Therefore, an error lower than this tick size is sufficient for our program. Moreover, the computation time is reasonable given our setup, and with better hardware, the same computation could be performed even faster.

Option type	**Market Parameters**	**Error**	**Computational time (s)**
Digital Put	σ=0.2,	1.03e-3	0.995
Digital Put	σ=0.05,	4.00e-3	0.858
Digital Put	σ=0.2, r=0.03,	9.69e-4	0.821
Digital Put	σ=0.2, r=0.06,	9.08e-4	0.845
Digital Put	σ=0.2, r=0.03, q=0.01	9.79e-4	0.810
Digital Call	σ=0.2,	8.50e-84	0.698
Digital Call	σ=0.05,	7.46e-8	0.720
Digital Call	σ=0.2, r=0.03,	3.53e-82	0.739
Digital Call	σ=0.2, r=0.06,	1.25e-80	0.869
Digital Call	σ=0.2, r=0.03, q=0.01	1.03e-82	0.993
Call	σ=0.2,	2.88e-3	1.073
Call	σ=0.05,	3.48e-4	0.933
Call	σ=0.2, r=0.03,	2.00e-3	0.589
Call	σ=0.2, r=0.06,	1.42e-3	0.577
Call	σ=0.2, r=0.03, q=0.01	2.22e-3	0.589
Put	σ=0.2,	2.91e-3	0.597
Put	σ=0.05,	3.48e-4	0.608
Put	σ=0.2, r=0.03,	1.95e-3	0.719
Put	σ=0.2, r=0.06,	1.28e-3	0.892
Put	σ=0.2, r=0.03, q=0.01	2.21e-3	1.113

Table 4.1: Error and Time used for each computation.

With these preliminary results, we have shown that the computed results are correct. However, the computation time varies depending on the method used. As previously discussed, our program allows the use of different time methods and finite difference orders. In short, we have the following time methods: Forward Euler, Backward Euler, Crank-Nicolson, Explicit Embedded Runge-Kutta 2, Diagonally Implicit Runge-Kutta 3, Embedded Runge-Kutta Lobatto 3C, and Embedded Runge-Kutta Lobatto 3B. The finite difference methods are available in second and fourth-order versions.

To study the errors and computational times of the different time methods used to solve the same problem, we again consider the previous framework with $\sigma = 0.2, r = 0, q = 0$ and compute the price for a European Call using 1000 time steps and 500 space discretizations for the second order and 2000 time steps and 300 space discretizations for the fourth order. Tables 4.2 and 4.3 present the results.

From these results (and from other runs that are not reported here), it is evident that some methods are generally better than others in achieving a similar error in less time. For example, Crank-Nicolson is unsurprisingly slightly faster than Backward Euler, but the fastest method is Runge-Kutta Lobatto 3B, which computes the solution in 36 milliseconds—an interesting computation time for a high-frequency trading program, for instance. The Forward Euler

method, however, did not converge due to its instability when the time step is not sufficiently smaller than the space discretization. This characteristic is well-known for the Forward Euler method (and explicit methods in general) and results in a model that is too parameter-dependent, making it unfeasible for our implementation. For this reason, we focus on implicit methods to ensure greater stability. Regarding space discretization methods, the fourth-order method can compute reasonable results, but it does not guarantee better performance due to the irregularity of the payoff function and the more complex matrices that complicate the resolution of the associated linear system.

In conclusion, the Runge-Kutta Lobatto 3B method is an interesting model for this type of computation and will undoubtedly consume less time on better hardware and after parameter tuning. On the space side, since most of our applications will involve payoffs that are not differentiable (or continuous), we will not use the fourth-order finite differences method as a reference but will instead rely on its second-order counterpart.

Time Method	**Error**	**Time Elapsed (s)**
Forward Euler	NaN [1]	0.043
Backward Euler	2.041e-3	0.965
Crank Nicolson	2.007e-3	0.753
Explicit Embedded Runge Kutta 2	1.955e-3	2.042
Diagonally Implicit Runge Kutta 3	2.006e-3	1.539
Embedded Runge Kutta Lobatto 3C	1.998e-3	3.764
Embedded Runge Kutta Lobatto 3B	2.015e-3	0.026

Table 4.2: Comparison between Time Methods with second-order finite differences.

Time Method	**Error**	**Time Elapsed (s)**
Forward Euler	NaN [1]	0.022
Backward Euler	4.671e-3	0.756
Crank Nicolson	4.654e-3	0.692
Explicit Embedded Runge Kutta 2	5.967e-3	0.852
Diagonally Implicit Runge Kutta 3	5.910e-3	1.370
Embedded Runge Kutta Lobatto 3C	5.722e-3	4.083
Embedded Runge Kutta Lobatto 3B	5.908e-3	0.024

Table 4.3: Comparison between Time Methods with fourth-order finite differences.

To test the order of convergence of our method, we divide the discussion into space and time variables. For space, we implemented only two types of finite difference matrices: second and fourth-order. To test the correctness of these matrices, we solved the same problem with $\sigma = 0.2, r = 0.3, q = 0, K = 100$ using both second-order and fourth-order discretizers in space, with

[1] With the given discretization of 1000 time steps and 500 spatial discretization points for the 2^{nd} discretization order (resp. 2000 and 300 for the 4^{th} discretization order), we observe the instability of the Forward Euler method because the length of the time step is not small enough to adequately handle the space discretization error.

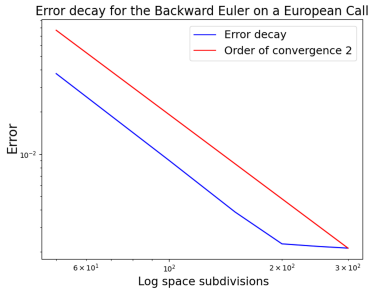
Backward Euler and Crank-Nicolson in time, on the domain $[0.1, 200]$ with time to maturity $T = 1$. Figure 4.6 shows the order of convergence between the computed solution and the analytical one for the four different computations, with space subdivisions of (50, 100, 150, 200, 250, 300) and 10000 time steps. It is clear from this figure that our program struggles to achieve an error lower than 1e-3, and moreover, the error does not decay as expected. This is because we are trying to solve the Black-Scholes equation, which is naturally defined on the half-line $\mathbb{R}^+$, on a truncated domain $[0.1, 200]$. This unavoidable truncation generates an error on the right boundary during pricing since it is impossible to impose a boundary condition on the right side that would correct this behavior. Indeed, if we look at the point-wise error between the computed and analytical solutions (Figure 4.7 for the point-wise error of the second-order Backward Euler computation; similar results were found in other cases), we can clearly see that our program correctly manages the pricing around the strike, with a decay of the error, but it maintains a constant error on the right boundary. Considering the error between the computed solutions and a reference solution with 1000 space subdivisions and 10000 time steps (Figure 4.8), we see that our program correctly follows the error decay of the space discretization but converges to the *wrong* solution used as a reference. From this, we can conclude that our implementation is correct but corrupted by the impossibility of solving the given problem in its proper domain. Nevertheless, this error is always around 1e-3 and is located far from the strike, where options are usually more liquid, making the error negligible compared to the usual bid-ask spread, which would be significant for options that are far out of the money (or far in the money).

Moving on to the time method order of convergence, we observe the same problem discussed above: all methods struggle to correctly estimate the solution on the right boundary due to the truncation of the domain. In the convergence graph in Figure 4.9, this behavior is not visible when compared with the analytical solution because of the chosen step size, but it becomes more evident with finer meshes. In Figure 4.10, we see that all the methods achieve their expected order of convergence when compared with a reference solution.

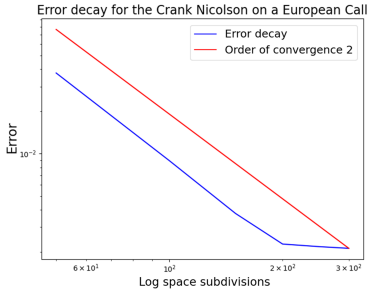
A specific discussion is required for the only embedded time model we developed. For this type of model, a classic order of convergence analysis is not meaningful. Instead, it is possible to test whether the developed model can match its time step size to the problem in such a way as to achieve the desired absolute and relative tolerances. Indeed, solving the same pricing problem for a Call Option with $\sigma = 0.2, r = 0.3, q = 0, K = 100$ on the domain $[0.1, 200]$ with time to maturity $T = 1$ using second-order finite differences, and using different tolerances, we obtain Figure 4.11 and Table 4.4, where the input tolerances and the output results are reported.

From these results, it is clear that our model correctly adapts the time step size to the problem to achieve the desired tolerance. Looking at the log-log plot, we see that all the points lie on a straight line, and the error is always lower than the requested tolerance. Unfortunately, the time required for this computation is too long compared to the time required by the Runge-Kutta Lobatto 3B method presented earlier. Therefore, our best choice remains the Runge-Kutta Lobatto 3B method.

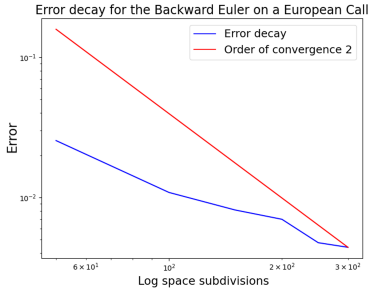
To conclude the discussion for the 1-dimensional case, we can state that, as expected, the implicit methods are preferable for this type of problem. Among them, the Runge-Kutta 3B method stands out in terms of order of convergence



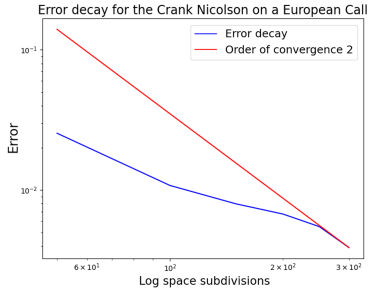
(a) *Second-order finite difference with Backward Euler.*



(b) *Second-order finite difference with Crank-Nicolson.*



(c) *Fourth-order finite difference with Backward Euler*

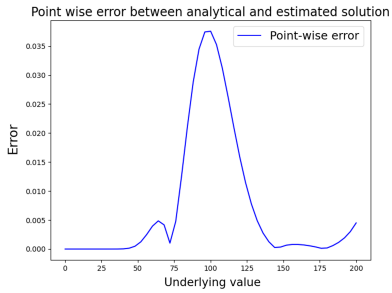


(d) *Fourth-order finite difference with Crank-Nicolson.*

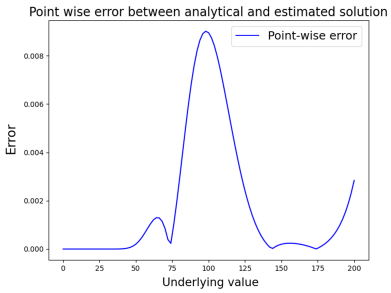
Figure 4.6: Space error order of convergence w.r.t. the analytical solution.

abs tol	**rel tol**	**Error**	**Elapsed Time (s)**
1e-2	1e-4	2.554e-3	1.838
1e-3	1e-5	3.673e-4	1.793
1e-4	1e-6	3.118e-5	2.013
1e-5	1e-7	2.799e-6	1.905
1e-6	1e-8	2.504e-7	2.035
1e-7	1e-9	4.532e-8	2.881

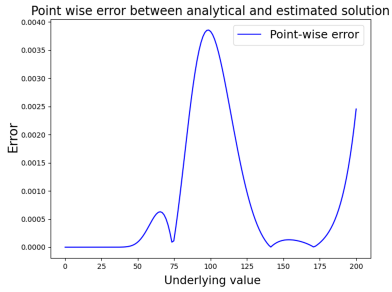
Table 4.4: Comparison of Errors and Elapsed Time for Different Tolerance Values



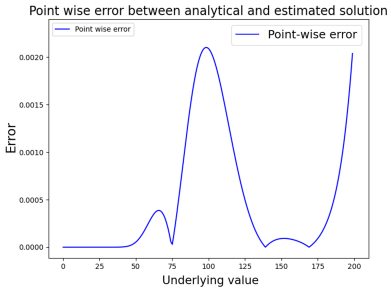
(a) *50 Space subdivision*



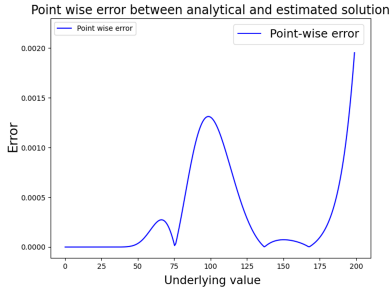
(b) *100 Space subdivision*



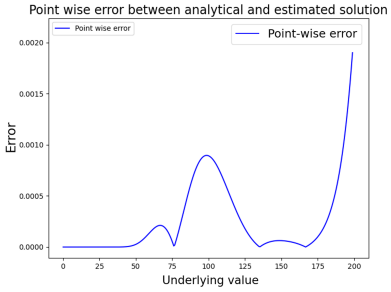
(c) *150 Space subdivision*



(d) *200 Space subdivision*

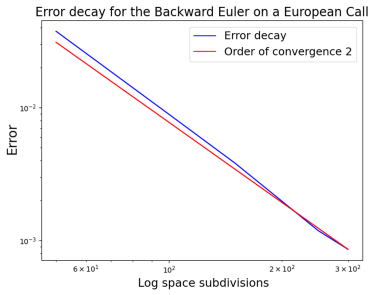


(e) *250 Space subdivision*

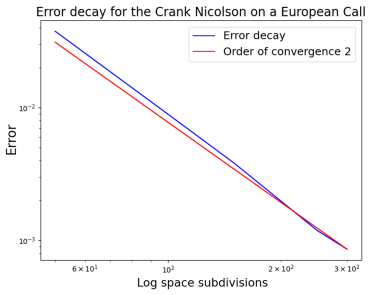


(f) *300 Space subdivision*

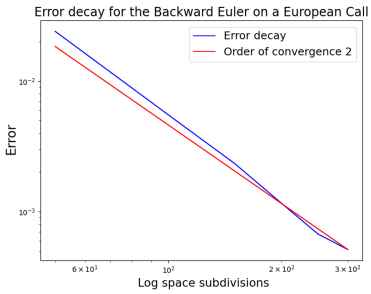
Figure 4.7: Point-wise error w.r.t the analytical solution for the second-order Backward Euler computation.



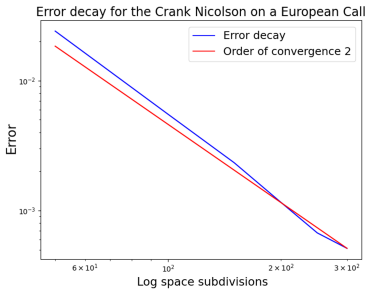
(a) *Second-order finite difference with Backward Euler.*



(b) *Second-order finite difference with Crank-Nicolson.*

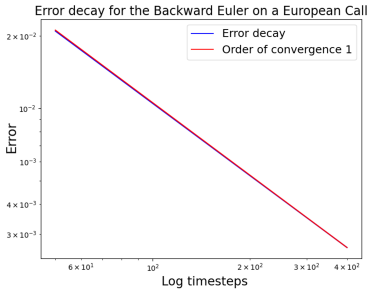


(c) *Fourth-order finite difference with Backward Euler*

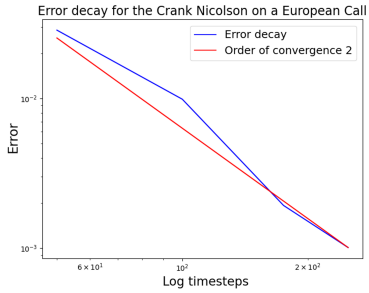


(d) *Fourth-order finite difference with Crank-Nicolson.*

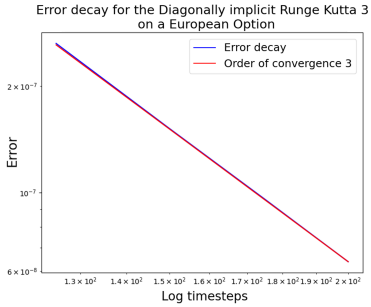
Figure 4.8: Space error order of convergence w.r.t. the reference solution.



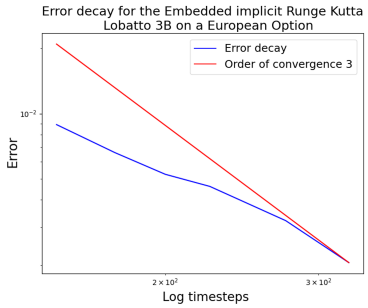
(a) *Order of convergence for the Backward Euler method.*



(b) *Order of convergence for the Crank-Nicolson method.*

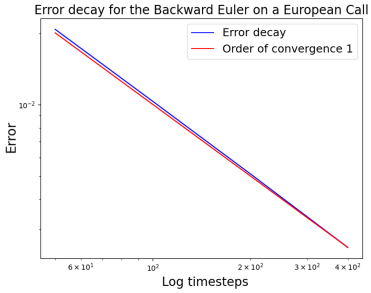


(c) *Order of convergence for the Explicit Runge-Kutta 3 method.*

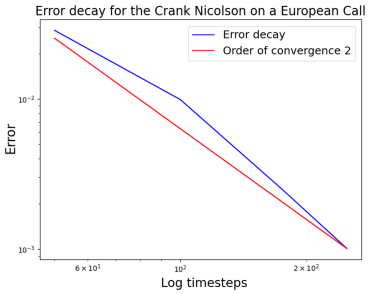


(d) *Order of convergence for the Runge-Kutta Lobatto 3B method.*

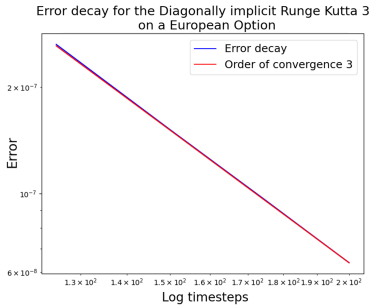
Figure 4.9: Time order of convergence for some of the implemented time methods w.r.t. the analytical solution.



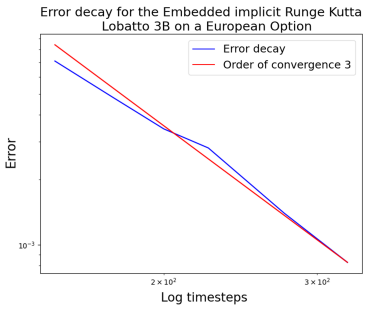
(a) *Order of convergence for the Backward Euler method.*



(b) *Order of convergence for the Crank-Nicolson method.*



(c) *Order of convergence for the Explicit Runge-Kutta 3 method.*



(d) *Order of convergence for the Runge-Kutta Lobatto 3B method.*

Figure 4.10: Time order of convergence for some of the implemented time methods w.r.t. the reference solution.

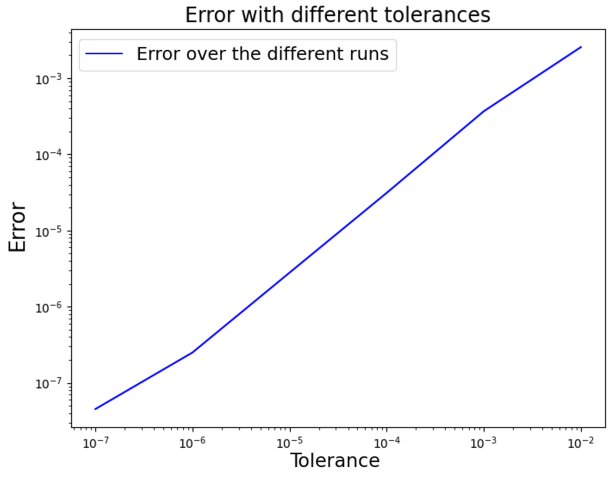


Figure 4.11: Decay of the error as the absolute tolerance is reduced.

and computational time. We are optimistic that, with better machines, the computation can be performed in less time, making it possible to use this tool to price European Options in a few milliseconds, as required for high-frequency trading models in the event of volatility or interest rate shocks. Furthermore, it may be possible to consider using these models after tuning the parameters to perform the computation on a smaller domain, thereby neglecting points near the boundaries where we are not interested in estimating a price.

4.2 Barrier Options in 1 dimension

We now focus on the 1-dimensional knockout barrier option within the context of European options. As discussed in Section 2.3.2, a barrier option can be computed by solving the Black-Scholes PDE on the domain between two barriers, with boundary conditions ensuring that the price becomes zero at the barriers. In Section 1.3.1, we also found that using the method of images, we can derive an explicit analytical solution for a 1-dimensional barrier option, which can serve as a reference for our solver.

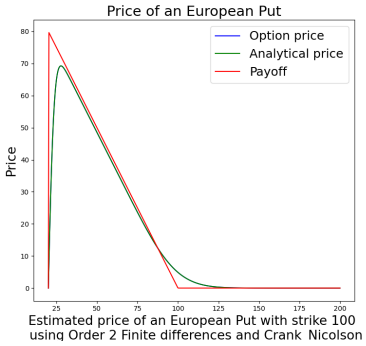
We will test our solver (capable of solving only knockout barrier options) on up-and-out Call options and down-and-out Put options. Explicit formulas for the prices of these options can be derived by combining the results found in Section 1.3.1. From Propositions 1.24, 1.25, and 1.26, along with the Put-Call parity of Proposition 1.27, we can obtain a closed-form expression for the price of a down-and-out Put $\boldsymbol{P_{bO}}(t, s, K)$. Similarly, the price of an up-and-out Call $\boldsymbol{C^{BO}}(t, s, K)$ can be obtained using the price of an up-and-out zero-coupon bond $\boldsymbol{BO}^{BO}(t, s)$, the price of an up-and-out stock $\boldsymbol{ST}^{BO}(t, s)$, and the price of an up-and-out European Put $\boldsymbol{P}^{BO}(t, s)$.

To test our solver, we consider a framework with a single stock with market conditions $\sigma = 0.2$, $r = 0.3$, $q = 0$, $K = 100$. We solve the pricing problems for down-and-out Puts with barriers $b = 20, 40, 60, 80$ and up-and-out Calls with barriers $B = 120, 140, 160, 180$ on the domain $[0.01, 200]$ (reduced to the relevant sub-domain above a lower barrier b or below an upper barrier B). Using a second-order finite difference method in space and a Crank-Nicolson method in time with 1000 time steps and 500 space subdivisions (on the relevant domain), we obtained the results shown in Figure 4.12 for the Puts and Figure 4.13 for the Calls, alongside the analytical solutions. The point-wise errors between our solutions and the analytical ones are reported in Table 4.5, Figure 4.14 for the Puts, and Figure 4.15 for the Calls.

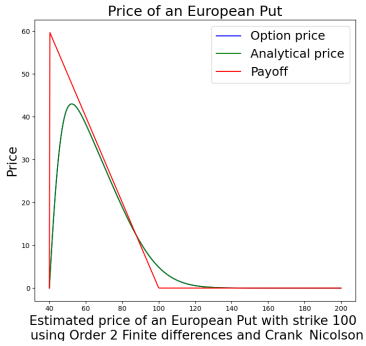
From these errors, we observe that, as expected, there are unavoidable errors on the right boundary where we cannot impose a boundary condition (e.g., for the Puts in this case). However, this error disappears when a boundary condition is imposed (e.g., for the Calls). This result is consistent with our earlier discussion, where we noted the issue of truncating the Black-Scholes model, leading to errors whenever we cannot impose a boundary condition.

We also solved more general pricing problems for options without an analytical pricing formula. Specifically, we tested our program on Puts and Calls with double knockout barriers and on digital Calls and Puts with single and double barriers. As before, we used $\sigma = 0.2$, $r = 0.3$, $q = 0$, $K = 100$ on the domain $[0.01, 200]$, applying a second-order finite difference method in space and a Crank-Nicolson method in time with 1000 time steps and 500 space sub-

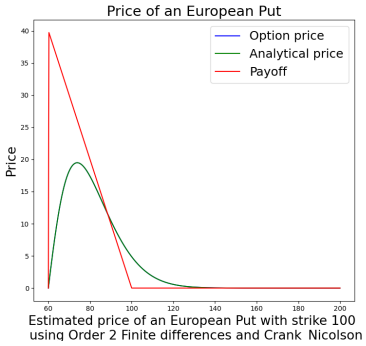
divisions. The results are shown in Figures 4.16, 4.17, and 4.18. These results are qualitatively consistent with financial intuition, though a deeper analysis is not possible without an analytical formula. Nonetheless, we are satisfied with these results because we were able to validate the prices when possible, and the qualitative behavior is as expected. The reader should note that the differing profiles in these figures are due to the fact that while the payoff intervals are the same length (30) in both cases, the options provide payoffs in different regions of the half-line $\mathbb{R}^+$, leading to slight differences in the expected probability distribution under the Black-Scholes model.



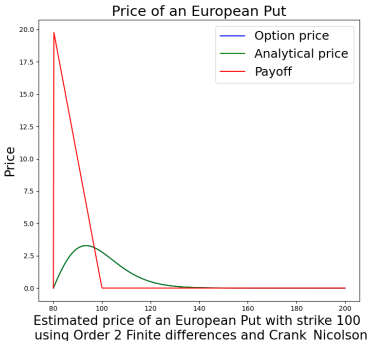
(a) *Down-and-Out Put Option with barrier* $b = 20$.



(b) *Down-and-Out Put Option with barrier* $b = 40$.



(c) *Down-and-Out Put Option with barrier* $b = 60$.

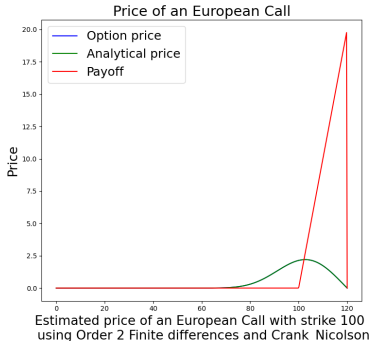


(d) *Down-and-Out Put Option with barrier* $b = 80$.

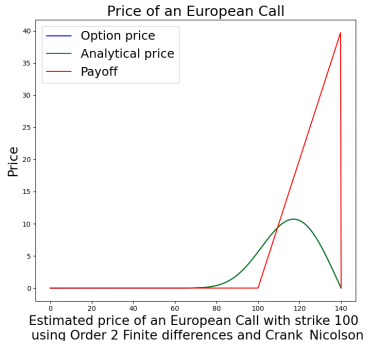
Figure 4.12: Computed Option price compared with analytical Option price for some Down-and-Out Put Options.

4.3 Asian Options in 1 Dimension

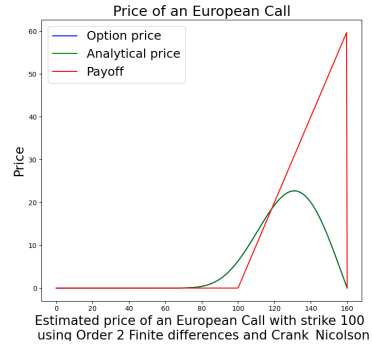
As discussed in Section 1.3.3, for the 1-dimensional case of Asian options, which involves a 2-dimensional PDE in space, we can simplify the problem through a change of variables in specific cases. Particularly, for the floating strike Asian Call, we can consider the PDE 1.29 instead of 1.28, where the price function F



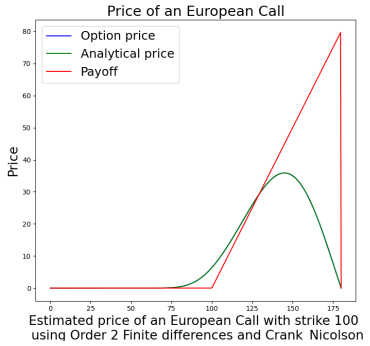
(a) *Up-and-Out Call Option with barrier* $B = 120$.



(b) *Up-and-Out Call Option with barrier* $B = 140$.

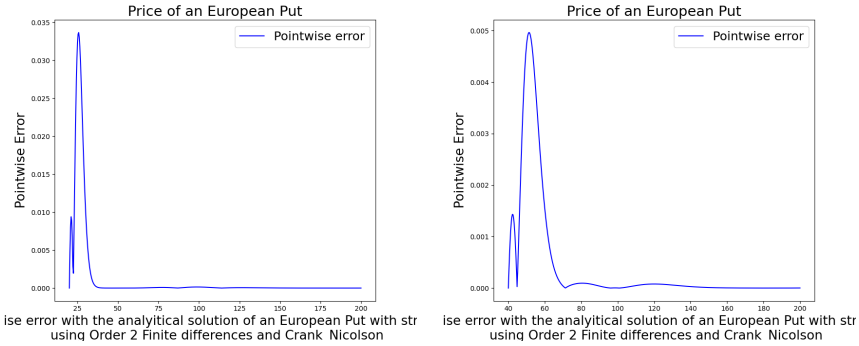


(c) *Up-and-Out Call Option with barrier* $B = 160$.



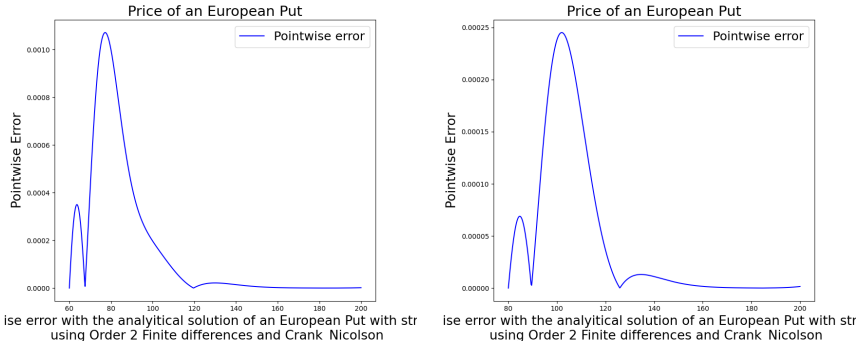
(d) *Up-and-Out Call Option with barrier* $B = 180$.

Figure 4.13: Computed Option price compared with analytical Option price for some Up-and-Out Call Options.



(a) *Point-wise error for Down-and-Out Put Option with barrier* $b = 20$.

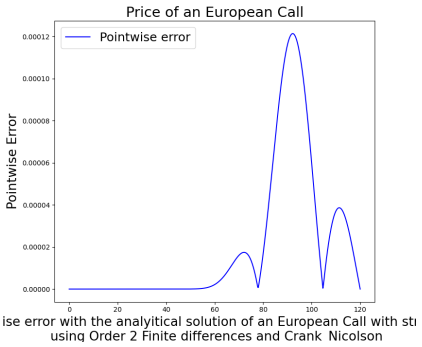
(b) *Point-wise error for Down-and-Out Put Option with barrier* $b = 40$.



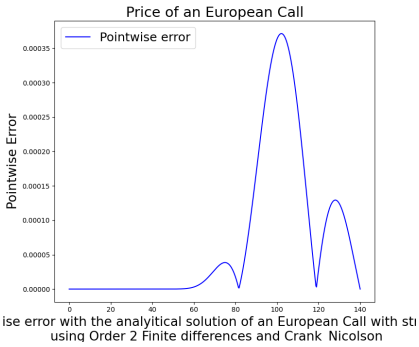
(c) *Point-wise error for Down-and-Out Put Option with barrier* $b = 60$.

(d) *Point-wise error for Down-and-Out Put Option with barrier* $b = 80$.

Figure 4.14: Point-wise error between the computed Option price and the analytical Option price for some Down-and-Out Put Options.

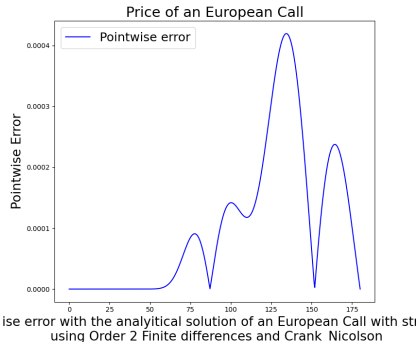


(a) *Point-wise error for Up-and-Out Call Option with barrier* $B = 120$.



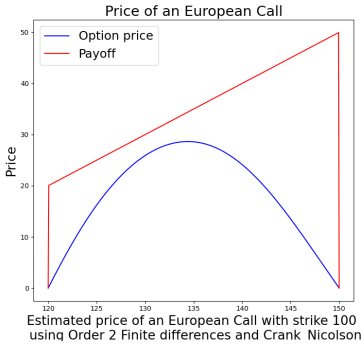
(b) *Point-wise error for Up-and-Out Call Option with barrier* $B = 140$.

(c) *Point-wise error for Up-and-Out Call Option with barrier* $B = 160$.

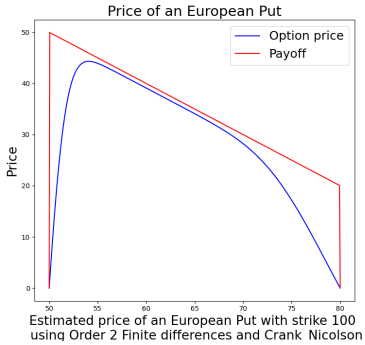


(d) *Point-wise error for Up-and-Out Call Option with barrier* $B = 180$.

Figure 4.15: Point-wise error between the computed Option price and the analytical Option price for some Up-and-Out Call Options.



(a) *Price for a double knockout Call Option with barriers* $b = 120, B = 150$.

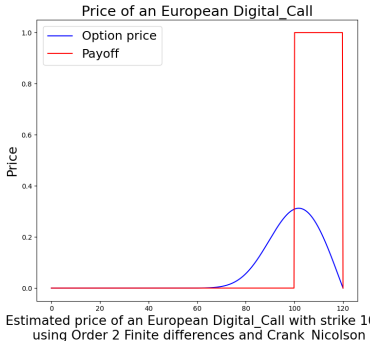


(b) *Price for a double knockout Call Option with barriers* $b = 50, B = 80$.

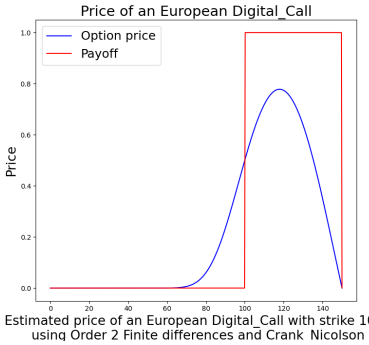
Figure 4.16: Computed price for Call and Put Options with double knockout barriers.

Option Type	Lower Barrier	Upper Barrier	Error	Elapsed Time (s)
Call	0	110	3.781e-5	0.772
Call	0	120	1.214e-4	0.687
Call	0	130	1.894e-4	0.648
Call	0	140	3.713e-4	0.575
Call	0	150	2.780e-4	0.547
Call	0	160	2.706e-4	0.543
Call	0	170	3.894e-4	0.544
Call	0	180	4.195e-4	0.559
Call	0	190	4.963e-4	0.551
Put	10	inf	0.1677	0.680
Put	20	inf	0.0337	0.694
Put	30	inf	0.0117	0.645
Put	40	inf	0.0050	0.827
Put	50	inf	0.0023	0.825
Put	60	inf	0.0011	0.618
Put	70	inf	0.0005	0.718
Put	80	inf	0.0002	0.696
Put	90	inf	6.029e-5	0.664

Table 4.5: Comparison of Errors and Elapsed Time for Different Option Barriers

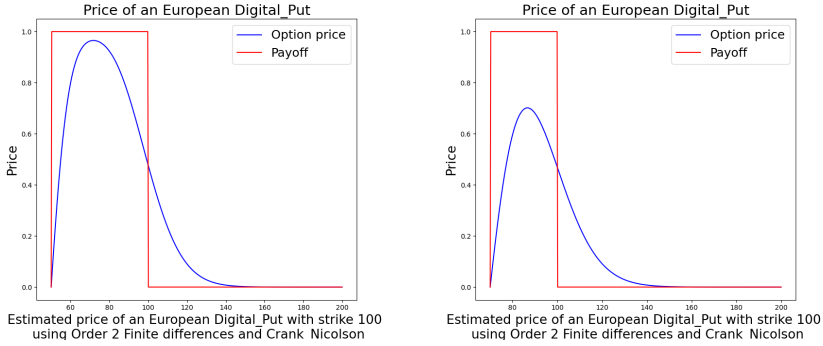


(a) *Price for a knockout Digital Call Option with barrier* $b = 120$.



(b) *Price for a knockout Digital Call Option with barrier* $b = 150$.

Figure 4.17: Computed price for Digital Call Options with single knockout barriers.



(a) *Price for a knockout Digital Put Option with barrier* $B = 50$.

(b) *Price for a knockout Digital Put Option with barrier* $B = 70$.

Figure 4.18: Computed price for Digital Put Options with single knockout barriers.

and the new variable ξ are given by:

$$F(t, S, A) = S\omega\left(t, \frac{K - tA/T}{S}\right), \quad \xi = \frac{K - tA/T}{S}.$$

Through algebraic manipulation, we can express the variable ξ as:

$$\xi = -\frac{1}{S}\left(\frac{t}{T}(A - K) + \frac{t - T}{T}K\right).$$

From this, we derive the transformation for the term $(A-K)$ used in our floating strike Asian Call:

$$(A - K) = -\frac{T}{t}S\xi + \frac{t - T}{t}K = -S\xi,$$

and the payoff becomes:

$$\Phi(S) = (A - K)^+ = (-S\xi)^+.$$

Assuming $\Phi = S\tilde{\Phi}$, we obtain:

$$\tilde{\Phi} = (-\xi)^+,$$

which serves as the new boundary condition.

To define the domain for ξ, we must analyze its behavior and interpretation. Notably, at $t = 0$, ξ equals $\frac{K}{S}$, indicating that when reversing the transformation, we are interested only in the domain where ξ is positive. However, during the resolution of the PDE, ξ can take both positive and negative values, typically ranging between -2 and 2. Indeed, ξ behaves similarly to the moneyness $(\frac{K}{S})$ at $t = 0$.

In implementing the solver for the Asian Option, we kept these developments in mind, alongside considerations for the Fixed Strike Put Option and the Floating Strike Call. Using our implementation, we solved the PDE and

obtained the plot in Figure 4.19, using the usual market parameters $\sigma = 0.2$, $r = 0.3$, $q = 0$, $K = 100$, and solving on the space domain $(0, 200)$. In the figure, the x-axis is represented as the moneyness according to the previously discussed development.

Since an analytical price for this type of option is not currently available, we cannot directly compare our results to validate the implementation. However, we are confident in its correctness due to the shape of the solution, which aligns with financial intuition.

Figure 4.19: Option price for a fixed strike Asian Call.

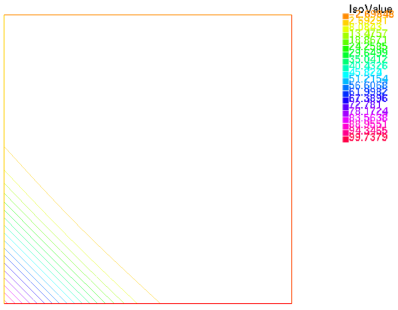
4.4 Basket Options in 2 Dimensions

For the implementation of Basket Options in multiple dimensions, we utilized the Python API *pyfreefem* [13], which enables communication with the FreeFEM language. This approach allowed us to leverage the built-in functions in FreeFEM for mesh refinement and numerical solutions of the Black-Scholes model. The FreeFEM documentation [19] demonstrates the use of the Black-Scholes equation as a prototype for introducing evolution problems (*click here*). We used this code as a foundation to develop our solver for Basket Options. As the correctness of this implementation has already been established in the documentation, we do not provide additional validation results here.

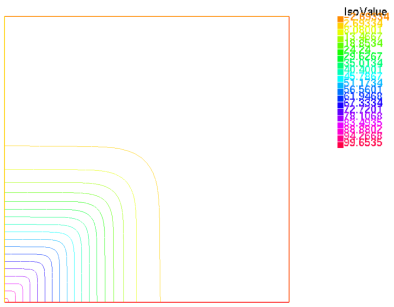
Among the various payoff functions we defined, we will focus on the payoffs related to Put Options. As indicated by theory (see Theorem 2.15 and subsequent results) and numerical findings, these options tend to be more stable due to their payoffs generally belonging to L^2 (see 1.12, 1.13, 1.14). When this regularity is present, the computed option prices are well-shaped and reliable. For instance, in the first two plots in Figure 4.20, the level sets of the solution align with the definition of the payoff. Conversely, in the third image, the level sets in the top-right region begin to exhibit concavity, resulting in an unrealistic shape not entirely explained by the payoff definition. This outcome is expected since the first two payoffs belong to the L^2 space, while the third does not.

Similarly, Call Option prices exhibit the same irregularities due to the same

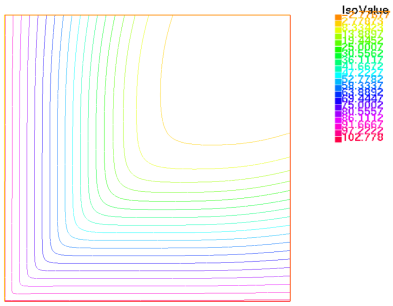
issues, as shown in Figure 4.21. The results presented in Figures 4.20 and 4.21 are based on a market framework with constant parameters $\sigma = 0.2$, $r = 0.03$, $q = 0$, and a correlation between the two stocks of $\rho = 0.3$, on the domain $[0.01, 200]$, with a strike price of 100.



(a) *Price for a Put Option with payoff 1.12.*

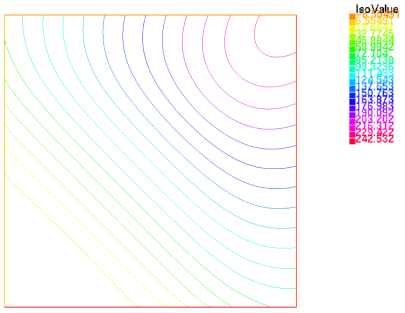


(b) *Price for a Put Option with payoff 1.13.*

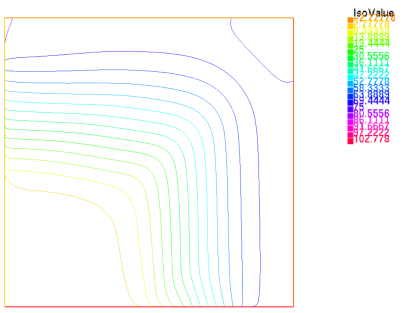


(c) *Price for a Put Option with payoff 1.14.*

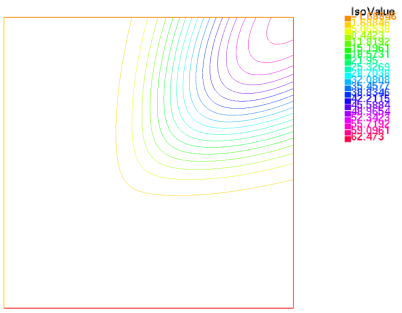
Figure 4.20: Computed price for Put Options with different payoffs.



(a) *Price for a Call Option with payoff 1.15.*



(b) *Price for a Call Option with payoff 1.16.*



(c) *Price for a Call Option with payoff 1.17.*

Figure 4.21: Computed price for Call Options with different payoffs.

Bibliography

[1] Yves Achdou, Olivier Bokanowski, and Tony Lelièvre. "Partial differential equations in finance". In: *The Encyclopedia of Financial Models* 2 (2012).

[2] Yves Achdou and Olivier Pironneau. *Computational methods for option pricing*. SIAM, 2005.

[3] Alain Bensoussan. "On the theory of option pricing". In: *Acta Applicandae Mathematica* 2 (1984), pp. 139–158.

[4] Alain Bensoussan and Jacques Louis Lions. "Impulse control and quasi-variational inequalities". In: *(No Title)* (1984).

[5] Tomas Björk. *Arbitrage theory in continuous time*. Oxford university press, 2009.

[6] Andrei N Borodin and Paavo Salminen. "Handbook of Brownian motion-facts and formulae". In: *Journal of the Royal Statistical Society-Series A Statistics in Society* 160.3 (1997), p. 596.

[7] Michael J Brennan and Eduardo S Schwartz. "The valuation of American put options". In: *The Journal of Finance* 32.2 (1977), pp. 449–462.

[8] Susanne C Brenner. *The mathematical theory of finite element methods*. Springer, 2008.

[9] John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[10] Thierry Cazenave and Alain Haraux. *An introduction to semilinear evolution equations*. Vol. 13. Oxford University Press, 1998.

[11] Antoine Conze and Viswanathan. "Path dependent options: The case of lookback options". In: *The Journal of Finance* 46.5 (1991), pp. 1893–1907.

[12] Black F. and Scholes M. "The pricing of options and corporate liabilities". In: *Journal of political economy* 81.3 (1973), pp. 637–654.

[13] Florian Feppon. "Shape and topology optimization of multiphysics systems". In: (2019).

[14] Avner Friedman. *Partial differential equations of parabolic type*. Courier Dover Publications, 2008.

[15] Dariusz Gatarek and Swiech Andrzej. "Optimal stopping in Hilbert spaces and pricing of American options". In: *Mathematical methods of operations research* 50.1 (1999), pp. 135–147.

[16] M Barry Goldman, Howard B Sosin, and Mary Ann Gatto. "Path dependent options:" Buy at the low, sell at the high"". In: *The Journal of Finance* 34.5 (1979), pp. 1111–1127.

[17] Ernst Hairer, Syvert Paul Nørsett, Gerhard Wanner, et al. "Solving Ordinary Differential Equations I [electronic resource]: Nonstiff Problems". In: ().

[18] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[19] Frédéric Hecht et al. *FreeFem++ Documentation*. Version 4.9. 2021. URL: https://doc.freefem.org/.

[20] Patrick Jaillet, Damien Lamberton, and Bernard Lapeyre. "Variational inequalities and the pricing of American options". In: *Acta Applicandae Mathematica* 21 (1990), pp. 263–289.

[21] Ioannis Karatzas. "On the pricing of American options". In: *Applied mathematics and optimization* 17.1 (1988), pp. 37–60.

[22] Ioannis Karatzas. "Optimization problems in the theory of continuous trading". In: *SIAM Journal on Control and Optimization* 27.6 (1989), pp. 1221–1259.

[23] Angelien Gertruda Zinnia Kemna and Antonius Cornelis Franciscus Vorst. "A pricing method for options based on average asset values". In: *Journal of Banking & Finance* 14.1 (1990), pp. 113–129.

[24] Enrico Magenes and Jacques Louis Lions. "Problèmes aux limites non homogènes et applications". In: *(No Title)* (1968).

[25] Andrea Pascucci. *A short course on American options*. 2009.

[26] Amnon Pazy. *Semigroups of linear operators and applications to partial differential equations*. Vol. 44. Springer Science & Business Media, 2012.

[27] Murray H Protter and Hans F Weinberger. *Maximum principles in differential equations*. Springer Science & Business Media, 2012.

[28] Kristin Reikvam. "Viscosity solutions of optimal stopping problems". In: *Stochastics and Stochastic Reports* 62.3-4 (1998), pp. 285–301.

[29] L Chris G Rogers and Zo Shi. "The value of an Asian option". In: *Journal of Applied Probability* 32.4 (1995), pp. 1077–1088.

[30] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[31] Xiao Lan Zhang. "Numerical analysis of American option pricing in a jump-diffusion model". In: *Mathematics of Operations Research* 22.3 (1997), pp. 668–690.