universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master Thesis

„Monte Carlo vs Finite Difference in the Computation of Mathematical Finance“

verfasst von / submitted by

Gabrijel Zelić, univ. bacc. math.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 821
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Mathematik UG2002
Betreut von / Supervisor:	Ass.-Prof. Dr. Julio Daniel Backhoff-Veraguas
Mitbetreut von / Co-Supervisor:	Xin Zhang, BSc PhD

Before I start with the formal part of my thesis, let me jot down a couple of sentences of gratefulness.

First of all, let me express my huge gratitude to my two menthors - Julio Daniel Backhoff-Veraguas and Xin Zhang - for their generous help and useful suggestions throughout the process of writing the thesis. All stages of the process, such as getting familiar with the topic, understanding it and tackling main problems, were facilitated by their extensive mathematical knowledge and great experience. It was a pleasure to work with both of you.

Secondly, I want to say thank you to all other professors that I have met during my studying at the University of Vienna and University of Zagreb. Studying mathematics is sometimes tough, but always exciting adventure and I am thankful that you were sharing your knowledge with me, allowing me to grow as a mathematician and a person. In addition, I am grateful for all colleagues and fellow students who were part of my journey and helped me in any way.

Thirdly and finally, I want to mention my family - mother, father, sister and brother. Thank you and God for making my study-abroad wish come true. Your unconditional love, huge support and faith in the triune God were crucial for me while studying both in Zagreb as well as in Vienna. Thank you once again!

Abstract [EN]

This master's thesis aims to compare numerical performances of two pricing methods - Monte Carlo and finite difference method - in the case of the European options. After providing a mathematical theory that supports both procedures, we will present Python implementations of the methods. The culmination of the thesis is the accuracy comparison between Monte Carlo and finite difference algorithm.

Abstract [DE]

Das Ziel dieser Masterarbeit ist es die numerische Effizienz zweier Pricing-Methoden zu vergleichen. Dabei werden wir die Monte Carlo Methode und die Methode der finiten Differenzen für den Fall der europäischen Optionen untersuchen. Es werden, sowohl die mathematischen Hintergründe, als auch eine Implementierung in Python präsentiert. Den Höhepunkt der Arbeit wird ein Genauigkeitsvergleich der beiden Methoden darstellen.

Contents

1 General introduction

As the title of this master's thesis suggests, its main goal is to present two numerical methods for option pricing, provide Python codes for their implementation and compare their accuracy in the case of European options. The methods we consider are Monte Carlo and finite difference method. To begin with, let us firstly introduce options, important contracts on financial markets.

Apart from "riskless" bank account and risky stocks, so called *derivatives* play significant role in the financial markets. They are called derivatives because their price is usually derived from another asset (most often called *underlying asset*). Typical example of derivatives are options.

Let us denote a stock price process with $S = (S_t)_{t\in[0,T]}$. A **call option** (or simply a call) is a contract between the buyer and the seller of the option that gives the buyer of the contract the right, but not the obligation, to **buy** a stock (from the seller) for a predefined (**strike**) price K at a certain time T (**the expiration date**). The value $C = V_C(T, S_T)$ of the call at the expiration date T is given with the payoff function Ψ:

$$C = V_C(T, S_T) = \Psi(S_T) := \begin{cases} S_T - K, & \text{if } S_T > K \\ 0, & \text{if } S_T \leq K \end{cases}$$

In contrast to that, a **put option** (or simply a put) is a contract between the buyer and the seller of the option that gives the buyer of the contract the right, but not the obligation, to **sell** a stock (to the seller of the put) at the strike price at the expiration date. The value $P = V_P(T, S_T)$ of the put at the expiration date T is

$$P = V_P(T, S_T) = \Psi(S_T) := \begin{cases} K - S_T, & \text{if } K > S_T \\ 0, & \text{if } K \leq S_T \end{cases}$$

The equation that connects the values of the put and the call option at time t with the same properties (same underlying, volatility, the expiry date and the strike price) is called put-call parity:

$$C(t, S(t)) - P(t, S(t)) = S(t) - e^{-r(T-t)}K$$

where $x \mapsto e^x$ is exponential function, and r is a constant risk free interest rate.

Let us mention that the full name of this contract is **European** call/put option, but we will usually omit the prefix European in this paper for simplicity reason. There are a lot of option types; for instance, options that can be exercised before the expiry date T are called American options. But we will stick to the European options in this paper and consider only this type. We will usually denote options with $V_C(t, S_t), V_P(t, S_t)$ (hint: value of a call respective put at time t) or just shortly with letters C, P.

An asymmetry between the rights and obligations of the buyer and the seller of the option is obvious. The buyer has the right, but not the obligation to exercise the contract; on the contrary, according to the contract, the seller must participate (i.e. has no other choice) in the selling/buying activity. Hence the seller of the option expects a **premium** as a compensation for this asymmetry - the premium is most often called the price of the option. The main goal of the theory of financial mathematics is to determine the fair price of the option, i.e. the price that does not allow a chance to earn money without risk (so called arbitrage opportunity).

In real life, option pricing procedure might be done in different ways and two of them are applying Monte Carlo (MC) or finite difference (FD) method. Additionally, when it comes to European options, evaluation of the, so called, Black-Scholes-Merton formula gives us the price of an option. We will use this analytic formula in order to compare as well as analyze accuracy of two methods.

The whole paper is more devoted to presenting how option pricing procedures are carried out rather than to writing all details of the rigorous and mathematically exact derivations that stand behind the algorithms. Therefore, the thesis will concentrate on giving hints, examples, useful exercises and intuitive explanations more than going deeply into proofs and derivations. Additionally, we will provide Python codes for implementation of both MC and FD method. The main source we will follow throughout this paper is the book of **Rüdiger U. Seydel - "Tools for Computational Finance"**. Sometimes we will denote it with [1] for brevity reasons, as this is the notation from the bibliography page.

Before we embark on this journey, let me say a couple of sentences about each chapter. The thesis consists of six chapters, including the first one which you are reading at the moment. The second one briefly summarizes basics of the probability theory and states key results from the stochastic (Itô) calculus and mathematical finance,

without proving any of those because they are usually part of many books and script notes that cope with these topics. Chapter 3 is devoted to Monte Carlo algorithm; we describe mathematical background of the method and thereafter present the basic algorithm which we improve applying variance reduction method. The core of the fourth chapter is explanation of the finite difference method which will tackle famous partial differential equation - Black-Scholes-Merton equation. Purpose of chapter 5 is a comparison between the two methods we considered; we will provide results of Python implementations of both methods and try to figure out which one seems to have better performances. The sixth (and the very last) chapter will be devoted to drawing some conclusions from the whole master's thesis, especially from chapter 5.

That was in a nutshell about this paper.

Let's get started! :)

2 Prerequisites

The main idea of chapter 2 is to remind ourselves of the foundations of probability theory and stochastic processes, as well as recall key concepts of stochastic analysis (Itô calculus) and mathematical finance in discrete and continuous time. We will introduce crucial definitions and state important results, but we will not spend time proving them because all of them are typical parts of many books related to the topic of stochastics. Main sources for this chapter are scrypt notes "Financijsko modeliranje 1" and "Financijsko modeliranje 2", both written by Vanja Wagner, "Advanced Probability" whose author was Perla Sousi, and "Stochastic Calculus and Applications" which was created by Nathanaël Berestycki. We start off with the field of probability and stochastic processes.

2.1 Basics of Probability Theory and Stochastic Processes

This section plays role of a reminder of elementary definitions and pivotal assertions in the world of stochastic processes. We recall notions like probability space, random variables, conditional expectation, martingales, Brownian motion and its properties.

Definition 1 (σ-algebra) *Let X be a set. A family $\mathcal{A}$ of subsets of X is called a σ-algebra over X (on X) if following conditions are satisfied:*

- $X \in \mathcal{A}$
- $A \in \mathcal{A} \Longrightarrow A^c \in \mathcal{A}$
- $A_n \in \mathcal{A} \ \forall n \in \mathbb{N} \Longrightarrow \bigcup_{n=1}^{\infty} A_n \in \mathcal{A}$

A tuple $(X, \mathcal{A})$ is called measurable space.

Definition 2 (Measure) *Let $(X, \mathcal{A})$ be a measurable space. A function $\mu : \mathcal{A} \to [0, \infty)$ is called a measure on $(X, \mathcal{A})$ if following conditions are satisfied:*

- $\mu(\emptyset) = 0$
- $\mu(\bigcup_{n=1}^{\infty} A_n) = \sum_{n=1}^{\infty} \mu(A_n)$ *if $(A_n)_{n\in\mathbb{N}}$ is a sequence of sets from $\mathcal{A}$ such that $A_i \cap A_j = \emptyset \ \forall i \neq j$. This property is called σ - additivity.*

A triple $(X, \mathcal{A}, \mu)$ is called a measure space. If, additionally, $\mu(X) = 1$ holds, then $(X, \mathcal{A}, \mu)$ is called a probability space.

Definition 3 (Measurable function) *Let $(X, \mathcal{A})$ and $(Y, \mathcal{B})$ be measurable spaces. A function $f : X \to Y$ is ($\mathcal{A}$, $\mathcal{B}$)-measurable if*

$$f^{-1}(B) \in \mathcal{A} \ \forall B \in \mathcal{B}.$$

From now on, we fix probability space (Ω, $\mathcal{F}$, $\mathbb{P}$) and work on it.

Definition 4 (Random variable) *$X : \Omega \to \mathbb{R}$ is a random variable if it is a $(\mathcal{F}, \mathcal{B}(\mathbb{R}))$-measurable function on $(\Omega, \mathcal{F}, \mathbb{P})$, where $\mathcal{B}(\mathbb{R})$ is the Borel σ-algebra generated by open sets in $\mathbb{R}$.*

We briefly recall the notion of expectation for a random variable X. To do this, we need three steps. Firstly, let us assume that $X = \sum_{i=1}^{n} c_i \mathbb{1}_{A_i}$ with $A_i \in \mathcal{F}$ for each $i = 1, ..., n$, i.e. X is a simple function (that is - linear combination of n indicator functions). In this case, expectation of X is defined as

$$\mathbb{E}(\textstyle\sum_{i=1}^{n} c_i \mathbb{1}_{A_i}) := \sum_{i=1}^{n} c_i \mathbb{P}(A_i).$$

Secondly, let X be a positive random variable. Since there exists a sequence $(X_n)_{n\in\mathbb{N}}$ of simple functions that increasingly converge to X ($X_n \nearrow X$), expectation of X is defined with

$$\mathbb{E}(X) := \lim_{n\to\infty} \mathbb{E}(X_n).$$

Thirdly and finally, for a general random variable X, expectation is defined as

$$\mathbb{E}(X) := \mathbb{E}(X^+) - \mathbb{E}(X^-)$$

as soon as one of the terms in the previous sum is finite, where $X = X^+ - X^-$ because $X^+ = max(X, 0)$ and $X^- = max(-X, 0)$. After these three steps, we can say that a random variable X is integrable if $\mathbb{E}(|X|) < \infty$.

We say that some property holds ***almost surely*** (a.s.) if there is a set $F \in \mathcal{F}$ such that the property holds for each $x \in \mathcal{F}$ and $\mathbb{P}(F^C) = 0$. Next, we state the definition of the conditional expectation, which is actually a theorem.

Theorem 1 (Conditional expectation) *Let X be an integrable random variable and let $\mathcal{F}$ and $\mathcal{G}$ be two σ-algebras such that $\mathcal{G} \subseteq \mathcal{F}$ (i.e. $\mathcal{G}$ is a sub-σ-algebra of $\mathcal{F}$). Then there exists a random variable Y such that:*

- *Y is $\mathcal{G}$ - measurable,*
- *Y is integrable,*
- *$\mathbb{E}(X\mathbb{1}_A) = \mathbb{E}(Y\mathbb{1}_A) \ \ \forall A \in \mathcal{G}$*

Y is called (a version of) conditional expectation of X given $\mathcal{G}$ and it is denoted with $Y = \mathbb{E}(X|\mathcal{G})$. Additionally, if there exists Y' that satisfies previous three conditions, then $Y' = Y$ a.s.

Definition 5 (generated σ-algebra) *Let $\mathcal{E}$ be a family of subsets of Ω. The σ-algebra $\sigma(\mathcal{E})$ generated by $\mathcal{E}$ is the smallest σ-algebra that contains $\mathcal{E}$, i.e.*

$$\sigma(\mathcal{E}) := \bigcap\{\mathcal{F} : \mathcal{F} \subseteq \mathcal{E}, \mathcal{F} \ \ \sigma - algebra\}.$$

Furthermore, if $(X_i)_{i\in I}$ is a family of random variables, then the smallest σ-algebra generated by $(X_i)_{i\in I}$ is

$$\sigma((X_i)_{i\in I}) := \sigma(\{\omega \in \Omega : X_i(\omega) \in B\} : i \in I, B \in \mathcal{B}(\mathbb{R}))$$

i.e. $\sigma((X_i)_{i\in I})$ is the smallest σ-algebra that makes $(X_i)_{i\in I}$ measurable.

Definition 6 (Stochastic process) *A stochastic process X is a sequence of random variables $(X_t)_{t\in T}$ for some index set T. The process $X = (X_t)_{t\in T}$ can be considered as a random map: for every $\omega \in \Omega$ we associate it with a path from T to $\mathbb{R}$, i.e. $\forall\omega \in \Omega \ \ t \mapsto X_t(\omega)$. If $T = \mathbb{N}_0$, X is called discrete time stochastic process; if $T = \mathbb{R}_+$, X is called continuous time stochastic process.*

Usually we will consider continuous time processes in the following definitions, but now we present one of the most famous stochastic process - the **random walk**, which is defined in discrete time.

Example 1 (Random walk) *Let $X_n, n \in \mathbb{N}$, be a random variable with Bernoulli distribution with parameter $p = \frac{1}{2}$, i.e.*

$$\mathbb{P}(X_n = 1) = \mathbb{P}(X_n = -1) = \tfrac{1}{2}$$

and let $X_0 = 0$. Then, it is clear that $\mathbb{E}(X_n) = 0, Var(X_n) = 1 \ \forall n \in \mathbb{N}$. Moreover, let us define new random variable

$$S_n := \textstyle\sum_{i=1}^{n} X_i, n \in \mathbb{N}, S_0 := 0 \ .$$

Stochastic process $(S_n)_{n\geq 0}$ is called the symmetric random walk.

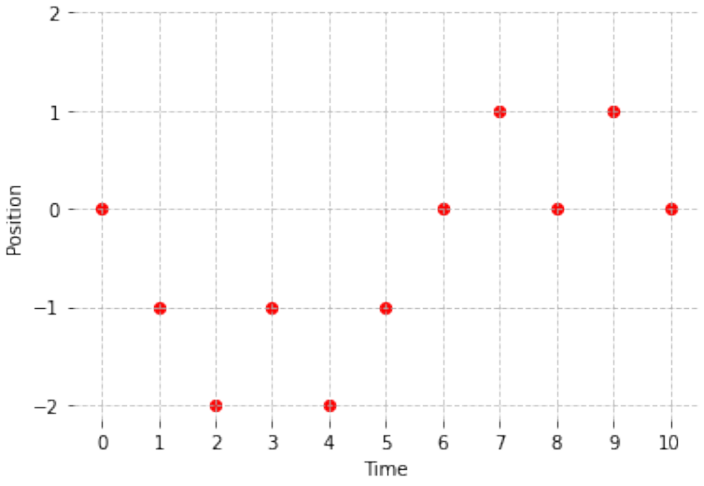

Figure 1: One path of a random walk for n = 10

Definition 7 (Filtration) *A family of σ-algebras $(\mathcal{F}_t)_{t\geq 0}$ ($\mathcal{F}_t \subseteq \mathcal{F}$ for every t) is called a filtration if it increases, i.e. if $m < n \Rightarrow \mathcal{F}_m \subseteq \mathcal{F}_n$. A probability space $(\Omega, \mathcal{F}, \mathbb{P})$ on which we have the filtration $(\mathcal{F}_t)_{t\geq 0}$ is called a filtered probability space and it is denoted with $(\Omega, (\mathcal{F}_t)_{t\geq 0}, \mathcal{F}, \mathbb{P})$.*

Definition 8 (Adapted stochastic process) *A stochastic process $X = (X_t)_{t\geq 0}$ is adapted to the filtration $(\mathcal{F}_t)_{t\geq 0}$ if X_t is $\mathcal{F}_t$-measurable $\forall t \geq 0$.*

Definition 9 (Stopping time) *Stopping time τ with respect to the filtration $(\mathcal{F}_t)_{t\geq 0}$ is a random variable $\tau : \Omega \to \mathbb{R}_+ \cup \{+\infty\}$ such that $\{\tau \leq t\} \in \mathcal{F}_t \ \ \forall t \in \mathbb{R}_+$*

Definition 10 (Martingale - continuous time) *A stochastic process $X = (X_t)_{t\geq 0}$ is called a martingale if:*

- *X is adapted to $(\mathcal{F}_t)_{t\geq 0}$*
- *X is integrable, i.e. $\mathbb{E}(|X_t|) < \infty \ \ \forall t \geq 0$*
- *$\mathbb{E}(X_t|\mathcal{F}_s) = X_s \ \ \forall t \geq s$.*

After introducing random walk $(S_n)_{n\in\mathbb{N}}$, we want to generalize it to more complex and continuous time stochastic process. To do so, by using linear interpolation, for fixed $n \in \mathbb{N}$ we define $B^{(n)} = (B_t^{(n)} : t \geq 0)$ with:

$$B_t^{(n)} := \frac{1}{\sqrt{n}} S_{[nt]} + \frac{1}{\sqrt{n}}(S_{[nt]+1} - S_{[nt]})(nt - [nt]), \qquad t \geq 0. \tag{1}$$

$B^{(n)}$ is usually called a rescaled random walk. It is worth noticing that, for fixed $n \in \mathbb{N}$ and for all $t \in \mathbb{R}_+$ such that $tn \in \mathbb{Z}$, it holds

$$B_t^{(n)} = \tfrac{1}{\sqrt{n}} S_{[nt]}.$$

Also, take note that for a fixed $\omega \in \Omega$, $t \mapsto B_t^{(n)}(\omega)$ is a continuous[1] map from $\mathbb{R}_+$ to $\mathbb{R}$, which is usually referred to as a path (trajectory) of the rescaled random walk for a given $\omega \in \Omega$. Even though it is not rigorously mathematically defined as a limit, we can think of a Brownian motion as a limit of the rescaled random walk $B^{(n)}$ when $n \to \infty$. Now let us state a version of central limit theorem.

Theorem 2 (Central limit theorem) *For $t \geq 0$ it holds that $B_t^{(n)}$ converges in distribution to $\mathcal{N}(0, t)$ when $n \to \infty$.*

Definition 11 (Brownian motion) *A stochastic process $B = (B_t)_{t\geq 0}$ is called a Brownian motion if:*

- *$t \mapsto B_t(\omega)$ is a continuous function from $\mathbb{R}_+$ to $\mathbb{R}$ almost surely*
- *$B_0 = 0$*
- *$\forall m \in \mathbb{N} \ \ 0 = t_0 < t_1 < t_2 < ... < t_m$ increments $B_{t_1}, B_{t_2} - B_{t_1}, B_{t_3} - B_{t_2}, ..., B_{t_m} - B_{t_{m-1}}$ are independent*
- *$\forall t, s \in \mathbb{R}_+, \ \ t > s \geq 0, \quad B_t - B_s \sim \mathcal{N}(0, t - s)$.*

Definition 12 (Variation of first order) *Let $f : [0, T] \to \mathbb{R}$ be a function and let $\Pi = \{t_0, t_1, ..., t_n\}$ be a partition of $[0, T]$ such that $0 = t_0 < t_1 < ... < t_n = T$ with $||T|| := \max_{j=0,1,...,n-1}(t_{j+1} - t_j)$. Variation of the first order of the function f on $[0, T]$ is defined as*

$$V_T(f) := \lim_{||\Pi||\to 0} \sum_{j=1}^{n-1} |f(t_{j+1}) - f(t_j)|.$$

Example 2 *Let $f : [0, T] \to \mathbb{R}$ be a monotone function such that $f(0) = a, f(T) = b$. Then $V_T(f) = b - a$.*

Definition 13 (Total variation process) *Let $A : \Omega \times [0, \infty] \to \mathbb{R}$ be a stochastic process. Its total variation process V is defined pathwise ($\forall \omega \in \Omega$) as the total variation of function $A(\omega, .)$. We say that A is of finite variation if $A(\omega, .)$ is of finite variation $\forall \omega \in \Omega$.*

Remark 1 *Brownian motion is **not** of finite variation.*

Definition 14 (Quadratic variation process) *Let $f : [0, T] \to \mathbb{R}$ be a function and let $\Pi = \{t_0, t_1, ..., t_n\}$ be a partition of $[0, T]$ such that $0 = t_0 < t_1 < ... < t_n = T$ with $||T|| := \max_{j=0,1,...,n-1}(t_{j+1} - t_j)$. Quadratic variation of f on $[0, T]$ is defined as*

[1] Informally, to visualize the plot of a path of the rescaled random walk, you might take a look at Figure 1 and connect adjacent red dots with straight line. If you, additionally, speed up time and diminish the step of a random walk (as it is done in the equation (1)), you end up with the plot of the path of the rescaled RW.

$$[f,f](T) := \lim_{||\Pi||\to 0} \sum_{j=1}^{n-1} |f(t_{j+1}) - f(t_j)|^2.$$

Stochastic process $X = (X_t, t \geq 0)$ *is of finite quadratic variation if there exists a stochastic process* $[X] = ([X]_t : t \geq 0)$ *such that*

$$\sum_{j=1}^{n-1} |X_{t_{j+1}} - X_{t_j}|^2 \xrightarrow{\text{in probability}} [X]_T \quad \text{when } ||\Pi|| \to 0.$$

$[X]$ *is usually called a quadratic variation process.*

Remark 2 *Recall, we say that* $(Y_n)_{n\in\mathbb{N}} \to Y$ *in probability if* $\forall \epsilon > 0 \lim_{n\to\infty} \mathbb{P}(\{\omega \in \Omega : |Y_n(\omega) - Y(\omega)| > \epsilon\}) = 0$

Theorem 3 *Let* $(B_t)_{t\geq 0}$ *be a Brownian motion. Then* $[B]_T = T \ \ \forall T \geq 0$ *almost surely.*

Theorem 4 *Let* $(B_t)_{t\geq 0}$ *be a Brownian motion. Then* $t \mapsto B_t(\omega)$ *is not differentiable in each point almost surely.*

2.2 Stochastic Calculus

In this section we provide foundations of stochastic analysis that are necessary to develop mathematical finance theory in continuous time. More precisely, we refresh our memory with crucial ideas of Itô calculus (Itô integral, processes and formula).
Let $T > 0$ and $H = (H_t : t \in [0,T])$ an adapted process. Also, let $B = (B_t : t \in [0,T])$ be a Brownian motion with the filtration $(\mathcal{F} = \mathcal{F}_t)_{t\in[0,T]}$ generated by B. The integral

$$\int_0^T H_t \, dB_t$$

cannot be defined pathwise (i.e. $\forall \omega \in \Omega$) as a Lebesque-Stieltjes integral because Brownian motion B is not of finite variation (Remark 1). Hence we need another approach, which consists of two steps and takes after the procedure of defining expectation of a random variable:

- Define Itô integral for simple integrands.
- Expand the definition to the general case.

Definition 15 (Itô integral of simple process) *Let adapted stochastic process* $H = (H_t : t \in [0,T])$ *(i.e.* $H : \Omega \times [0,T] \to \mathbf{R}$*) be a simple process, in other words let*

$$H_t(\omega) = \sum_{k=0}^{n-1} Z_k(\omega) \mathbb{1}_{[t_k, t_{k+1})}(t)$$

where $n \in \mathbb{N}, 0 = t_0 < t_2 < ... < t_n = T$ *partition of* $[0,T]$ *and* $Z_k, k = 0, 1, 2, ..., n-1$ *is bounded random variable such that it is* $\mathcal{F}_{t_k}$ *- measurable. Itô integral of a simple process* $H = (H_t : t \in [0,T])$ *is a stochastic process* $I = (I_t : t \in [0,T])$ *defined with*

$$I_t = (H \cdot B)_t := \sum_{k=0}^{n-1} Z_k (B_{t_{j+1}\wedge t} - B_{t_j \wedge t}).$$

The process $I = (I_t : t \in [0,T])$ *might be also denoted with* $I_t = \int_0^t H_s \, dB_s$.

Remark 3 *We will denote a family of simple processes on* $[0,T]$ *with* $\mathcal{S}_T$.

Theorem 5 *For Ito integral* I *from the previous definition it holds that:*

- I *is adapted to* $(\mathcal{F}_t)_{t\in[0,T]}$ *and the map* $H \mapsto I = (H \cdot B)$ *on* $\mathcal{S}_T$ *is linear.*
- I *is a martingale.*
- ***(Itô isometry)*** $\mathbb{E}(I_t^2) = \mathbb{E}(\int_0^t H_u^2 \, du)$
- $[I]_t = \int_0^t H_u^2 \, du \quad$ *(i.e.* $[H \cdot B] = H^2 \cdot [B]$*)*

Remark 4 *The third statement from the latest theorem secures that there exists an isometry between the space* $\mathcal{S}_T$ *with inner product* $\langle H, K\rangle = \mathbb{E}(\int_0^T H_t K_t \, dt)$ *and the space* $L^2(\Omega)$ *with inner product* $\langle X, Y\rangle = \mathbb{E}(XY)$.

Now we move to the second step of the construction of Ito integral which is way more complex. To begin with, let us define a space $\mathcal{H}$ of general integrands

$$\mathcal{H} := \{H = (H_t : t \in [0,T]) : H \text{ adapted such that } ||H|| := \mathbb{E}(\int_0^T H_t^2 \, dt) < \infty\}$$

with the inner product $\langle H, K\rangle = \mathbb{E}(\int_0^T H_t K_t\, dt), H, K \in \mathcal{H}$. Note that $\mathcal{S}_T \subseteq \mathcal{H}$. The following lemma states that each $H \in \mathcal{H}$ can be approximated with the sequence $(H^{(n)})_{n\in\mathbb{N}}$ from $\mathcal{S}_T$.

Lemma 1 $\forall H \in \mathcal{H}\ \ \exists (H^{(n)})_{n\in\mathbb{N}}\ \ (H^{(n)} \in \mathcal{S}_T\ \ \forall n \in \mathbb{N})$ *such that*

$$\lim_{n\to\infty} ||H^{(n)} - H|| = \lim_{n\to\infty} \mathbb{E}(\int_0^T |H_t^{(n)} - H_t|^2\, dt) = 0.$$

Now, let $H \in \mathcal{H}$ and let $(H^{(n)})_{n\in\mathbb{N}}$ be a sequence in $\mathcal{S}_T$ from the previous lemma. Then $(H^{(n)})_{n\in\mathbb{N}}$ is a Cauchy sequence in $H \in \mathcal{H}$ because:

$$\lim_{n,m\to\infty} \mathbb{E}(\int_0^T |H_t^{(n)} - H_t^{(m)}|^2\, dt) \le 2\Big(\lim_{n\to\infty} \mathbb{E}(\int_0^T |H_t^{(n)} - H_t|^2\, dt) + \lim_{m\to\infty} \mathbb{E}(\int_0^T |H_t^{(m)} - H_t|^2\, dt)\Big) = 0.$$

Denote $I_t^{(n)} = \int_0^t H_u^{(n)}\, dB_u$. Since Itô isometry holds

$$\mathbb{E}\Big[\int_0^t |H_t^{(n)} - H_t^{(m)}|^2\, dt\Big] = \mathbb{E}\Big[(I_t^{(n)} - I_t^{(m)})^2\Big],$$

we obtain

$$\lim_{n,m\to\infty} \mathbb{E}\Big[(I_t^{(n)} - I_t^{(m)})^2\Big] = 0\ \ \forall t \in [0,T].$$

This implies that a sequence of Itô integrals $(I_t^{(n)})_{n\in\mathbb{N}}$ at time $t \ge 0$ is a Cauchy sequence in Hilbert space $L^2(\Omega, \mathcal{A}, \mathbb{P})\ \ \forall t \in [0,T]$. Since $L^2(\Omega, \mathcal{A}, \mathbb{P})$ is a complete space, it follows that $(I_t^{(n)})_{n\in\mathbb{N}}$ converges in L^2 and the limit is called **Itô integral** of H with respect to Brownian motion B. To wrap up this discussion, we present a usual notation of Itô integral:

$$H_t dB_t = (H \cdot B)_t = I_t = \int_0^t H_u\, dB_u := \lim_{n\to\infty} \int_0^t H_u^{(n)}\, dB_u \tag{2}$$

where *lim* **in the previous equality denotes** L^2 **convergence**.

Theorem 6 *Let $T > 0$ and $H \in \mathcal{H}$. Then, for $(I_t)_{t\in[0,T]}$ defined as in the (2), it holds:*

- *I is adapted to $(\mathcal{F}_t)_{t\in[0,T]}$ and the map $t \mapsto I_t = (H \cdot B)_t$ is continuous on $[0,T]$ almost surely.*
- *linearity: $((aH + bK) \cdot B)_t = a(H \cdot B)_t + b(K \cdot B)_t\ \ \forall H, K \in \mathcal{H},\ \ a, b \in \mathbb{R}$*
- *(**Ito's isometry**) $\mathbb{E}(I_t^2) = \mathbb{E}(\int_0^t H_u^2\, du)$*
- *I is a martingale with respect to filtration $(\mathcal{F}_t)_{t\in[0,T]}$.*

In order to derive Itô formula, we need to introduce Itô process.

Definition 16 (Itô process) *Let $(B_t)_{t\ge0}$ be a Brownian motion. $X = (X_t : t \ge 0)$ is called an Itô process if*

$$X_t = X_0 + \int_0^t H_s\, dB_s + \int_0^t V_s\, ds$$

where $X_0 \in \mathbb{R}$ and H, V adapted processes such that $H \in \mathcal{H}$ and $\forall t \ge 0\ \int_0^t |V_s|\, ds < \infty$ a.s. We also write:

$$\begin{cases} dX_t = V_t dt + H_t dB_t \\ X(t=0) = X_0 \end{cases}$$

Remark 5 *Take note that if X is an Itô process, then its quadratic variation is $[X]_t = \int_0^t H_s^2\, ds$.*

Now we state one of the key results of Itô calculus - celebrated Itô formula.

Theorem 7 (Itô formula) *Let $X_t = X_0 + \int_0^t H_s\, dB_s + \int_0^t V_s\, ds$ be an Itô process and $f : [0,T] \times \mathbb{R} \to \mathbb{R}, f \in \mathcal{C}^{1,2}$ (i.e. partial derivatives f_t, f_x, f_{xx} are continuous). Then $\forall T \ge 0$:*

$$f(T, X_T) = f(0, X_0) + \int_0^T f_t(t, X_t)\, dt + \int_0^T f_x(t, X_t)\, dX_t + \tfrac{1}{2}\int_0^T f_{xx}(t, X_t)\, d[X]_t =$$

$$= f(0, X_0) + \int_0^T f_t(t, X_t)\, dt + \int_0^T f_x(t, X_t) H_t\, dB_t + \int_0^T f_x(t, X_t) V_t\, dt + \tfrac{1}{2}\int_0^T f_{xx}(t, X_t) H_t^2\, dt$$

Equivalently, differential form of the same formula is:

$$df(t, X_t) = f_t(t, X_t) dt + f_x(t, X_t) dX_t + \tfrac{1}{2} f_{xx}(t, X_t) dX_t \cdot dX_t$$

Let us apply Itô formula in the following exercise.

Example 3 *Let $(B_t)_{t\geq 0}$ be a Brownian motion and $f : [0,T] \times \mathbb{R} \to \mathbb{R}$, $f(t,x) := exp(\frac{\lambda^2}{2}t)sin(\lambda x)$ a function. The partial derivatives of f are calculated below:*

$$f_t(t,x) = \tfrac{\lambda^2}{2}\exp(\tfrac{\lambda^2}{2}t)\sin(\lambda x)$$
$$f_x(t,x) = \lambda\exp(\tfrac{\lambda^2}{2}t)\cos(\lambda x)$$
$$f_{xx}(t,x) = -\lambda^2\exp(\tfrac{\lambda^2}{2}t)\sin(\lambda x)$$

Hence, by the previous theorem it follows:

$$f(T,B_T) = f(0,B_0) + \int_0^T \tfrac{\lambda^2}{2}\exp(\tfrac{\lambda^2}{2}t)\sin(\lambda B_t)\,dt + \int_0^T \lambda\exp(\tfrac{\lambda^2}{2}t)\cos(\lambda B_t)\,dB_t + \tfrac{1}{2}\int_0^T -\lambda^2\exp(\tfrac{\lambda^2}{2}t)\sin(\lambda B_t)\,d[B]_t.$$

Notice that $\frac{-\lambda^2}{2}$ can be factored out from the third term and that $d[B]_t = t$ holds by the Theorem 3. Therefore, after cancellation of the first and the third term, and since $B_0 = 0$, we get

$$f(T,B_T) = \lambda\int_0^T \exp(\tfrac{\lambda^2}{2}t)\cos(\lambda B_t)\,dB_t.$$

2.3 Mathematical Finance - Discrete Time

This section copes with a discrete time model of a financial market. In order to define it, we naturally need a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with the filtration $(\mathcal{F}_t)_{t=0}^T$, which plays the role of available information at time $t = 0, 1, 2, ..., T$. We will denote prices of d different assets (stocks) at time t with $S_t^1, S_t^2, ..., S_t^d$. In addition, $S^0 = (S_t^0)_{t=0}^T$ will be the process that refers to a bank account, i.e. for S^0 it holds that

$$S_t^0 = (1+r)^t S_0^0$$

where $r > -1$ is risk free interest rate. Usually, S^0 is considered riskless whereas $S_t^1, S_t^2, ..., S_t^d$ are risky assets. The random vector $S = (S^0, S^1, ..., S^d)$ is adapted (i.e. all processes $S^i, i = 0, 1, ..., d$ are adapted) and S_0 is a constant. Rather than $(S_t)_{t=0}^T$, **discounted price processes**

$$X_t^i := \tfrac{S_t^i}{S_t^0}$$

will be of our greater interest. Consequently, $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t=0}^T, \mathbb{P}, X)$ is called a **market**.

For a short moment, let us assume that there is only one risky asset (stock), i.e. let $d = 1$ and denote its price with $(S_t)_{t=0}^T$. Recall that the value of a call option is

$$C = V_C(T, S_T) = \Psi(S_T) := \begin{cases} S_T - K, & \text{if } S_T > K \\ 0, & \text{if } S_T \leq K \end{cases} \tag{3}$$

where Ψ is a payoff function. Similarly, a put option is define with

$$P = V_P(T, S_T) = \Psi(S_T) := \begin{cases} K - S_T, & \text{if } K > S_T \\ 0, & \text{if } K \leq S_T \end{cases} \tag{4}$$

Recall that **put-call parity** holds

$$C(t, S(t)) - P(t, S(t)) = S(t) - e^{-r(T-t)}K \tag{5}$$

when the put and the call have the same properties. As we have already stated in the introduction of the thesis (chapter 1), major goal of the financial mathematics is to determine the fair price (premium) at time $t = 0$ which the owner of the option should pay to the writer (of the option). In the following pages we provide most significant results of the discrete mathematical finance model.

Definition 17 (Previsible stochastic process) *A stochastic process $X = (X_t)_{t=0}^T$ is previsible if X_t is F_{t-1}-measurable $\forall t = 1, ..., T$ and X_0 is $\mathcal{F}_0$-measurable.*

Definition 18 *A couple of important definitions:*

- *A trading strategy is a predictable $\mathbb{R}$-valued process $H = (H_t)_{t=1}^T = (H_t^0, H_t^1, ..., H_t^d)$, where H_t^i represents the amount of the i-th stock (asset) we hold during time interval $[t-1, t)$.*
- *A process $W = (W_t)_{t=1}^T$ defined with $W_t := H_t \cdot S_t = \sum_{i=0}^d H_i S_i$ is called a **wealth process** (with respect to H).*

- *A process* $Y = (Y_t)_{t=1}^T$ *defined with* $Y_t := H_t \cdot X_t = \sum_{i=0}^d H_i X_i$ *is called a* ***discounted value process*** *(with respect to* H*).*
- *A process* $G = (G_t)_{t=1}^T$ *defined with* $G_t := \sum_{j=1}^t H_j \cdot (X_j - X_{j-1})$ *is called a* ***gain process*** *(with respect to* H*).*
- *Trading strategy* H *is called* ***self-financing*** *if* $H_t \cdot S_t = H_{t+1} \cdot S_t \ \ \forall t \in 0, 1, ..., T-1$.
- *Trading strategy* H *is called an* ***arbitrage*** *(opportunity) if*

$$\mathbb{P}(Y_0 = 0) = 1, \mathbb{P}(Y_T \geq 0) = 1 \text{ and } \mathbb{P}(Y_T > 0) > 0$$

where Y *is the discounted value process with respect to* H*.*

Definition 19 (Equivalent measure) *Let* $\mathbb{P}^*$ *be another probability measure on* $(\Omega, \mathcal{F}, \mathbb{P})$*. We say that* $\mathbb{P}^*$ *is equivalent with* $\mathbb{P}$ *(notation:* $\mathbb{P}^* \approx \mathbb{P}$*) if:*

$$\forall A \in \mathcal{F} \ \ \mathbb{P}^*(A) = 0 \iff \mathbb{P}(A) = 0.$$

As later will turn out, the next definition is crucial in the option pricing procedure.

Definition 20 (Martingale measure) *Probability measure* $\mathbb{P}^*$ *on* $(\Omega, \mathcal{F})$ *is a martingale measure (or risk neutral measure) if* $X = (X_t)_{t=0}^T$ *is a martingale under* $\mathbb{P}^*$*. Additionally,* $\mathbb{P}^*$ *is called an* ***equivalent*** *martingale measure if* $\mathbb{P}^* \approx \mathbb{P}$*.*

The intuitive interpretation of an arbitrage opportunity is the chance to earn money without risk. Therefore the arbitrage is usually considered as an inefficiency of the market because in real life it is very complex to find the opportunity for arbitrage. The following result gives us sufficient and necessary condition for our model to be arbitrage-free (hence realistic). In the statement of the theorem $\mathcal{P}$ denotes the set of all equivalent martingale measures on the market $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t=0}^T, \mathbb{P}, X)$, in other words

$$\mathcal{P} := \{\mathbb{P}^* : \mathbb{P}^* \text{ eq. mtg. measure}\}.$$

Theorem 8 (Fundamental theorem of asset pricing I - discrete time) *There is no arbitrage opportunity in the market model if and only if* $\mathcal{P} \neq \emptyset$*.*

Definition 21 (Contingent claim) *Contingent claim* C *is a* $\mathcal{F}_T$*-measurable random variable such that* $0 \leq C < +\infty \ \ \mathbb{P}$*-a.s.*

As stated at the beginning of the chapter, typical examples of contingent claim are put and call options.

Definition 22 (Attainable/replicable contingent claim) *In a market model that is free of arbitrage, a contingent claim* C *with corresponding discounted claim* $D = \frac{C}{S_T^0}$ *is called attainable/replicable if there exists a self-financing strategy* H *such that*

$$W_T = C \iff Y_T = D$$

where $(W_t)_{t=0}^T$ *and* $(Y_t)_{t=0}^T$ *are wealth process and discounted value process, both with respect to* H*. We say that* H *replicates* C*.*

Now, we want to define a price of a contingent claim D. Intuitively, $\Pi(D) \in \mathbb{R}$ is a price of D if trading of D at price $\Pi(D)$ (at time 0) does **not** give rise to the arbitrage opportunity. Mathematically rigorous statement follows below.

Definition 23 (Price of a contingent claim) *Given a contingent claim* C*, a non-negative* $\Pi_C \in \mathbb{R}$ *is called an arbitrage free price of* C *if there exists an adapted process* $S^{d+1} = (S_t^{d+1})_{t=0}^T$ *such that*

$$S_0^{d+1} = \Pi_C, \ S_t^{d+1} \geq 0, \ X_T^{d+1} = C$$

and such that the enlarged market $(X^0, X^1, ..., X^{d+1})$ *is arbitrage free.* $\Pi(C)$ *denotes the set of all arbitrage free prices of* C*. We will sometimes, instead of* $S^{d+1} = (S_t^{d+1} : t \in [0, T])$*, use following notation:* $(\Pi(C))_{t \in [0,T]}$ *where* $\Pi(C)_0 = \Pi_C$ *and* $\Pi(C)_T = C$*.*

Theorem 9 *Let* C *be replicable contingent claim and assume that* $\mathcal{P} \neq \emptyset$*. Then*

$$\Pi(C) = \{\mathbb{E}_{\mathbb{Q}}(D) : \mathbb{Q} \in \mathcal{P}\}$$

with $D = \frac{C}{S_0^T}$*.*

Theorem 10 *Let market be arbitrage free and let C be a contingent claim. Then*

- *If C is replicable, then $\Pi(C) = \{W_0\}$, where $W_0 = H_1 S_0$ is wealth at time 0 for any strategy H that replicates C.*
- *If C is not replicable, then $\inf \Pi(C) < \sup \Pi(C)$ and $\Pi(C) = \Big(\inf \Pi(C), \sup \Pi(C) \Big)$.*

Definition 24 (Complete market) *An arbitrage free market model is said to be complete if every contingent claim is replicable.*

Note that the completeness of the market is much more complicated and restrictive assumption than no-arbitrage condition. In general, the assumption of completeness is not economically justified.

Theorem 11 (Fundamental theorem of asset pricing II - discrete time) *An arbitrage free market is complete if and only if $\mathcal{P}$ consists of only one element.*

To sum up, let us emphasize once again that in order to determine the price of the option, we only need to know equivalent martingale measure $\mathbb{P}^*$, whereas probability $\mathbb{P}$ plays no role in calculation of the price. Moreover, the price of a contingent claim is the same as the wealth of a hedging strategy that replicates the contingent claim. We finish the chapter with the simple, but famous model of market, due to Cox-Ross-Rubenstein.

Example 4 (CRR model) *In the CRR model, we consider only one risky asset $S = (S_t)_{t=0}^T$ with only two possible returns a, b, such that $-1 < a < b$. In other words, $S_t = (1 + c)S_{t-1}$, where $c \in \{a, b\}$. We define*

$$\Omega := \{-1, 1\}^T = \{\omega = (\omega_1, \omega_2, ..., \omega_T) : \omega_i \in \{-1, 1\}\}$$

and denote projection maps $Y_t : \omega \mapsto \omega_t$ such that

$$\mathbb{P}(Y_t = -1) = p \in (0, 1) \text{ and } \mathbb{P}(Y_t = 1) = 1 - p$$

Additionally, with $\mathcal{F}_t = \sigma(Y_1, Y_2, ..., Y_t)$ and $\mathcal{F}_0 = \{\emptyset, \Omega\}$ define filtration $(\mathcal{F}_t)_{t=0}^T$ and fix probability $\mathbb{P}$. The returns are given with

$$R_t := \begin{cases} a, & \text{if } Y_t = -1 \\ b, & \text{if } Y_t = 1 \end{cases}$$

so for the undiscounted $S = (S_t)_{t=0}^T$ and discounted stock prices $X = (X_t)_{t=0}^T$ holds

$$S_t = S_0 \prod_{i=1}^t (1 + R_t)$$

$$X_t = S_0 \prod_{i=1}^t \frac{(1+R_t)}{(1+r)^t}$$

for some $S_0 > 0$, where r is risk free interest rate as usual.
It can be shown that CRR model is arbitrage free if and only if $a < r < b$. Furthermore, if $a < r < b$, the model is complete.

2.4 Mathematical Finance - Continuous Time

After presenting a discrete time theory of mathematical finance, our current intention is to generalize it to the continuous time environment. We will follow the Black-Scholes-Merton model of a financial market which includes these assumptions:

- All participants of the market have the same access to all relevant information.
- No transaction costs.
- Interest rates are equal for both borrowing and lending money.
- All asset variables are infinitely and perfectly divisible.
- No individual trade affects stock price S.

In addition, note that we will not consider the case which includes dividend payments in the model.
Let us get started with the rigorous mathematical setup of the market model. As in the discrete time case, we fix a

probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with the filtration $(\mathcal{F}_t)_{t\in[0,T]}$ and a Brownian motion $(B_t)_{t\in[0,T]}$. The dynamics of the financial instruments is described with the system

$$\begin{cases} dS_t^0 = rS_t^0 dt \\ dS_t = \mu S_t dt + \sigma S_t dB_t \end{cases} \tag{6}$$

where $S^0 = (S_t^0)_{t\in[0,T]}$ denotes the dynamics of money on the bank account with the risk free interest rate $r > 0$ and $S = (S_t)_{t\in[0,T]}$ denotes adapted process of stock price movement. We say that $(S_t)_{t\in[0,T]}$ follows a **geometric Brownian motion**; μ is called the *drift*, whereas σ is said to be the *volatility* of S. In this paper, we will assume that μ and σ are always constant. The solution of the previous system satisfies:

$$S_t^0 = S_0^0 \exp(rt) \tag{7}$$

$$S_t = S_0 \exp\Big((\mu - \frac{\sigma^2}{2})t + \sigma B_t\Big). \tag{8}$$

Therefore, for the log-returns between times 0 and t, it holds

$$\ln(\tfrac{S_t}{S_0}) \sim \mathcal{N}((\mu - \tfrac{\sigma^2}{2})t,\ \sigma^2 t).$$

We briefly recall basic definitions from the discrete case and give their continuous time analogues:

- **Discounted price process** $X = (X_t)_{t\in[0,T]}$ is adapted process defined as $X_t := e^{-rt}S_t$.
- A **trading strategy (portfolio)** is a pair of adapted stochastic processes (H^0, H) such that $\int_0^T (|H_0^t| + |H_t|^2)\, dt < +\infty$.
- **Wealth process** $W = (W_t)_{t\in[0,T]}$ and **discounted value process** $Y = (Y_t)_{t\in[0,T]}$ with respect to the trading strategy (H^0, H) are defined as

$$W_t := H_t^0 S_t^0 + H_t S_t \quad \text{and} \quad V_t := e^{-rt} W_t.$$

- A strategy (H^0, H) is said to be **self-financing** if it holds

$$W_t - W_0 = \int_0^t H_0^u\, dS_u^0 + \int_0^t H^u\, dS_u \quad \mathbb{P}-\text{almost surely.}$$

Moreover, it can be showed that the process of the wealth change with respect to the portfolio (H^0, H) follows the equation

$$dW_t = rW_t dt + (\mu - r)H_t S_t dt + \sigma H_t S_t dB_t. \tag{9}$$

We interpret the previous expression as the change in wealth depends on three points:

- the first term - mean return rate of the portfolio (H^0, H)
- the second term - risk premium $\mu - r$ for investing in a risky asset (i.e. stock)
- the third term - volatility term depending on the size of investment in the stock.

The following lemma tells us that after discounting the stock price process $S = (S_t : t \in [0, T])$, we end up with decreased mean return $\mu - r$ (initially it was μ). Note that volatility does not change after discounting the stock prices.

Lemma 2 *The discounted price process $X = (X_t, t \geq 0)$ follows the dynamics*

$$dX_t = (\mu - r)X_t dt + \sigma X_t dB_t.$$

The solution of the previous equation is

$$X_t = S_0 \exp\{(\mu - r - \tfrac{\sigma^2}{2})t + \sigma B_t\}.$$

Furthermore, for the discounted value process $Y = (Y_t, t \geq 0)$ it holds

$$dY_t = (\mu - r)H_t X_t dt + \sigma H_t X_t dB_t = H_t dX_t.$$

In order to determine a price of an option (contingent claim), we will again require the notion of the equivalent martingale measure. Before introducing it, we state two theorems that are fundamental for the framework of continuous time financial mathematics.

Theorem 12 (Girsanov) *Let $B = (B_t : t \in [0,T])$ be a Brownian motion on $(\Omega, \mathcal{F}, \mathbb{P})$ and let $(\mathcal{F}_t)_{t\in[0,T]}$ be a natural filtration generated by B. For adapted process $\theta = (\theta_t : t \in [0,T])$ such that $\mathbb{E}(\int_0^T \theta_s^2\, ds) < \infty$ define $(Z_t)_{t\in[0,T]}$*

$$Z_t := \exp(-\int_0^t \theta_u\, dB_u - \tfrac{1}{2}\int_0^t \theta_u^2\, du)\ , \quad Z := Z_T$$

and $(B_t^*)_{t\in[0,T]}$

$$B_t^* := B_t + \int_0^t \theta_u\, du.$$

If $\mathbb{E}(\int_0^T \theta_u^2 Z_u^2\, du) < \infty$, then the stochastic process $B^* = (B_t^* : t \in [0,T])$ is a Brownian motion with respect to probability $\mathbb{P}^*$, where $\mathbb{P}^*(A) := \mathbb{E}[\mathbb{1}_A Z]$.

Theorem 13 (Martingale representation theorem) *Let $B = (B_t : t \in [0,T])$ be a Brownian motion on $(\Omega, \mathcal{F}, \mathbb{P})$ and let $(\mathcal{F}_t)_{t\in[0,T]}$ be a natural filtration generated by B. In addition, assume that $M = (M_t : t \in [0,T])$ is a martingale with respect to $(\mathcal{F}_t)_{t\in[0,T]}$ and probability $\mathbb{P}$. Then there exists an adapted process $\Gamma = (\Gamma_t : t \in [0,T])$ such that*

$$M_t = M_0 + \int_0^t \Gamma_s\, dB_s.$$

Recall that a self-financing trading strategy (H^0, H) is called an arbitrage opportunity if for the discounted value process associated to (H^0, H) holds that $\mathbb{P}(Y_T \geq 0) = 1, \mathbb{P}(Y_T > 0) > 0$ and $\mathbb{P}(Y_0 = 0) = 1$. Now we state a continuous time analogue of the 1st fundamental theorem of asset pricing.

Theorem 14 (Fundamental theorem of asset pricing I - continuous time) *Market model is arbitrage free if and only if there exists equivalent martingale measure.*

The definition of arbitrage free price of a contingent claim (an option) is the same as in the discrete case, i.e. a process $\Pi(C) = (\Pi(C)_t : t \in [0,T])$ is called arbitrage free price of contingent claim C if the extended market model $(S^0, S, \Pi(C))$ is arbitrage free. Also, the definition of attainability/replicability of the contingent claim remains the same as in the discrete time. The following proposition give us instructions on how the price of the option can be calculated. Again, the notion of the equivalent martingale measure plays crucial role in the option pricing.

Proposition 1 *The following statements hold:*

- *If there exists an equivalent martingale measure $\mathbb{Q}$ on the extended market $(S^0, S, \Pi(C))$, then $C \in L^1(\Omega, \mathcal{F}_T, \mathbb{Q})$ and*

$$\Pi(D)_t = e^{-rt}\Pi(C)_t = \mathbb{E}_{\mathbb{Q}}[e^{-rT}C_T|\mathcal{F}_t].$$

- *If C is attainable, then for each equivalent martingale measure $\mathbb{Q}$ and replicating portfolio (H^0, H) holds*

$$Y_t = \mathbb{E}_{\mathbb{Q}}[e^{-rT}C_T|\mathcal{F}_t]$$

  where $Y = (Y_t : t \in [0,T])$ is the discounted value process with respect to (H^0, H).

At the same time, the price $C(t,x)$ of a call option satisfies the famous partial differential equation, so called **Black-Scholes-Merton equation**

$$\frac{\partial C}{\partial t} + \frac{\sigma^2}{2}x^2\frac{\partial^2 C}{\partial x^2} + rx\frac{\partial C}{\partial x} - rC = 0 \tag{10}$$

with the terminal condition $C(T,x) = (x-K)_+$, where t is a time variable and x a stock price variable. Black, Scholes and Merton managed to find an analytic solution of the above-mentioned PDE which is summarized in the following theorem.

Theorem 15 (Black-Scholes-Merton formula) *The solution of the Black-Scholes-Merton equation with the terminal condition $C(T,x) = (x-K)_+$ is*

$$C(t,x) = x\Phi(d_1(T-t,x)) - e^{-r(T-t)}K\Phi(d_2(T-t,x)) \quad t \in [0,T),\ x > 0 \tag{11}$$

where $\Phi(y) = \frac{1}{\sqrt{2\pi}}\displaystyle\int_{-\infty}^{y} e^{-\frac{z^2}{2}}\, dz$ is a distribution function of a standard normal distribution and

$$d_1(\tau, x) = \frac{1}{\sigma\sqrt{\tau}}\Big(\ln\frac{x}{K} + (r + \frac{\sigma^2}{2})\tau\Big),$$
$$d_2(\tau, x) = \frac{1}{\sigma\sqrt{\tau}}\Big(\ln\frac{x}{K} + (r - \frac{\sigma^2}{2})\tau\Big).$$

Using the put-call parity, Black-Scholes-Merton formula can be also derived for a put option:

$$P(t, x) = e^{-r(T-t)}K\Phi(-d_2(T-t, x)) - x\Phi(-d_1(T-t, x)) \quad t \in [0, T),\ x > 0. \tag{12}$$

As in the discrete theory, completeness of $(\Omega, \mathcal{F}, \mathbb{P})$ is characterized by the uniqueness of the equivalent martingale measure.

Theorem 16 (Fundamental theorem of asset pricing II - continuous time) *Let us assume that a market $(\Omega, \mathcal{F}, \mathbb{P})$ is arbitrage free. Then the market is complete if and only if there exists exactly one equivalent martingale measure, i.e.*

$$card(\mathcal{P}) = 1.$$

One question remains open: is Black-Scholes-Merton model arbitrage free? To check this, let us introduce *Sharpe ratio* (or *market price of risk*)

$$\Theta := \frac{\mu - r}{\sigma}$$

Given Sharpe ratio Θ, $X = (X_t : t \in [0, T])$ satisfies

$$dX_t = (\mu - r)X_t dt + \sigma X_t dB_t = \sigma X_t(\Theta dt + dB_t)$$

by Lemma 2. Now we define an adapted process $\Theta_t := \Theta$ and introduce a probability measure $\mathbb{P}^*$ as it is defined in the statement of the Theorem 12. According to the assertion of the Girsanov theorem, $B_t^* = B_t + \Theta t$ is a Brownian motion with respect to $\mathbb{P}^*$. Therefore,

$$dX_t = \sigma X_t(\Theta dt + dB_t) = \sigma X_t dB_t^*$$

which implies that $X = (X_t : t \in [0, T])$ is Itô integral, hence a martingale with respect to $\mathbb{P}^*$. Finally, we can conclude that $\mathbb{P}^*$ is an equivalent martingale measure and BSM model is arbitrage free by the Theorem 14.
When it comes to the completeness of the Black-Scholes-Merton model, it is complete if the filtration $(\mathcal{F}_t)_{t\in[0,T]}$ is generated by Brownian motion $(B_t)_{t\in[0,T]}$.

We have arrived at the end of the Chapter 2 and now we turn to introducing the first of two option pricing methods we will consider - Monte Carlo method.

3 Monte Carlo Method

Before we start to cope with the option pricing using the Monte Carlo method, let us say that we will mainly follow the chapter 3 of the book [1]. The main goal of this chapter is presenting the mathematical background of the method and clarifying the algorithm used for the option pricing. In addition to the basic algorithm, the error decreasing procedure will be provided, followed by a couple of plots that simplify visualization of our topic. Since all necessary explanations for the MC method can be found in this book, we will provide sketches of the derivations rather than writing down all details of the proofs. Note that $V(t, S_t)$ denotes the value of the option at time $t \in [0, T]$.

3.1 Monte Carlo - Overview

Instead of immediately going into mathematical derivations of Monte Carlo method, let us briefly zoom out and try to explain this method by using somewhat loose and less formal language. According to the Proposition 1, the price $\Pi(C)_0$ at time $t = 0$ of a contingent claim C is given by the equality

$$\Pi(C)_0 = e^{-rT}\,\mathbb{E}_Q(C|\mathcal{F}_0) = e^{-rT}\,\mathbb{E}_Q(V_C(T, S_T) \mid (S_t)_{t\in[0,T]} \text{ starting at } S_0)$$

where value $C = V_C(T, S_T) = \Psi(S_T)$ of a contingent claim at time $t = T$ is defined by the payoff function Ψ. The idea of the Monte Carlo method is numerically solving the second stochastic differential equation (SDE) from the system (6) for a huge number of times (usually $N = 10000$) and calculating the payoff function in each iteration. In order to estimate the expectation, it remains to compute the mean payoff and discount it by the factor e^{rT}. Let us now switch to more rigorous expressions.

The most difficult part of the whole procedure is numerical integration of the stochastic differential equation. For the equation[2]

$$dX_t = a(X_t, t)dt + b(X_t, t)dB_t \tag{13}$$

the simplest numerical method is provided below with t-grid $0 = t_0 < t_1 < t_2 < ... < t_M = T$ and equidistant step length $\Delta t = t_{j+1} - t_j = \frac{T}{M} = h$ for some integer M.

Algorithm 1 (Euler discretization of SDE) *Approximations y_j of X_{t_j} for the equation (13) are calculated in the following way:*

- *Start: t_0 and $y_0 = X_0$; choose Δt.*
- *Loop for $j = 0, 1, 2, ..., m - 1$*

$$\begin{aligned} &t_{j+1} = t_j + \Delta t \\ &\Delta W = Z\sqrt{\Delta t} \quad \textit{with } Z \sim \mathcal{N}(0,\,1) \\ &y_{j+1} = y_j + a(y_j, t_j)\Delta t + b(y_j, t_j)\Delta B \end{aligned}$$

In the case of $b = 0$, a discretization error of Euler algorithm is $X_T - y_T^h = O(h)$. In general case, this will not hold for $b \neq 0$.

For analysis of approximation errors, we consider the linear SDE, that we are already familiar with

$$dX_t = \alpha X_t dt + \beta X_t dB_t \tag{14}$$

where α, β are constant coefficients and initial value X_0 for $t = 0$. Given the equation (14), we know that the analytic solution can be computed. It is

$$X_t = X_0 \exp\Big((\alpha - \frac{\beta^2}{2})t + \beta B_t\Big). \tag{15}$$

One technical definition follows.

Definition 25 *We say that a solution X of the equation (13) is **strong** if X is adapted to the natural filtration generated by Brownian motion B. We will call **weak** any solution that is not strong.*

We need the notion of strong solution for the introduction of an absolute error.

[2]One example of this is the equation for discounted stock price process $(X_t)_{t\in[0,T]}$ from Lemma 2 (section 2.4)

Definition 26 (Absolute error) *For a strong solution X_t of the SDE with approximation y_t^h, the absolute error at T is*

$$\epsilon(h) := \mathbb{E}(|X_T - y_T^h|).$$

It can be shown that the order of convergence of Euler method is $O(h^{\frac{1}{2}})$ when dealing with SDEs (that is $b \neq 0$), which is worse than the $O(h)$ in the case of deterministic equation ($b = 0$). There are two assumptions that must hold so that these orders of convergence are true: coefficient functions a and b must satisfy Lipschitz condition and grow at most linearly. Both of these assumptions are satisfied for the geometric Brownian motion (curious reader might check the chapter 3.1. of the book [1] for more details).

To be able to analyze numerical errors more carefully, we introduce two types of convergence, both of them referring to fixed intervals of finite length.

Definition 27 (Strong convergence) *We say that y_T^h converges strongly to X_T with order $\gamma > 0$ if*

$$\epsilon(h) = \mathbb{E}(|X_T - y_T^h|) = O(h^\gamma).$$

Furthermore, y_T^h converges strongly if

$$\lim_{h\to 0} \mathbb{E}(|X_T - y_T^h|) = 0.$$

Note that the algorithm of Euler, for SDEs that satisfy the above assumptions, converges strongly with order $\frac{1}{2}$.

Definition 28 (Weak convergence) *y_T^h converges strongly to X_T with respect to g with order $\beta > 0$ if*

$$\mathbb{E}(g(X_T)) - \mathbb{E}(g(y_T^h)) = O(h^\beta).$$

y_T^h converges weakly to X_T with order β it this holds for all polynomials g.

The Euler method is weakly $O(h^1)$ convergent given the coefficient functions a and b are four times continuously differentiable.

Now we present a sketch of derivations of Milstein algorithm, which shows better order of convergence than Euler procedure. The purpose of all these derivations is to, based on them, formulate the numerical algorithm for the integration of SDE. Full details of transformations might be looked up in the section 3.2 of the book [1].

Consider autonomous stochastic differential equation

$$dX_t = a(X_t)dt + b(X_t)dB_t \tag{16}$$

and denote operators $L^0 f(X_t) = a\frac{\partial}{\partial x} f(X_t) + \frac{1}{2} b^2 \frac{\partial^2}{\partial x^2} f(X_t)$, $L^1 f(X_t) = b\frac{\partial}{\partial x} f(X_t)$. Then by Itô lemma it holds

$$df(X_t) = L^0 f(X_t)dt + L^1 f(X_t)dB_t = \left\{ a\frac{\partial}{\partial x} f(X_t) + \frac{1}{2} b^2 \frac{\partial^2}{\partial x^2} f(X_t) \right\} dt + b\frac{\partial}{\partial x} f(X_t)dB_t$$

so the Itô-Taylor expansion of the equation (16) yields

$$X_t = X_{t_0} + \int_{t_0}^t a(X_s)\, ds + \int_{t_0}^t b(X_s)\, dB_s \tag{17}$$

and we can repeat this procedure for the functions a and b. That gives us

$$X_t = X_{t_0} + \int_{t_0}^t \left\{ a(X_{t_0}) + \int_{t_0}^s L^0 a(X_z)\, dz + \int_{t_0}^s L^1 a(X_z)\, dB_z \right\} ds + \int_{t_0}^t \left\{ b(X_{t_0}) + \int_{t_0}^s L^0 b(X_z)\, dz + \int_{t_0}^s L^1 b(X_z)\, dB_z \right\} dB_s. \tag{18}$$

The previous expression can be slightly modified

$$X_t = X_{t_0} + a(X_{t_0}) \int_{t_0}^t ds + b(X_{t_0}) \int_{t_0}^t dB_s + R \tag{19}$$

where

$$R = \int_{t_0}^t \int_{t_0}^s L^0 a(X_z) dz ds + \int_{t_0}^t \int_{t_0}^s L^1 a(X_z) dB_z ds + \int_{t_0}^t \int_{t_0}^s L^0 b(X_z) dz dB_s + \int_{t_0}^t \int_{t_0}^s L^1 b(X_z) dB_z dB_s. \tag{20}$$

Additionally, L^0 and L^1 are now redefined:

$$L^0 a = aa' + \tfrac{1}{2}b^2 a'', \qquad L^1 a = ba',$$
$$L^0 b = ab' + \tfrac{1}{2}b^2 b'', \qquad L^1 b = bb'.$$

Since the last integral of the remainder R has constant integrand, R can be written as

$$R = L^1 b(X_{t_0}) \int_{t_0}^{t} \int_{s_0}^{s} dB_z \, dB_s + \tilde{R}. \tag{21}$$

Combining (19), (21) and above-mentioned definitions of operators L^0, L^1, we obtain the following result:

$$X_t = X_{t_0} + a(X_{t_0}) \int_{t_0}^{t} ds + b(X_{t_0}) \int_{t_0}^{t} dB_s + b(X_{t_0}) b'(X_{t_0}) \int_{t_0}^{t} \int_{t_0}^{s} dB_z dB_s + \tilde{R}. \tag{22}$$

It can be shown that

$$\int_{t_0}^{t} \int_{t_0}^{s} dB_z \, dB_s = \frac{1}{2}(B_t - B_{t_0})^2 - \frac{1}{2}(t - t_0) = \frac{1}{2}(\Delta B)^2 - \frac{1}{2}\Delta t. \tag{23}$$

If we plug previous equality into the equation (22), we obtain Milstein method.

Algorithm 2 (Milstein) *Milstein algorithm contains of these steps:*

- *Start:* t_0, $y_0 = X_0$, $\Delta t = \frac{T}{M}$.
- *Loop for* $j = 0, 1, 2, ..., M-1$*:*

  $t_{j+1} = t_j + \Delta t$
  Calculate the values $a(y_j, t_j), b(y_j, t_j), b'(y_j, t_j)$,
  $\Delta W = Z\sqrt{\Delta t}$ *with* $Z \sim \mathcal{N}(0, 1)$
  $y_{j+1} = y_j + a(y_j, t_j)\Delta t + b(y_j, t_j)\Delta W + \frac{1}{2} b(y_j, t_j) b'(y_j, t_j)((\Delta W)^2 - \Delta t)$.

Milstein numerical algorithm is strongly convergent with order 1, which is better than Euler's order of convergence. Furthermore, other methods that, for instance, do not use derivatives a', b' (Runge-Kutta method) could also be derived, but we will not pay attention to them here. In fact, in the option pricing process, we will use only Euler (and anayltic[3]) step, which means that, even though they depict the mathematical background of the Milstein algorithm nicely, we will not return to the previous derivations anymore.

3.2 Monte Carlo - Implemention

In this section we will firstly formulate a basic Monte Carlo algorithm for the option pricing, thereafter we will incorporate variance reduction procedure in order to improve accuracy of the initial algorithm. The section will contain several plots that visualize the whole process. Python codes for the option pricing will be provided at the end of the section.

We have already described the core of Monte Carlo method. For a "large" number N of numerical solutions of particular stochastic differenital equation, we are searching for the average of all payoffs. In the case of Black-Scholes-Merton model, samples $(S_T)_{k=1}^N$ are given by solving the geometric Brownian motion SDE under the risk-neutral measure. That is, instead of μ, we set r to be the mean return of the equation (6). At the end of algorithm it remains to discount the calculated mean at the risk free interest rate r in order to get the value of the option at time $t = 0$.

Algorithm 3 (Monte Carlo) *Monte Carlo algorithm consist of these steps:*

- *For* $k = 1, ..., N$ *choose a seed and integrate the SDE of the model for* $0 \le t \le T$ *under the risk neutral measure. More specifically, solve SDE* $dS_t = rS_t dt + \sigma S_t dB_t$. *Let the final results be* $(S_T)_{k=1}^N$.
- *Evaluate the payoff function* Ψ *for* $k = 1, ..., N$ *and obtain values of the option* $(V(S_T, T))_{k=1}^N$*:*

$$(V(S_T, T))_k = \Psi((S_T)_k)$$

- *Calculate the estimation of the risk neutral expectation:*

$$\tilde{E}(V(S_T, T)) = \tfrac{1}{N} \textstyle\sum_{k=1}^{N} (V(S_T, T))_k$$

[3] In the next chapter you can find more about the difference between Euler and analytic step.

- *Discount the previous result at risk free rate:*

$$\tilde{V} = e^{-rT}\tilde{E}(V(S_T, T))$$

and obtain a random variable $\tilde{V}$ such that $\mathbb{E}(\tilde{V}) = V(S_0, 0)$.

Definition 29 (Bias) *Let $\tilde{x}$ be estimate of a true value x. The bias is defined as*

$$bias(\tilde{x}) := \mathbb{E}(\tilde{x}) - x.$$

The estimate $\tilde{x}$ is said to be unbiased if $\mathbb{E}(\tilde{x}) = x$, i.e. $bias(\tilde{x}) = 0$.

Before presenting Python implementation of the algorithm, we should consider ways for improving accuracy of Monte Carlo method. Note that MC algorithm uses only an approximation $\tilde{S_T}$[4] of the theoretical solution (random variable) S_T. This gives rise to an error ϵ, let alone rounding errors caused by limitations of computers

$$\epsilon := e^{-rT}\tilde{E}(\Psi(\tilde{S_T})) - e^{-rT}\,\mathbb{E}(\Psi(S_T)). \tag{24}$$

Equivalently, above equation may be written as

$$e^{rT}\epsilon := \left(\frac{1}{N}\sum_{k=1}^{N}\Psi((\tilde{S}_T)_k) - E(\Psi(\tilde{S}_T))\right) + \left(E(\Psi(\tilde{S}_T)) - \mathbb{E}(\Psi(S_T))\right) \tag{25}$$

where the expression in the first parenthesis is called a **statistical error**, whereas the second parenthesis is referred to as the **bias**. The statistical error can be decreased by the procedure of variance reduction or by enlarging the number of simulations N, which is quite costly because the variance of Monte Carlo method is of order $O(N^{-1})$. On the other hand, the bias can be made arbitrarily small for sufficiently big M. There is a tradeoff between decreasing variance ($N \to +\infty$) and doing same with the bias ($M \to +\infty$). In other words, the statistical error and the bias cannot be diminished at the same time. The tradeoff is given by the **mean square error**

$$MSE(\tilde{x}) := \mathbb{E}[(\tilde{x} - x)^2] = (bias(\tilde{x}))^2 + Var(\tilde{x}). \tag{26}$$

In the case of geometric Brownian motion, we can reduce the bias (even make it disappears!) by using closed form solution of the initial stochastic differenital equation. In other words, the analytic step

$$S_{t+\Delta t} = S_t \exp((\mu - \frac{\sigma^2}{2})\Delta t + \sigma\Delta B) \tag{27}$$

is unbiased, but the Euler step

$$S_{t+\Delta t} = S_t(1 + \mu\Delta t + \sigma\Delta B) \tag{28}$$

is not. In general case, the final goal is to make MSE as small as possible. The following exercise (Ex.3.14. from the book [1]) is very illustrative in this context.

Exercise 1 (Error of biased Monte Carlo) *Assume*

$$MSE = g(N, h) = \alpha_1^2 h^{2\beta} + \tfrac{\alpha_2}{N}$$

as error model of a Monte Carlo simulation with sample size N, based on a discretization of an SDE with stepsize h, where α_1, α_2 are two positive constants.

- *Why is $C(N, h) := \alpha_3 \frac{N}{h}$, for some constant α_3, a reasonable model for the costs of the MC simulation?*

Without loss of generality we can assume that the endpoint of time interval is equal to 1, so set $T = 1$. After plugging this condition into the equation above, we obtain

$$C(N, h) := \alpha_3 NM$$

which can be interpreted as: overall computational costs are product of the number of simulations and the number of subintervals due to time discretization, scaled by some factor α_3. Hence, that equation seems like a reasonable model for the costs of the Monte Carlo method.

- *Minimize function $g(N, h)$ with respect to N, h and the side condition $\alpha_3 \frac{N}{h} = C$, for given budget C.*

[4] In principle, that is the same as y_T^h from the previous part of this chapter.

Let us begin with the substitution $x = N, y = h$ and define function $f(x, y) = \alpha_1^2 y^{2\beta} + \frac{\alpha_2}{x}$ for positive x and y. Using the condition $\alpha_3 \frac{x}{y} = C$, we can reduce the number of arguments of the function f. So we end up with the function $\tilde{f}$:

$$f(x, y) = \alpha_1^2 y^{2\beta} + \frac{\alpha_2}{x} = \alpha_1^2 \alpha_3^{2\beta} \frac{x^{2\beta}}{C^{2\beta}} + \frac{\alpha_2}{x} =: \tilde{f}(x) \tag{29}$$

To find a minimum, we are looking for the solution of the equation $\tilde{f}'(x) = 0$. After computation we obtain $x_0 = D \cdot C^{\frac{2\beta}{2\beta+1}}$ and $y_0 = \alpha_3 D \cdot C^{\frac{-1}{2\beta+1}}$, where $D = \sqrt[2\beta+1]{\frac{\alpha_2}{2\beta \alpha_1^2 \alpha_3^{2\beta}}}$. The second derivative is

$$\tilde{f}''(x) = 2\beta(2\beta - 1)\frac{\alpha_1^2 \alpha_3^{2\beta}}{C^{2\beta}} x^{2\beta-2} + 2\alpha_2 \frac{1}{x^3}.$$

Therefore, f'' is positive if and only if $\beta > \frac{1}{2}$. Hence, (x_0, y_0) is minimum of the function f if and only if $\beta > \frac{1}{2}$.

- *Show that for optimally chosen N, h it holds that $\sqrt{MSE} = \alpha_4 C^{-\frac{\beta}{1+2\beta}}$.*

After plugging solution of the second part in the MSE equation, we end up with

$$MSE = \alpha_1^2 \alpha_3^{2\beta} D^{2\beta} \cdot C^{\frac{-2\beta}{2\beta+1}} + \frac{\alpha_2}{D} \cdot C^{\frac{-2\beta}{2\beta+1}}.$$

Hence, $\sqrt{MSE} = \alpha_4 C^{-\frac{\beta}{2\beta+1}}$, for some constant $\alpha_4 = \sqrt{(\alpha_1^2 \alpha_3^{2\beta} D^{2\beta} + \frac{\alpha_2}{D})}$.

Remark 6 *Let us consider why the form of the function g in the Exercise 1 makes sense. In particular, under the assumption that the bias of the discretization is of order β, that is*

$$bias(\tilde{x}) = \alpha_1 h^{\beta},$$

expression for the first term of the sum should be clear. In addition, the second term of the sum - $\frac{\alpha_2}{N}$ - stems from the fact that the variance of the Monte Carlo method is of order N^{-1}, where N is the sample size (for more details check Chapter 2.4. of the book [1]). Bearing this in mind, we conclude that using Euler method (with $\beta = 1$) gives the exponent $-\frac{1}{3}$, which is definitely worse than the exponent $-\frac{1}{2}$ of an unbiased MC method[5].

When it comes to variance reduction, two procedures (*control variate* method and *antithetic variates*) are presented in the section 3.5.4. of [1]. We will use the latter one. Essentially, simultaneously as calculating V with respect to $Z \sim \mathcal{N}(0, 1)$, we will run a calculation of $\tilde{V}$ with respect to $-Z$. This means that we will just have a "brother" path that looks like a mirror image of the initial. Then we define average variable $V_{AV} = \frac{1}{2}(V + \tilde{V})$ for which it holds

$$Var(V_{AV}) = \tfrac{1}{4} Var(V + \tilde{V}) = \tfrac{1}{4} Var(V) + \tfrac{1}{4} Var(\tilde{V}) + \tfrac{1}{2} Cov(V, \tilde{V}).$$

Since following inequality holds

$$|Cov(V, \tilde{V})| \leq \tfrac{1}{2}(Var(V) + Var(\tilde{V})),$$

by combining last two expressions, we can deduce

$$Var(V_{AV}) \leq \tfrac{1}{2}(Var(V) + Var(\tilde{V}))$$

which means that the antithetic variates procedure should enhance the accuracy of the MC method. Two plots that are provided below suggest the same; the first one depicts Monte Carlo simulations implemented without variance reduction procedure, whereas the second one is the visualization of the Python code which includes antithetic variates method. Note that black dashed line denotes the option price calculated by Black-Scholes-Merton formula.

[5] To analyze improvement of the accuracy of Monte Carlo method more extensively, you can check section 3.5.3. from the book [1].

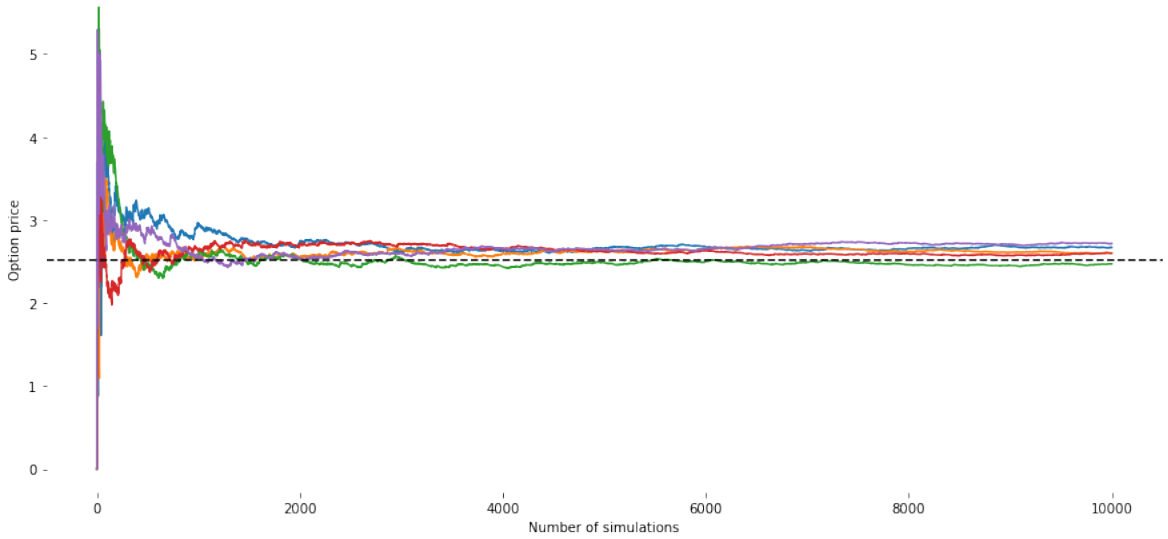

Figure 2: Monte Carlo simulations - without variance reduction

From the pictures, it is quite obvious that simulations in the plot 1 converge slower and have worse accuracy than the ones from the plot 2.

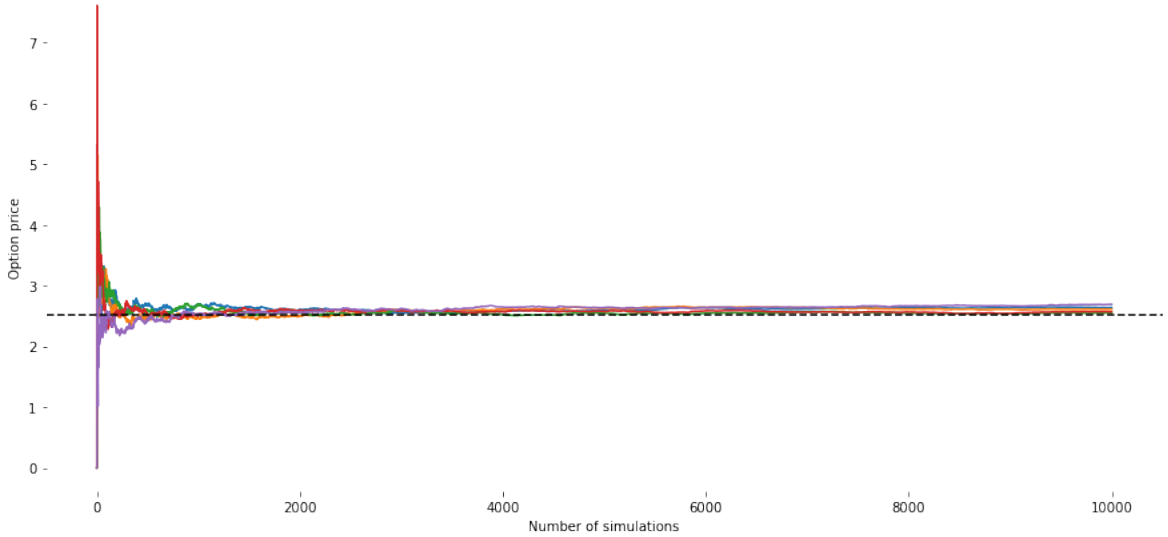

Figure 3: Monte Carlo simulations - with variance reduction

Finally, we present Python codes for Monte Carlo method and plots that illustrate the whole procedure. Take note that we use the analytic step in the algorithm (not Euler step) and apply variance reduction method of *antithetic variates*, both in order to get more accurate results. Horizontal green line symbolizes the theoretical price of an option obtained by evaluating Black Scholes formula.

Firstly, we implement MC method for the call option with parameters: $S_0 = 25, K = 27, T = 1, r = 0.03, \sigma = 0.30$.

```
import numpy as np
from matplotlib import pyplot as plt
from math import sqrt, exp

plt.figure(figsize=(15,7))

```

```
# parameters of a call option
S_0 = 25  # initial stock price
K = 27   # strike price
T = 1   # expiration date
r = 0.03   # risk free interest rate
sigma = 0.30   # volatility

N = 10000   # number of MC simulations
M = 20   # number of intervals due to time discretization
delta_t = T/M

stock_price_1 = np.empty(shape = (N, M + 1), dtype = float)  # array where we store stock prices for each iteration
stock_price_1[:, 0] = S_0   # setting initial stock price
stock_price_2 = np.empty(shape = (N, M + 1), dtype = float)  # array for the variance reduction method
stock_price_2[:, 0] = S_0 # setting initial stock price
payoff = np.empty(N, dtype = float)
payoff_means = np.empty(N, dtype = float)

for i in range(N):

    rng = np.random.default_rng(i + 1)    # different seed in each iteration

    for j in range(M):

        Z = rng.standard_normal()
        delta_W = Z*sqrt(delta_t)
        stock_price_1[i][j + 1] = stock_price_1[i][j]*exp((r - (sigma**2)/2)*delta_t + sigma*delta_W)
        stock_price_2[i][j + 1] = stock_price_2[i][j]*exp((r - (sigma**2)/2)*delta_t - sigma*delta_W)

    if(stock_price_1[i][-1] >= K and stock_price_2[i][-1] >= K):
        payoff[i] = ((stock_price_1[i][-1] - K) + (stock_price_2[i][-1] - K))/2
    elif(stock_price_1[i][-1] >= K and stock_price_2[i][-1] < K):
        payoff[i] = (stock_price_1[i][-1] - K)/2
    elif(stock_price_1[i][-1] < K and stock_price_2[i][-1] >= K):
        payoff[i] = (stock_price_2[i][-1] - K)/2
    else:
        payoff[i] = 0
    if(i == 0):
        payoff_means[0] = payoff[0]
        continue

    payoff_means[i] = np.mean(payoff[0:i])

plt.plot(payoff_means, color = 'red')
plt.axhline(y = 2.49, color = 'limegreen')
plt.annotate(text = 'BSM formula = 2.49', xy = (9000, 2.38), color = 'limegreen')
plt.xlabel('Number of simulations')
plt.ylabel('Option price')
plt.box(False)
price = payoff_means[-1]
print('The price of a call option is', price)
```

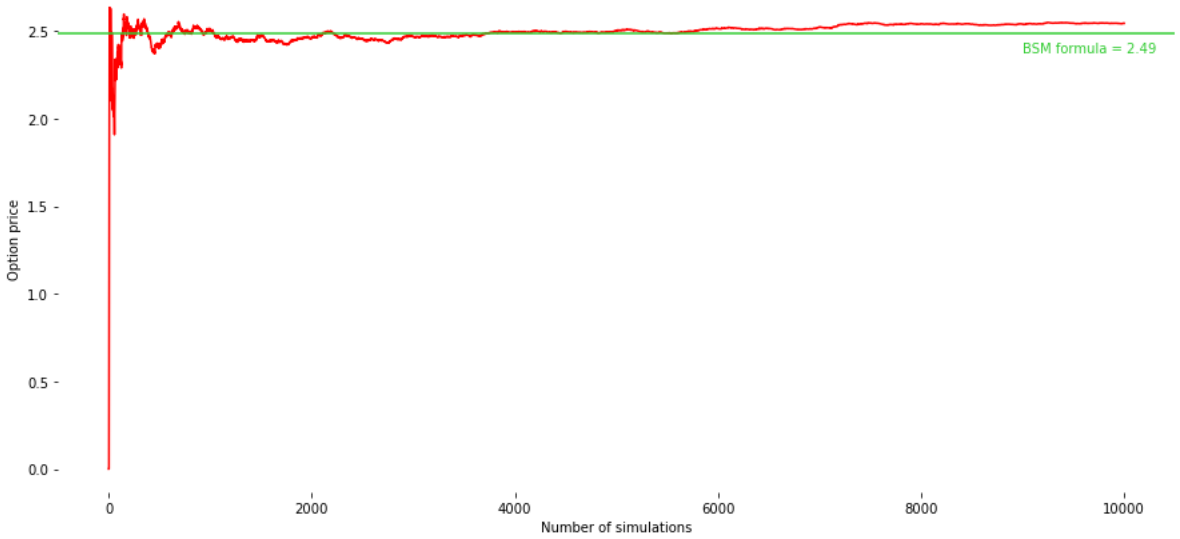

Figure 4: Monte Carlo: call option

Secondly, we consider the case of the put option with parameters: $S_0 = 45, K = 35, T = 1, r = 0.05, \sigma = 0.25$.

```
import numpy as np
from matplotlib import pyplot as plt
from math import sqrt, exp


plt.figure(figsize=(15,7))

# parameters of a put option
S_0 = 45   # initial stock price
K = 35   # strike price
T = 1   # expiration date
r = 0.05   # risk free interest rate
sigma = 0.25   # volatility

N = 10000   # number of MC simulations
M = 20   # number of intervals due to discretization
delta_t = T/M

stock_price_1 = np.empty(shape = (N, M + 1), dtype = float) # array where we store stock prices for each iteration
stock_price_1[:, 0] = S_0  # setting initial stock price
stock_price_2 = np.empty(shape = (N, M + 1), dtype = float)  # array for the variance reduction method
stock_price_2[:, 0] = S_0  # setting initial stock price
payoff = np.empty(N, dtype = float)
payoff_means = np.empty(N, dtype = float)

for i in range(N):

    rng = np.random.default_rng(i + 1)    # different seed in each iteration

    for j in range(M):

        Z = rng.standard_normal()
        delta_W = Z*sqrt(delta_t)
        stock_price_1[i][j + 1] = stock_price_1[i][j]*exp((r - (sigma**2)/2)*delta_t + sigma*delta_W)
        stock_price_2[i][j + 1] = stock_price_2[i][j]*exp((r - (sigma**2)/2)*delta_t - sigma*delta_W)

    if(stock_price_1[i][-1] >= K and stock_price_2[i][-1] >= K):
        payoff[i] = 0
    elif(stock_price_1[i][-1] >= K and stock_price_2[i][-1] < K):
```

```
        payoff[i] = (K - stock_price_2[i][-1])/2
    elif(stock_price_1[i][-1] < K and stock_price_2[i][-1] >= K):
        payoff[i] = (K - stock_price_1[i][-1])/2
    else:
        payoff[i] = ((K - stock_price_1[i][-1]) + (K - stock_price_2[i][-1]))/2
    if(i == 0):
        payoff_means[0] = payoff[0]
        continue

    payoff_means[i] = np.mean(payoff[0:i])

plt.plot(payoff_means, color = 'red')
plt.axhline(y = 0.53, color = 'limegreen')
plt.annotate(text = 'BSM formula = 0.53', xy = (9000, 0.50), color = 'limegreen')
plt.xlabel('Number of simulations')
plt.ylabel('Option price')
plt.box(False)
price = payoff_means[-1]
print('The price of a put option is', price)
```

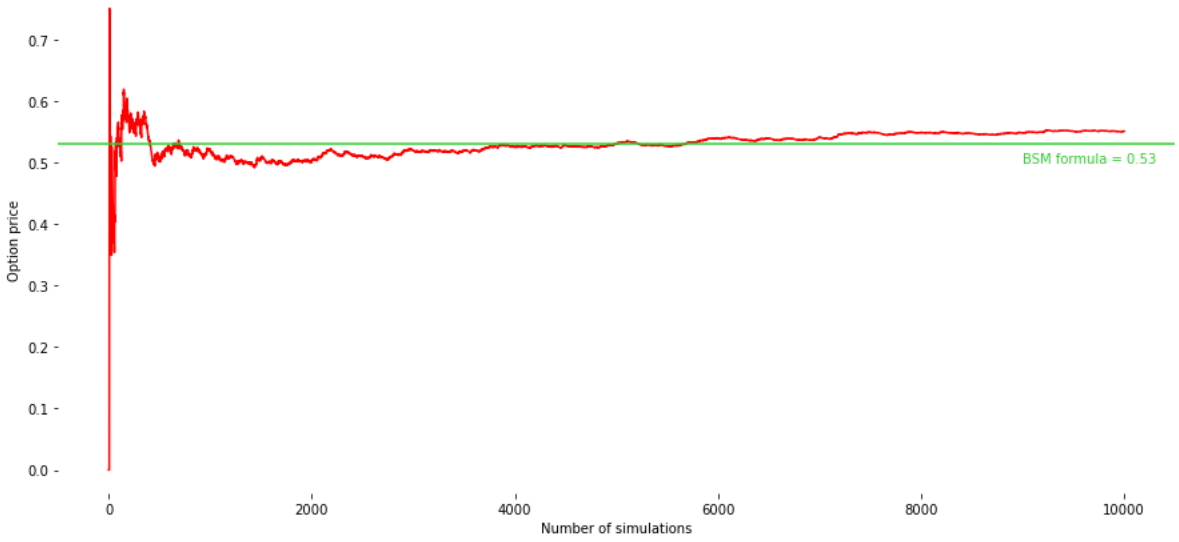

Figure 5: Monte Carlo: put option

4 Finite Difference Method

Chapter 4 will follow the same chapter from the book [1] and will be concentrated on the finite difference (FD) method applied on the Black-Scholes-Merton partial differential equation (PDE). Essentially, we will use a variable substitution in order to approximate solutions of the PDE on the two-dimensional grid. As in the previous chapter, $V(t, S_t)$ denotes the value of the option at time $t \in [0, T]$, all other details will be provided below.

4.1 Finite Difference Method - Overview

To begin with, let us remind ourselves of this famous equation

$$\frac{\partial V}{\partial t} + \frac{\sigma^2}{2} S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0. \tag{30}$$

For constant r and σ, we implement the following substitution[6]:

$$S = Ke^x,\ t = T - \tfrac{2\tau}{\sigma^2},\ q = \tfrac{2r}{\sigma^2}$$
$$V(S,t) = V(Ke^x, T - \tfrac{2\tau}{\sigma^2}) = v(x,\tau)$$
$$v(x,\tau) = K\exp\{-\tfrac{1}{2}(q-1)x - (\tfrac{1}{4}(q-1)^2 + q)\tau\}y(x,\tau).$$

Black Scholes PDE is equivalent to

$$\frac{\partial y}{\partial \tau} = \frac{\partial^2 y}{\partial x^2} \tag{31}$$

for function $y(x,\tau)$ such that $\tau \geq 0,\ x \in \mathbb{R}$.

After this transformation was completed, new bounds for variables are obtained. In particular, from

$$S > 0 \text{ and } 0 \leq t \leq T$$

we have arrived to

$$0 \leq \tau \leq \tfrac{1}{2}\sigma^2 T \text{ and } -\infty < x < \infty.$$

Hence, under the above-mentioned substitutions we get initial conditions:

$$\begin{cases} y(x,0) = \max\{e^{\frac{x}{2}(q+1)} - e^{\frac{x}{2}(q-1)}, 0\}, & \text{for a call} \\ y(x,0) = \max\{e^{\frac{x}{2}(q-1)} - e^{\frac{x}{2}(q+1)}, 0\}, & \text{for a put} \end{cases} \tag{32}$$

Where does the name of this method come from? Let us explain it in next couple of sentences. According to Taylor expansion, for each function $f \in C^2$ it holds that

$$f'(x) = \tfrac{f(x+h)-f(x)}{h} - \tfrac{h}{2}f''(\xi)$$

where ξ is an intermediate number between x and $x + h$. By introducing a one-dimensional grid of equidistant points $x_i, h := x_{i+1} - x_i$ for each i, we discretize $x \in \mathbb{R}$. Since the second derivative f'' is bounded for $f \in C^2$, the last equation might be more conveniently written as

$$f'(x_i) = \frac{f_{i+1} - f_i}{h} + O(h) \tag{33}$$

An analogous procedure can be done for the partial derivatives of $y(x,\tau)$ when a discretization in τ is also imposed. The fraction in the equation (33) approximates the derivative with the finitely small, positive denominator h, hence the name of the method. Also, the $O(h)$ is considered as an error term.

Next, we list two equations with the errors of order $p = 2$:

$$f'(x_i) = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2) \quad (for\ f \in C^3) \tag{34}$$

$$f''(x_i) = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + O(h^2) \quad (for\ f \in C^4). \tag{35}$$

[6]The substitution is slightly different from the one which is in the [1] because we consider no-dividend situation in this thesis.

In order to formulate a reasonable numerical algorithm, the boundaries $x \to -\infty$ and $x \to \infty$ have to be truncated so that we get a rectangular domain for (x, τ). Hence $x \in [a, b]$ where a and b must be chosen so that it does make financial sense:

$$a < \min\{0, \ln(\tfrac{S_0}{K})\} \text{ and } \max\{0, \ln(\tfrac{S_0}{K})\} < b.$$

Before fixing boundary conditions, let us define two-dimensional grid. Pick integers m and v_{max} such that

$$\Delta x = \tfrac{b-a}{m} \text{ and } \Delta\tau = \tfrac{\sigma^2 T}{2v_{max}}.$$

Then notation for the two-dimensional grid is:

$$x_i = i\Delta x \text{ for } i = 0, 1, ..., m$$

$$\tau_v = v\Delta\tau \text{ for } v = 0, 1, ..., v_{max}$$

$$y_{i,v} = y(x_i, \tau_v) \text{ and } w_{i,v} \text{ is an approximation of } y_{i,v}$$

$$\lambda := \tfrac{\Delta\tau}{\Delta x}$$

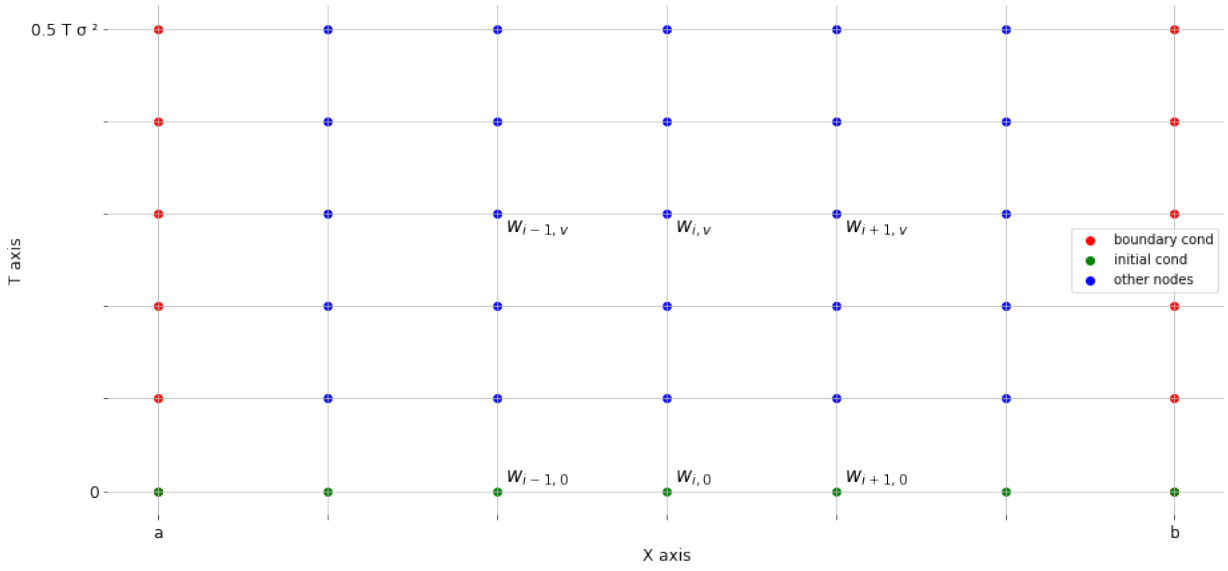

Figure 6: 2-dimensional grid

Our aim is to arrive at the explanation of Crank-Nicolson algorithm. In order to do so, we will have to introduce explicit method and then implicit method, while commenting why those procedures are not sufficiently good to solve our problem.

Let us begin with explicit method. Using the expressions (33) and (35) for the partial derivatives, we get

$$\frac{\partial y_{i,v}}{\partial \tau} := \frac{\partial y(x_i, \tau_v)}{\partial \tau} = \frac{y_{i,v+1} - y_{i,v}}{\Delta\tau} + O(\Delta\tau), \tag{36}$$

$$\frac{\partial^2 y_{i,v}}{\partial x^2} = \frac{y_{i+1,v} - 2y_{i,v} + y_{i-1,v}}{\Delta x^2} + O(\Delta x^2). \tag{37}$$

After discarding the O-error terms and introducing the notation w for the approximations of the initial function y, we end up with

$$w_{i,v+1} = \lambda w_{i-1,v} + (1 - 2\lambda)w_{i,v} + \lambda w_{i+1,v} \tag{38}$$

where $\lambda := \frac{\Delta\tau}{\Delta x^2}$.

For index $v = 0$, $w_{i,0}$ are calculated by evaluating the conditions (32) for each $i = 0, 1, ..., m$. According to (38), evaluation of $w_{i,v}$ for general v is done by time levels: after carrying out computation for all nodes with time index v, we proceed to the next time level $v + 1$ and repeat the procedure of calculation. Since expression (38) gives

an explicit formula for calculation of $w_{i,v+1}$ for each $i = 0, 1, ..., m$, this procedure is called **explicit method** or **forward-difference method**.

Approximations $w_{0,v}$ and $w_{m,v}$, for $v = 1, ..., v_{max}$, are defined by boundary conditions, which we set to zero for now. So $w_{0,v} = w_{m,v} = 0$ for every $v = 1, ..., v_{max}$.

Using vector notation $w^{(v)} = (w_{1,v}, ..., w_{m-1,v})$, the explicit method can be written as

$$w^{(v+1)} = A_{ex} w^{(v)}, \qquad v = 0, 1, ..., v_{max} - 1$$

where A_{ex} is $(m-1) \times (m-1)$ tridiagonal matrix such that

$$A_{ex} = \begin{pmatrix} 1-2\lambda & \lambda & 0 & . & . & 0 \\ \lambda & 1-2\lambda & \lambda & & & . \\ 0 & & . & & & . \\ . & & & . & & 0 \\ . & & & & . & \lambda \\ 0 & . & . & 0 & \lambda & 1-2\lambda \end{pmatrix}.$$

The main weakness of the explicit method is its numerical instability, i.e. possibility of the occurence of significant computational errors. This is nicely illustrated in the Example 4.1. in our main source [1], which we will explain in the following sentences. Let us consider an iteration

$$w^{(v+1)} = A_{ex} w^{(v)} + d^{(v)}$$

where $d^{(v)}$ denotes vector of boundary conditions (that are temporarily set to zero). We have to make a distinction between theoretical value $w^{(v)}$ (which is free of rounding errors) and numerical computation $\bar{w}^{(v)}$ calculated by computer. Hence, the vector of computational errors is given by

$$e^{(v)} = \bar{w}^{(v)} - w^{(v)}$$

for $v \geq 0$. Then for $\bar{w}^{(v)}$ holds

$$\bar{w}^{(v+1)} = A_{ex} \bar{w}^{(v)} + d^{(v)} + r^{(v+1)}$$

where the vector $r^{(v+1)}$ denotes the rounding errors which pop up while calculating $A_{ex}\bar{w}^{(v)} + d^{(v)}$. Consider index $v^* = 0$ and, for simplicity, set $r^{(v)} = 0$ for $v > 1$. In other words, we want to check which effect initial rounding error $e^{(0)}$ has on the algorithm. Since $\bar{w}^{(v+1)} = A_{ex}\bar{w}^{(v)} + d^{(v)}$ for $v \geq 1$, the evolution of the rounding error is given by

$$A_{ex} e^{(v)} = A_{ex}\bar{w}^{(v)} - A_{ex} w^{(v)} = \bar{w}^{(v+1)} - w^{(v+1)} = e^{(v+1)}.$$

This results in

$$e^{(v)} = A_{ex}^{(v)} e^{(0)}, v \geq 1. \tag{39}$$

The rounding errors should be damped if we want to have a stable method. Therefore

$$\lim_{v \to \infty} A_{ij}^{(v)} = 0$$

should hold for every pair of indices i, j.

Lemma 3 *Let A be $N \times N$ matrix. Then*

$$\rho(A) < 1 \iff A^v z \to 0 \text{ for all } z \text{ and } v \to \infty \iff lim_{v\to\infty}\{(A^v)_{i,j}\} = 0$$

where $\rho(A) := \max_{k=1,...,m}\{\mu_k : \mu_k$ is an eigenvalue of matrix $A\}$, $m \in \mathbb{N}$.

Lemma 4 *Let*

$$G = \begin{pmatrix} \alpha & \beta & 0 & . & . & 0 \\ \gamma & \alpha & \beta & & & . \\ 0 & \gamma & . & & & . \\ . & & & . & & 0 \\ . & & & & . & \beta \\ 0 & . & . & 0 & \gamma & \alpha \end{pmatrix}$$

be $N \times N$ matrix. The eigenvalues μ_k^G are

$$\mu_k^G = \alpha + 2\beta\sqrt{\tfrac{\gamma}{\beta}}\cos\tfrac{k\pi}{N+1}, \quad k = 1, ..., N$$

Using these two lemmas, we derive that the key condition for the stability of the explicit method is

$$0 < \lambda \leq \tfrac{1}{2}.$$

If it does not hold, the explicit method is not stable. Consequently, one has to be very cautious when choosing parameters m and v_{max} for the explicit method because they depend on each other. Given this result, we should look for an algorithm that is unconditionally stable. Hence, we will now consider an implicit method.

We have already tried to approximate time derivative (36) with the forward difference, but now we want to do the same with the backward difference

$$\frac{\partial y_{i,v}}{\partial \tau} = \frac{y_{i,v} - y_{i,v-1}}{\Delta\tau} + O(\Delta\tau). \tag{40}$$

This yields

$$-\lambda w_{i+1,v} + (1+2\lambda)w_{i,v} - \lambda w_{i-1,v} = \lambda w_{i,v-1}. \tag{41}$$

Equivalently, vector notation reads

$$A_{im} w^{(v+1)} = w^{(v)}, \qquad v = 0, 1, ..., v_{max} - 1 \tag{42}$$

with the tridiagonal matrix

$$A_{im} = \begin{pmatrix} 1+2\lambda & -\lambda & 0 & . & . & 0 \\ -\lambda & 1+2\lambda & -\lambda & & & . \\ 0 & & . & & & . \\ . & & & . & & 0 \\ . & & & & . & -\lambda \\ 0 & . & . & 0 & -\lambda & 1+2\lambda \end{pmatrix}.$$

We call this procedure **fully implicit** or **backward-difference method** and it is unconditionally stable for every $\Delta\tau > 0$. Even though the implicit method is stable, we want that the time discretization of our algorithm has the better order - $O(\tau^2)$.

As Crank and Nicolson suggested, we will combine explicit and implicit method in order to get more accurate algorithm. We express forward difference fraction for index v

$$\frac{w_{i,v+1} - w_{i,v}}{\Delta\tau} = \frac{w_{i+1,v} - 2w_{i,v} + w_{i-1,v}}{\Delta x^2} \tag{43}$$

and backward difference for index $v+1$

$$\frac{w_{i,v+1} - w_{i,v}}{\Delta\tau} = \frac{w_{i+1,v+1} - 2w_{i,v+1} + w_{i-1,v+1}}{\Delta x^2}. \tag{44}$$

Adding up last two equations yields

$$-\frac{\lambda}{2}w_{i-1,v+1} + (1+\lambda)w_{i,v+1} - \frac{\lambda}{2}w_{i+1,v+1} = \frac{\lambda}{2}w_{i-1,v} + (1-\lambda)w_{i,v} + \frac{\lambda}{2}w_{i+1,v}. \tag{45}$$

Using vector notation $w^{(v)} = (w_{1,v}, ..., w_{m-1,v})$, expression from above can be written as

$$Aw^{(v+1)} = Bw^{(v)} + d^{(v)}$$

where $d^{(v)} = 0$, A and B are $(m-1) \times (m-1)$ tridiagonal matrices such that

$$A = \begin{pmatrix} 1+\lambda & -\frac{\lambda}{2} & 0 & . & . & 0 \\ -\frac{\lambda}{2} & 1+\lambda & -\frac{\lambda}{2} & & & . \\ 0 & & . & & & . \\ . & & & . & & 0 \\ . & & & & . & -\frac{\lambda}{2} \\ 0 & . & . & 0 & -\frac{\lambda}{2} & 1+\lambda \end{pmatrix}, \qquad B = \begin{pmatrix} 1-\lambda & \frac{\lambda}{2} & 0 & . & . & 0 \\ \frac{\lambda}{2} & 1-\lambda & \frac{\lambda}{2} & & & . \\ 0 & & . & & & . \\ . & & & . & & 0 \\ . & & & & . & \frac{\lambda}{2} \\ 0 & . & . & 0 & \frac{\lambda}{2} & 1-\lambda \end{pmatrix}.$$

Take notice that boundary conditions are still trivial and equal to zero. Let us change that now. Intuitively, it should be clear that for values V_C and V_P of a call respective put it holds

$$V_C(S,t) = 0 \text{ for } S = 0,$$
$$V_P(S,t) \to 0 \text{ for } S \to \infty.$$

In order to get boundary conditions for a call when $S \to \infty$ and for a put when $S = 0$, the put-call parity as well as transformations (such as one from the beginning of this chapter) are utilized. We skip the derivations[7] here and present the boundary conditions for European options:

$$y(x,\tau) := \begin{cases} r_1(x,\tau), & \text{if } x \to -\infty \\ r_2(x,\tau), & \text{if } x \to \infty \end{cases} \tag{46}$$

with

$$r_1(x,\tau) := \begin{cases} 0, & \text{for a call} \\ \exp\{\frac{1}{2}(q-1)x + \frac{1}{4}(q-1)^2\tau\}, & \text{for a put} \end{cases} \tag{47}$$

$$r_2(x,\tau) := \begin{cases} 0, & \text{for a put} \\ \exp\{\frac{1}{2}(q+1)x + \frac{1}{4}(q+1)^2\tau\}, & \text{for a call} \end{cases} \tag{48}$$

Of course, since computers cannot process limit behaviour $|x| \to \infty$, we have to impose truncation to the finite interval $[a,b]$. Therefore, it holds

$$w_{0,v} = r_1(a,\tau_v) \text{ and } w_{m,v} = r_2(b,\tau_v) \text{ for } v = 0,...,v_{max}.$$

When we plug those conditions into to the equation (45), we get

- for $i = 1$

$$-\frac{\lambda}{2}r_1(a,\tau_{v+1}) + (1+\lambda)w_{i,v+1} - \frac{\lambda}{2}w_{i+1,v+1} = \frac{\lambda}{2}r_1(a,\tau_v) + (1-\lambda)w_{i,v} + \frac{\lambda}{2}w_{i+1,v}; \tag{49}$$

- for $i = m-1$

$$-\frac{\lambda}{2}r_2(b,\tau_{v+1}) + (1+\lambda)w_{i,v+1} - \frac{\lambda}{2}w_{i+1,v+1} = \frac{\lambda}{2}r_2(b,\tau_v) + (1-\lambda)w_{i,v} + \frac{\lambda}{2}w_{i+1,v}. \tag{50}$$

Eventually, we conclude this discussion about the boundary conditions with the vector notation

$$d^{(v)} = \begin{pmatrix} r_1(a,\tau_{v+1}) + r_1(a,\tau_v) \\ 0 \\ . \\ . \\ . \\ 0 \\ r_2(b,\tau_{v+1}) + r_2(b,\tau_v) \end{pmatrix}.$$

Finally, we present the summary of the Crank-Nicolson algorithm.

Algorithm 4 (Crank-Nicolson) *CN algorithm consists of these steps:*

- *Start: Choose m, v_{max} and calculate $\Delta x, \Delta\tau$; compute initial conditions $w_i^{(0)}, i = 0,1,...,m$ from (32);*
- *Calculate LU-decomposition of A*
- *Loop: for $v = 0,1,...,v_{max}-1$*

  Calculate $c = Bw^{(v)} + d^{(v)}$; then solve $Ax = c$ using LU composition
  Compute $w^{(v)} = x$

Note that, after we have computed approximation $w_{i,v}$ of $y(x_i,\tau_i)$, it remains to apply transformations once again in order to obtain $V(S_0,0)$. To wrap up this section, let us state a theorem that secures stability of the Crank-Nicolson procedure.

[7] The procedure of defining boundary conditions is described in the section 4.4 of the book [1]

Theorem 17 *Concerning Crank-Nicolson method, it holds:*

- *For $y \in C^4$ the order of the method is $O(\Delta\tau^2) + O(\Delta x^2)$*
- *For each v a linear system of a simple tridiagonal structure must be solved.*
- *Stability holds for all $\Delta\tau > 0$.*

We will not prove the theorem completely, but we will just give a quick hint about how the first statement might be shown. Hence, given

$$\epsilon = \frac{y_{i,v+1} - y_{i,v}}{\Delta\tau} - \frac{1}{2}(\delta_{xx} y_{i,v} + \delta_{xx} y_{i,v+1}) \tag{51}$$

as in the proof from the book [1], using the Taylor expansion in time (up to the second order), we obtain:

$$\epsilon = \frac{\partial y_{i,v}}{\partial\tau} + \frac{\Delta\tau}{2}\frac{\partial^2 y_{i,v}}{\partial\tau^2} + O(\Delta\tau^2) - \frac{1}{2}\left(\frac{\partial^2 y_{i,v}}{\partial x^2} + \frac{\Delta x^2}{12}\frac{\partial^4 y_{i,v}}{\partial x^4} + O(\Delta x^4) + \frac{\partial^2 y_{i,v}}{\partial x^2} + \Delta\tau\frac{\partial}{\partial\tau}\left(\frac{\partial y_{i,v}}{\partial x^2}\right) + O(\Delta\tau^2)\right). \tag{52}$$

Now, after applying the equation $\frac{\partial y}{\partial\tau} = \frac{\partial^2 y}{\partial x^2}$, above expression boils down to

$$\epsilon = O(\Delta\tau^2) + O(\Delta x^2) \tag{53}$$

what we wanted to check. Note that the proof of all three statements is given in the book [1].

4.2 Finite Difference Method - Implementation

After we have briefly presented finite difference method of Crank and Nicolson, at the moment our goal is to implement the algorithm in Python. Firstly, we consider the case of the call option with same parameters as in the Monte Carlo case: $S_0 = 25, K = 27, T = 1, r = 0.03, \sigma = 0.30$.

```
import numpy as np
from math import sqrt, log, exp, pi
import scipy.linalg as linalg

# parameters of a call option
S = 25     # initial stock price
K = 27     # strike price
T = 1      # expiration date
r = 0.03   # risk free interest rate
sigma = 0.30    # volatility

# variables that we need for substitution
q = (2*r)/sigma**2
v_max = 6  # number of time intervals
m = 300  # number of space intervals
a = -5   # truncation of x axis
b = 5    # truncation of x axis
w = np.empty(shape = (v_max + 1, m - 1), dtype = float) # 2-dim array in which we store
    approximations of y

# introducing substitution: 0 <= tau <= (T*sigma**2)/2
delta_x = (b - a)/m
delta_tau = (T*sigma**2/2)/v_max
parameter_lambda =  delta_tau/delta_x**2

# a function that evaluates initial conditions for a call/put option
def y(x, option):
    if(option == 'call'):
        result = exp(x*(q + 1)/2) - exp(x*(q - 1)/2)
        if(result < 0): return 0
        return result
    if(option == 'put'):
        result = exp(x*(q - 1)/2) - exp(x*(q + 1)/2)
        if(result < 0): return 0
        return result

# a function that evaluates boundary conditions for a call/put option when x = a
def r_1(x, tau, option):
    if(option == 'call'): return 0
```

```
    if(option == 'put'):
        result = exp(x*(q - 1)/2 + ((q - 1)**2)*tau/4)
        return result

# a function that evaluates boundary conditions for a call/put option when x = b
def r_2(x, tau, option):
    if(option == 'put'): return 0
    if(option == 'call'):
        result = exp(x*(q + 1)/2 + ((q + 1)**2)*tau/4)
        return result

# evaluating initial conditions
for i in range(m - 1):
    w[0][i] = y(a + (i + 1) * delta_x, 'call')

# evaluating boundary conditions
d = np.zeros(shape = (v_max + 1, m - 1), dtype = float)  # array in which we store evaluated
    boundary conditions
for v in range(v_max + 1):
    time_index = v * delta_tau
    first = r_1(a, time_index, 'call') + r_1(a, time_index + delta_tau, 'call')
    last = r_2(b, time_index, 'call') + r_2(b, time_index + delta_tau, 'call')
    d[v, 0] = first * parameter_lambda / 2
    d[v, -1] = last * parameter_lambda / 2

# computing matrices A and B
A = (1 + parameter_lambda)*np.identity(m - 1) + (parameter_lambda/(-2))*np.eye(m - 1, k = 1) + (
    parameter_lambda/(-2))*np.eye(m - 1, k = -1)
B = (1 - parameter_lambda)*np.identity(m - 1) + (parameter_lambda/2)*np.eye(m - 1, k = 1) + (
    parameter_lambda/2)*np.eye(m - 1, k = -1)

# computing LU factorization of A
LU = linalg.lu_factor(A)

# computing vector w for each time index v
for v in range(v_max):
    c = B @ w[v, :] + d[v, :]
    y = linalg.lu_solve(LU, c)
    w[v + 1, :] = y

# a function that returns us the x-axis index which we have to consider to for our initial stock
    price S
def location(a, b, m, x):
    delta_x = (b - a)/m
    if(a <= x and x <= a + delta_x): return 0
    if(b >= x and x >= b - delta_x): return -1
    for i in range(m - 1):
        if(x >= a + (i + 1)*delta_x and x <= a + (i + 2)*delta_x): return i + 1

x = log(S/K)
i = location(a, b, m, x)
x_index = a + i*delta_x
if(i == -1): x_index = b - delta_x
y_result = w[-1, i]
print('The price of a call option is', K*exp((q - 1)*x/(-2) - ((q - 1)**2/4 + q)*(T*sigma**2)/2)*
    y_result)
```

Next, we will present Python code for a put option, again with same parameters as in the MC case: $S_0 = 45, K = 35, T = 1, r = 0.05, \sigma = 0.25$.

```
import numpy as np
from math import sqrt, log, exp, pi
import scipy.linalg as linalg

# parameters of a put option
S = 45     # initial stock price
K = 35     # stock price
T = 1      # expiration date
r = 0.05   # risk free interest rate
sigma = 0.25 # volatility

```

```
# variables that we need for substitution
q = (2*r)/sigma**2
v_max = 10  # number of time intervals
m = 500  # # number of space intervals
a = -5   # truncation of x axis
b = 5    # truncation of x axis
w = np.empty(shape = (v_max + 1, m - 1), dtype = float) # 2-dim array in which we store approximations of y

# introducing substitution: 0 <= tau <= (T*sigma**2)/2
delta_x = (b - a)/m
delta_tau = (T*sigma**2/2)/v_max
parameter_lambda =  delta_tau/delta_x**2

# a function that evaluates initial conditions for a call/put option
def y(x, option):
    if(option == 'call'):
        result = exp(x*(q + 1)/2) - exp(x*(q - 1)/2)
        if(result < 0): return 0
        return result
    if(option == 'put'):
        result = exp(x*(q - 1)/2) - exp(x*(q + 1)/2)
        if(result < 0): return 0
        return result

# a function that evaluates boundary conditions for a call/put option when x = a
def r_1(x, tau, option):
    if(option == 'call'): return 0
    if(option == 'put'):
        result = exp(x*(q - 1)/2 + ((q - 1)**2)*tau/4)
        return result

# a function that evaluates boundary conditions for a call/put option when x = b
def r_2(x, tau, option):
    if(option == 'put'): return 0
    if(option == 'call'):
        result = exp(x*(q + 1)/2 + ((q + 1)**2)*tau/4)
        return result

# evaluating initial conditions
for i in range(m - 1):
    w[0][i] = y(a + (i + 1) * delta_x, 'put')

# evaluating boundary conditions
d = np.zeros(shape = (v_max + 1, m - 1), dtype = float)  # array in which we store evaluated boundary conditions
for v in range(v_max + 1):
    time_index = v * delta_tau
    first = r_1(a, time_index, 'put') + r_1(a, time_index + delta_tau, 'put')
    last = r_2(b, time_index, 'put') + r_2(b, time_index + delta_tau, 'put')
    d[v, 0] = first * parameter_lambda / 2
    d[v, -1] = last * parameter_lambda / 2

# computing matrices A and B
A = (1 + parameter_lambda)*np.identity(m - 1) + (parameter_lambda/(-2))*np.eye(m - 1, k = 1) + (parameter_lambda/(-2))*np.eye(m - 1, k = -1)
B = (1 - parameter_lambda)*np.identity(m - 1) + (parameter_lambda/2)*np.eye(m - 1, k = 1) + (parameter_lambda/2)*np.eye(m - 1, k = -1)

# computing LU factorization of A
LU = linalg.lu_factor(A)

# computing vector w for each time index v
for v in range(v_max):
    c = B @ w[v, :] + d[v, :]
    y = linalg.lu_solve(LU, c)
    w[v + 1, :] = y

# a function that returns us the x-axis index which we have to consider to for our initial stock price S
def location(a, b, m, x):
```

```
    delta_x = (b - a)/m
    if(a <= x and x <= a + delta_x): return 0
    if(b >= x and x >= b - delta_x): return -1
    for i in range(m - 1):
        if(x >= a + (i + 1)*delta_x and x <= a + (i + 2)*delta_x): return i + 1

x = log(S/K)
i = location(a, b, m, x)
x_index = a + i*delta_x
if(i == -1): x_index = b - delta_x
y_result = w[-1, i]
print('The price of a put option is', K*exp((q - 1)*x/(-2) - ((q - 1)**2/4 + q)*(T*sigma**2)/2)*
    y_result)
```

5 Analysis of Results: Monte Carlo vs Finite Difference Method

In this section we compare performances of two pricing methods we have applied in cases of European put and call option. As a comparison benchmark, we will use theoretical price of the option, calculated by Black-Scholes-Merton formula. Python code that evaluates the formula is given below:

```
from math import sqrt, exp, log
from scipy.stats import norm

# evaluating Black Scholes formula

def d1(S,K,T,r,sigma):
    return(log(S/K) + (r + sigma**2/2.)*T) / (sigma*sqrt(T))

def d2(S,K,T,r,sigma):
    return(log(S/K) + (r - sigma**2/2.)*T) / (sigma*sqrt(T))

def BSM_formula_call(S,K,T,r,sigma):
    return S*norm.cdf(d1(S,K,T,r,sigma)) - K*exp(-r*T)*norm.cdf(d2(S,K,T,r,sigma))

def BSM_formula_put(S,K,T,r,sigma):
    return K*exp(-r*T)*norm.cdf(-d2(S,K,T,r,sigma)) - S*norm.cdf(-d1(S,K,T,r,sigma))

print('call: ', BSM_formula_call(25, 27, 1, 0.03, 0.30))
print('put: ', BSM_formula_put(45, 35, 1, 0.05, 0.25))
```

We will analyze results of two algorithms for Monte Carlo method (with unbiased analytic step (27) as well as with biased Euler step (28)) and one algorithm for finite difference method. Antithetic variates procedure was part of both MC algorithms in order to decrease variance.

5.1 Call Option

Let parameters of a call be $S_0 = 25, K = 27, T = 1, r = 0.03, \sigma = 0.30$. Its price obtained by BSM formula is 2.488568. In the following two sections we will compare performances of the two methods in pricing this call.

5.1.1 Monte Carlo for a Call

To begin with, let us consider figures obtained by Monte Carlo algorithms. The first table provides the results of MC method with unbiased analytic step, depending on the different number of intervals due to time-discretization.

Time intervals	MC - analytic	Absolute error
10	2.558831	0.070263
20	2.544556	0.055988
30	2.579558	0.09099
40	2.563111	0.074543
50	2.56244	0.073872
60	2.581716	0.093149
70	2.551618	0.06305
80	2.551113	0.062545
90	2.567861	0.079293
BSM formula	2.488568	

Figure 7: MC method with analytic step: time intervals increase

We see that the absolute errors oscillate around 0.07. Also, it can be noticed that the accuracy of the method did not improve when the number of time intervals rose. This is somehow expected because we use unbiased analytic step in the pricing algorithm, for which the mean square error (MSE) does not depend on the number of time-discretization subintervals.

Next, we present the table of MC algorithm with Euler step. The oscillation of the errors is still noticeable but the results are slightly worse than in the previous case. The reason for poorer accuracy is the fact that we applied Euler step which is biased and makes MSE greater than in the previous case.

Time intervals	MC - Euler	Absolute error
10	2.565805	0.077237
20	2.554109	0.065541
30	2.632046	0.143479
40	2.588268	0.0997
50	2.595841	0.107273
60	2.608941	0.120374
70	2.575907	0.087339
80	2.563939	0.075371
90	2.584016	0.095448
BSM formula	2.488568	

Figure 8: MC method with Euler step: time intervals increase

The following plot compares magnitudes of errors of two different MC algorithms (with Euler or analytic step). The oscillation is probably a consequence of the volatile trajectories of simulated stock price. When it comes to Euler method, a decreasing trend with respect to soaring number of time intervals is visible, which is logical if we know that the bias gets smaller when the number M of intervals due to time-discretization jumps.

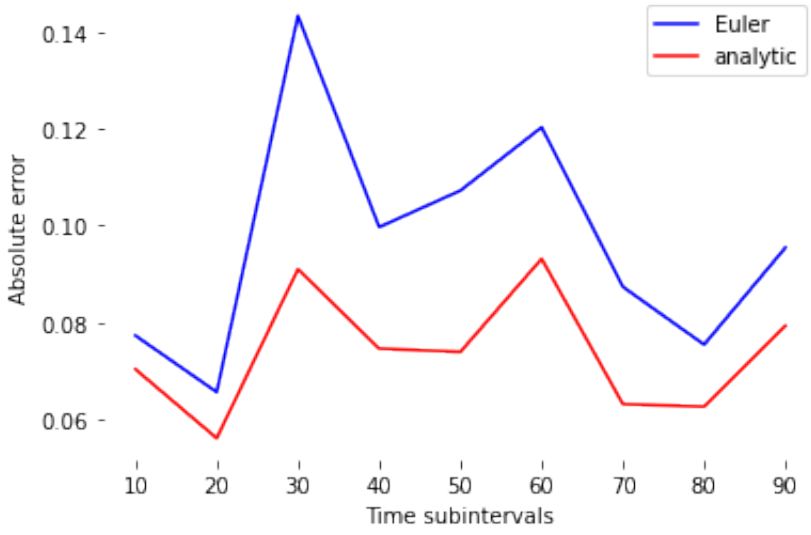

Figure 9: Absolute errors - Euler vs analytic step

A new table, that depicts the results of MC method (with unbiased analytic step) with respect to increasing number of simulations, is provided below.

Simulations	MC - analytic	Absolute error
10000	2.544556	0.055988
20000	2.568402	0.079834
30000	2.567637	0.079069
40000	2.569965	0.081397
50000	2.567561	0.078993
BSM formula	2.488568	

Figure 10: MC method with analytic step

It is obvious that the soar in the number of simulations N, did not lead to the better accuracy of the estimated option price. A reasonable explanation for this is the bias-variance tradeoff; since we set bias to zero (with implementation of the analytic step), we cannot decrease variance at the same time.

5.1.2 Finite Difference for a Call

Now we turn to the finite difference method by Crank and Nicolson. The following table shows us that the accuracy of this method enhanced significantly by enlarging the number of intervals for time- and space-discretizations. For instance, in the case when the number of time intervals $v_{max} = 2$ and the number of space intervals $m = 200$, the result is poor: 2.869586. In contrast to that, when the number of time and space subintervals jumped to 6 respective 400, the predicted option price was clearly closer to the value obtained by the Black-Scholes-Merton formula.

	Space intervals = 200	Space intervals = 300	Space intervals = 400
Time intervals = 2	2.869586	2.692115	2.601940
Time intervals = 4	2.862608	2.650359	2.545548
Time intervals = 6	2.831916	2.618690	2.515134

Figure 11: FD method - call option

The following table shows the absolute errors for the FD method. The greatest error is in the left-upper cell and the smallest is in the right-lower corner.

	Space intervals = 200	Space intervals = 300	Space intervals = 400
Time intervals = 2	0.381018	0.203547	0.113372
Time intervals = 4	0.374040	0.161791	0.056980
Time intervals = 6	0.343348	0.130122	0.026566

Figure 12: FD method - call option

5.2 Put Option

Now we will analyze the results of two methods in the case of the put option with parameters: $S_0 = 45, K = 35, T = 1, r = 0.05, \sigma = 0.25$. The price of the put computed by Black-Scholes-Merton formula is 0.53423.

5.2.1 Monte Carlo for a Put

Again, we start with the table related to Monte Carlo method with analytic algorithm.

Time intervals	MC – analytic	Absolute error
10	0.559212	0.024982
20	0.551183	0.016953
30	0.571613	0.037383
40	0.566267	0.032037
50	0.558701	0.024471
60	0.56526	0.03103
70	0.562842	0.028612
80	0.559755	0.025525
90	0.565484	0.031254
BSM formula	0.53423	

Figure 13: MC method with analytic step: time intervals increase

As in the case of the call option, greater number of intervals due to time-discretization did not make estimations of the put option price more accurate. As already stated, that can be explained by the fact that, when simulating the paths of a stock price, we use unbiased analytic step for which the mean square error (MSE) does not depend on the number of time-discretization subintervals.

Regarding Euler step in the MC algorithm, generally the results are still worse than in the analytic case, but with two outliers when time subintervals are equal to 30 and 50.

Time intervals	MC – Euler	Absolute error
10	0.581982	0.047752
20	0.561061	0.026831
30	0.562869	0.028639
40	0.567433	0.033203
50	0.557016	0.022786
60	0.567187	0.032957
70	0.565109	0.030879
80	0.563384	0.029154
90	0.567946	0.033716
BSM formula	0.53423	

Figure 14: MC method with Euler step: time intervals increase

The absolute errors again oscillate for analytic step. The same holds true for Euler step as well, but in this case a downward trend can be noticed. Note that that all errors are smaller than 0.05.

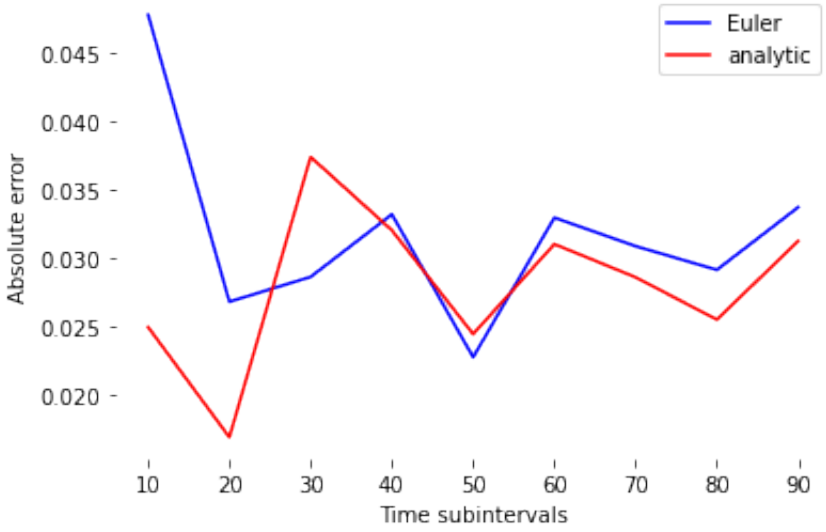

Figure 15: Absolute errors - Euler vs analytic step

Furthermore, soaring number of the number N of simulations did not give rise to the better results for the MC method with analytic step. The explanation is the same as in the call case; since we made bias equal to zero, variance cannot be reduced by growing N because of the beforementioned bias-variance tradeoff.

Simulations	MC analytic	Absolute error
10000	0.551183	0.016953
20000	0.561762	0.027532
30000	0.561401	0.027171
40000	0.560151	0.025921
50000	0.558121	0.023891
BSM formula	0.53423	

Figure 16: MC method with analytic step

5.2.2 Finite Difference for a Put

It remains to take a look at the table of FD method results. Similarly as before, finite difference method remarkably improved its accuracy when numbers of intervals (for the time- and space- discretization) jumped. In the table, the worst result is obtained when the number v_{max} of time subintervals is equal to 2 and the number m of space subintervals is 100. On the other hand, the results in the last column (i.e. $m = 500$) are almost equivalent for $v_{max} = 4$ and $v_{max} = 8$ and, at the same time, the best of all in the table.

	Space intervals = 100	Space intervals = 300	Space intervals = 500
Time intervals = 2	0.326003	0.449691	0.477401
Time intervals = 4	0.345915	0.472593	0.499226
Time intervals = 8	0.348150	0.472698	0.499107

Figure 17: FD method: put option

	Space intervals = 100	Space intervals = 300	Space intervals = 500
Time intervals = 2	0.208227	0.084539	0.056829
Time intervals = 4	0.188315	0.061637	0.035004
Time intervals = 8	0.186079	0.061532	0.035123

Figure 18: FD method: absolute errors

6 Conclusion

The aim of this master's thesis was to present and implement Monte Carlo and finite difference method with the intention of pricing European options. In chapter 3, we described the core of MC method, how to improve the accuracy of the initial algorithm and provided Python codes for the method. Similarly, chapter 4 includes mathematical background of FD method and all necessary transformations that allow us to implement the algorithm using the same programming language. Furthermore, chapter 5 was the culmination of the thesis as we showed the results that those two methods achieved in pricing the call and the put option. Now it is time to try to draw some conclusions about the methods we considered.

According to the outcome of our implementations that are provided in chapter 5, finite difference method applied on Black-Scholes-Merton partial differential equation showed better performances in the option pricing because we could significantly improve the accuracy of this method by enlarging the number of subintervals for the time- and space-discretization. This is somehow expected because greater number of subintervals makes the points on the two-dimensional grid denser, which implies better approximation of the solution of the initial partial differential problem. Note that, the fact that bigger number of time and space subintervals caused better approximations in our examples from chapter 5, is aligned with the Theorem 17 from the fourth chapter of the thesis.

When it comes to Monte Carlo method, we tried two approaches: Euler step which is biased and the analytic step which is not, both accompanied with the variance reduction procedure. In general, the analytic step achieved better accuracy; this is logical because of the fact that in this case the bias does not contribute to the mean square error MSE, which does not hold if we implement the Euler algorithm. Soaring number M of time-discretization subintervals did not affect results of the algorithm with the analytic step because MSE does not depend on M in this case (check Ex.1. and the following remark from the section 3.2). Concerning the algorithm with Euler step, even though oscillations are present, decreasing trend of the error is visible when M jumps. Furthermore, when we started to increase the number N of simulations, Monte Carlo procedure with the analytic step still did not enhance its accuracy because of the bias-variance tradeoff. In other words, since we have already set bias to zero (using the analytic step), we could not decrease the error part that stems from the variance at the same time. All in all, according to the analysis of the results from chapter 5, Monte Carlo method showed poorer performances compared to the finite difference algorithm because its accuracy could not be improved, whereas the results of FD method could be enhanced remarkably.

To wrap up, even though Monte Carlo method provided weaker results than the finite difference algorithm in pricing European options, MC procedure might be a valuable asset in multidimensional cases. Also, when pricing other type of options, for instance those that are path dependent (e.g. barrier options), Monte Carlo method might be very important and useful tool.

7 Bibliography

References

[1] Rüdiger U. Seydel: "Tools for Computational Finance"; Sixth Edition, Springer, London, 2017.

[2] Perla Sousi: "Advanced Probability"; University of Cambridge, Cambridge, 2013.

[3] Nathanaël Berestycki: "Stochastic Calculus and Applications", Cambridge

[4] Vanja Wagner: "Financijsko modeliranje 1", Prirodoslovno matematički fakultet – Matematički odsjek, Sveučilište u Zagrebu, Zagreb

[5] Vanja Wagner: "Financijsko modeliranje 2", Prirodoslovno matematički fakultet – Matematički odsjek, Sveučilište u Zagrebu, Zagreb