Faculty of Natural and Mathematical Sciences
Department of Information

King's College London
Strand Campus, London,
United Kingdom

7CCSMPRJ

Individual Project Submission 2022/23

Name:	Yingming Luo
Student Number:	K2257345
Degree Programme:	MSc Computational Finance
Project Title:	Pricing of Exotic Options Using Monte Carlo Simulation
Supervisor:	Riaz Ahmad
Word Count:	9710

RELEASE OF PROJECT

Following the submission of your project, the Department would like to make it publicly available via the library electronic resources. You will retain copyright of the project.

☑ I **agree** to the release of my project

☐ I **do not** agree to the release of my project

Signature: Yingming Luo **Date:** August 15, 2023

Department of Information
King's College London
United Kingdom

7CCSMPRJ Individual Project

Pricing of Exotic Options Using Monte Carlo Simulation

Name: **Yingming Luo**
Student Number: K2257345
Course: MSc Computational Finance

Supervisor: Riaz Ahmad

This dissertation is submitted for the degree of MSc in MSc Computational Finance.

Acknowledgements

I extend my heartfelt gratitude to Professor Riaz Ahmad for his invaluable guidance throughout this project. My sincere thanks also go to my parents for their unwavering support. I am deeply appreciative of my girlfriend, Lylian, for her constant encouragement. Their combined support has been pivotal in driving my efforts. Lastly, I am thankful to everyone who has contributed to this project in any way, shaping its outcome and making this journey meaningful.

Abstract

This thesis explores the pricing of Barrier and Asian options through Monte Carlo simulation. The research begins by establishing the foundational theories of these exotic options and investigating their closed-form solutions under the Black-Scholes framework. Additionally, three different random walk models are introduced: Euler Scheme, Euler-Maruyama, and Milstein's method, to generate underlying asset paths for simulation.

The simulations demonstrate the effectiveness of the antithetic variate technique in reducing standard error and accelerating convergence when pricing both Barrier and Asian options. This variance reduction method emerges as a valuable tool for enhancing the precision of Monte Carlo estimates. Interestingly, the study highlights that the choice of underlying random walk models has minimal influence on accuracy, implying the potential for further research in exploring advanced variance reduction techniques. In summary, this research provides crucial insights into pricing exotic options, offering valuable implications for finance professionals and researchers, and driving innovation in option pricing methods for the dynamic financial markets.

Nomenclature

$S0$	Initial asset price
E	Strike price (in equations)
K	Strike price (in code)
r	risk-free interest rate
σ	volatility
T_0	current time
t	current time
T	maturity day
$T-t$	time to expiry
N	number of time steps
n	number of time steps
M	number of simulations
$N(.)$	cumulative distribution function of standard normal distribution
dt	single time-step
Sb	barrier price
S_u	upper barrier price
S_d	lower barrier price
S_u^*	adjusted upper barrier price
S_d^*	adjusted lower barrier price

Contents

List of Figures

List of Tables

1 Introduction

The primary objective of this research endeavor is to delve into the utilisation of Monte Carlo (MC) simulation techniques for the valuation of exotic options, focusing particularly on Asian options and Barrier options. This investigation will further encompass the integration of the renowned Black-Scholes model, an extensively employed pricing framework within the financial domain, to capture the dynamics of the underlying asset's price behavior.

One of the current challenges in financial derivative trading lies in valuing exotic options, which possess intricate payoffs influenced by non-linear characteristics and path dependencies. This complexity often poses considerable difficulties for financial institutions and investors alike. Established frameworks such as the Black-Scholes model may falter in effectively pricing such options, primarily due to their reliance on numerous unrealistic assumptions. Furthermore, the absence of closed-form mathematical solutions complicates matters further, as these options exhibit multifaceted structures that are challenging to capture using traditional methods. They are also constrained by real-world considerations. For instance, in the actual market, option contract terms often rely on prices that are monitored on specified discrete time intervals to prevent disputes over the triggering of certain features in exotic options. This approach is necessary due to the near-impossibility of continuous monitoring of underlying prices, primarily because of network delays. Consequently, the adoption of Monte Carlo simulation, a robust numerical technique, has witnessed a surge in popularity within the financial sector as a means of estimating the prices of exotic options with a high degree of accuracy.

This study aims to assess the efficacy of Monte Carlo simulation in pricing Barrier and Asian options. To enhance the efficiency and precision of the simulations, the Antithetic Variate method, a variance reduction technique, will be employed. The project's technical requirements involve utilising Python for implementing the Monte Carlo simulation and conducting an in-depth analysis of the simulated data.

The structure of this thesis is designed to thoroughly explore various aspects of option pricing, with each chapter contributing to a comprehensive understanding of the topic. In Chapter 2, the fundamental concepts of vanilla option pricing are introduced, emphasising their differences from exotic options, particularly concerning path dependency. Chapter 3 bridges the theoretical foundations with practical applications, covering the Black-Scholes model and deriving the Black-Scholes Partial Differential Equation. Chapter 4 delves into the intricacies of barrier options, discussing their various types and examining available closed-form solutions. Chapter 5 focuses on Asian options, exploring their unique characteristics and investigating the feasibility of closed-form solutions. Moving on to Chapter 6, we delve into the Monte Carlo simulation method, a powerful numerical approach for pricing complex options, where the antithetic variates technique is employed to enhance efficiency and accuracy. Chapter 7 presents the simulated re-

sults and conducts an in-depth analysis, comparing the efficiency of the Monte Carlo method against specific benchmarks. Chapter 8 discusses the legal, social, ethical and professional issues associated with the context of this study. Finally, Chapter 9 concludes by summarising key findings, highlighting essential contributions, and discussing future aspects of the work. Throughout this thesis, the aim is to provide readers with comprehensive insights into option pricing methods, with a crucial focus on their role in financial markets. Importantly, the thesis incorporates literature review throughout its content, seamlessly weaving it into the discussion rather than dedicating a separate chapter solely for this purpose.

2 Background Theories

Before getting into exotic options it is best to understand what are financial derivatives. As the term suggests, derivatives are financial contracts negotiated and agreed upon between two or more parties, where its values are derived from the price movements of one or more underlying assets or some benchmarks [1]. Derivatives are vital to the financial industry for several reasons with the most important one being risk management. Investors often trade derivatives, either through the exchange or over the counter, to hedge against risks in the market [1].

2.1 Vanila Options

One particular category of financial derivatives is the Options contract, which has two subclass: Call option and Put option. Ever since 1973, organised exchanges have facilitated the trading of modern call and put options using stocks as their underlying asset [2]. The purchase of a call option on a stock grants the buyer the right to buy the stock at a predetermined price, called the strike price (denoted as E), on or before a predetermined expiration date (denoted as T), but it does not impose an obligation to do so [3]. Conversely, the purchase of a put option grants the buyer the right to sell the stock at a predetermined strike price on or before an expiration date, but again, it does not impose an obligation to do so. When the buyers do decide to use their right to perform the action of buying or selling, it is said that the option is exercised [3].

The buyer, also known as the option holder, incurs the cost of purchasing the option, which is referred to as the premium. The seller on the other hand is known as the option writer, is obligated to sell or buy the underlying asset at the strike price to the holder if the option is exercised. Thus the study of option pricing hinges on this central aspect: without knowing the future changes in the underlying asset's price, how can one determine the premium of an option.

Plain vanilla options are often used to refer to basic call and put options that lack any special or complex features. There are two general classes of vanilla options, one is the European Option, where the buyer can only exercise their right to buy or sell on the expiration date; the other one is the American Option, where the buyers can exercise their right anytime before the expiration date [4]. In this chapter, more emphasis will be placed on European options because the exotic options investigated later on are also European style options.

2.2 European Option Payoffs

The payoff of an European option refers to the profit or loss gained by the option holder at the maturity date, based on the underlying asset price at expiry (denotes as S_T) and

the terms of the contract.

European Call Option:
For European Call option, the payoff function is given by:

$$C = max(S_T - E, 0) \tag{2.1}$$

This means that, if:

- $S_T > E$, the option holder should exercise the option to buy the underlying asset at the lower strike price E and sell at the higher market price S_T to generate an positive payoff of $S_T - E > 0$.
- $S_T \leq E$, the option holder should not exercise the option, and payoff is 0.

Thus, at expiry, this call option would give the holder a value of $max(S_T - E, 0)$, which is never negative. Whereas the option seller has a payoff of $-max(S_T - E, 0)$, which is never positive. Thus the option writer must be paid a premium for the contract.

European Put Option:
For European Call option, the payoff function is given by:

$$P = max(E - S_T, 0) \tag{2.2}$$

This means that, if:

- $S_T < E$, the option holder should exercise the option to sell the underlying asset at the higher strike price E to generate an positive payoff of $E - S_T > 0$.
- $S_T \geq E$, the option holder should not exercise the option, and payoff is 0.

Based on the payoff function, a European option can be in one of the three states at expiry: in the money, at the money, or out of the money. Summarised in Table 2. It is clear that only when the option is in the money, it will be exercised by the holder [5].

	Call Option	Put Option
At the money	$S_T = E$	$S_T = E$
In the money	$S_T > E$	$S_T < E$
Our of the money	$S_T < E$	$S_T > E$

Table 2: Three states of European Option at expiry.

2.3 Put-Call parity

Given the payoff functions discussed above, under the condition of no arbitrage opportunities, one can construct a portfolio of a long Call and short Put, which is guaranteed

to receive a payoff of holding a long asset S and short cash $Ee^{-r(T-t)}$ position [3]. This important relationship between the two types of options is called the Put-Call parity, given by:

$$C - P = S - Ee^{-r(T-t)} \tag{2.3}$$

This parity is valid at any time up to expiry of the options contract and is independent of both the future movements of the underlying asset price and the model used to simulate the underlying [3].

Using the Put-Call parity, it reveal that if the current price of the underlying, the strike price and maturity date is known, then given the price of a European Call option, one can easily obtain the price of European Put option that shares the same contract conditions and vice versa.

2.4 Further reviews of vanilla options

However, there are many limitations to these vanilla options due to its simplicity and fixed payoff structure, which limits their flexibility to meet the needs of different investors. As discussed above, when investors want to use options to hedge against certain risks, they must pay a premium upfront in order to acquire the option, which can be expensive. Furthermore, vanilla options can be limited in their effectiveness as a hedging tool in volatile markets. One reason is that they have fixed strike prices, which means that if the market moves significantly in one direction, the option may become out of the money and lose much of its value. Additionally, vanilla options are often priced using models that assume constant volatility, such as the original Black-Scholes model, which "often fail when compared to the real market data" [6]. Therefore, in order to address these limitations, a new category of options called Exotic options has been developed.

2.5 Exotic Options

Unlike the fixed and straightforward framework of vanilla options, exotic options are designed to offer more customised or tailored risk management solutions to investors. They can provide more flexibility in terms of the payoff structure and can be tuned to meet the specific needs of an investor. Moreover exotic options can also be used to hedge against more complex risks, which cannot be hedged using standard vanilla options, and enhance the returns on an investment portfolio by taking on more sophisticated and complex trading strategies.

Aside from the nuanced details in payoff structure between the vanilla and exotic options, there is also a fundamental difference between the two that we must emphasise. That is, vanilla options are path-independent [7]. The payoff of the American and European

options depends only on the price of the underlying asset at the time of exercise, not on the path that the asset price took to get there. Exotic options on the other hand are path-dependent [3], which means their value is not only determined by the underlying asset's price at expiration but also the path it takes to get there. As Wilmott [3] has pointed out, there are two types of path dependency, namely strong and weak, with the difference being strong path-dependence options work in higher dimensions that require us to introduce an additional variable to deal with path dependency, whereas the weak path-dependence ones do not. In order to understand the effectiveness of MC simulation methods in pricing both weakly and strongly path-dependent options, two exotic options will be considered in this thesis: Barrier Options (weakly path-dependent) and Asian Options (strongly path-dependent).

3 Black-Scholes Model

This chapter aims to present a succinct overview of the Wiener Process in subsection 3.1 and Ito's Lemma in subsection 3.2, as they constitute the fundamental building blocks necessary for deriving the Black-Scholes model. The subsequent sections present a detailed derivation of the Black-Scholes partial derivative equation, culminating in the final equations for pricing vanilla European options.

One of the very popular models used in industry to price options is the Black-Scholes model developed and made publicly available by Fischer Black and Myron Scholes in 1973 [8]. The model is a mathematical formula used to determine the fair price for a European Call or Put option, in other words the premium, by considering the following factors: underlying asset price (S), expiration date (T), volatility of the underlying (σ), drift rate (μ), risk-free interest rate (r) and strike price (E) [9]. Hence, for option pricing, the value V of an option can be a function denoted as:

$$V(S, t; \sigma, \mu; E, T; r) \tag{3.1}$$

where t is the current time, and $T - t$ represents time to expiry. To simplify notations, in the derivation of the Black-Scholes equation below, the option value will simply be written as $V(S, t)$ or V.

3.1 Brownian Motion / Wiener Process

According to the efficient market hypothesis, asset prices move randomly following two key principles. The first principle emphasises that the present price fully incorporates the entire past history. The second principle asserts that markets instantaneously respond to any new information concerning an asset [10]. This means that the asset prices should behaves the same way as a Markov process.

Suppose at time t, the price of an asset is S, with a small change in time interval dt, the asset price would change to $S + dS$. The associated return of the asset price change can be calculated as dS/S, which is the change in price divided by the original price. In order to simulate such price return, a simple model called Brownian motion, or Wiener process, is developed. This model consist of two parts:

- **deterministic part (μdt)**: a predictable return comparable to the interest gained from investing in a risk-free bank, where μ is the average growth rate of the asset price, also known as the drift rate.
- **random part (σdW_t)**: a random part that mimics the arrival of unanticipated news that causes the price to change. Here the dW_t term is a random variable drawn from a normal distribution, which represents the Wiener process with a mean of 0 and variance dt. σ is the volatility, a constant term that measures the standard deviation of the price returns, also known as the diffusion rate.

By adding the terms together we can form a **stochastic differential equation** capable of generating random price movements for assets, resembling random walks:

$$\frac{dS}{S} = \mu dt + \sigma dW_t \tag{3.2}$$

To be more precise, Equation 3.2 is also called a Geometric Brownian Motion (GBM) because asset's price changes proportionally to its current value. This implies that the percentage returns are normally distributed with a constant drift and volatility, and the asset's price trajectory resembles a geometric growth or decay over time[3].

3.2 Ito's Lemma

For the Brownian motion defined above, it exhibits the property of being continuous everywhere but differentiable no where. This is due to the Wiener process, being a random variable, create "jumps" equivalent to infinitesimal variations along its path. Therefore the standard calculus cannot be applied to stochastic differential equations, when we want to model the random walk in continuous time limit $dt \to 0$.

However, according to [10], "Ito's lemma relates small changes in a function of a random variable to the small change in the random variable itself." Thus allowing us to compute the derivative of a function that depends on a stochastic variable, providing a way of handling the Wiener process dW_t as $dt \to 0$.

In general form:
If a function $G(t)$ satisfies a GBM, such that it can be represented in the following form of a stochastic differential equation:

$$dG_t = A(t, G_t)dt + B(t, G_t)dW_t \tag{3.3}$$

where A and B are functions of G_t and t, corresponding to the drift and diffusion rates.

Now consider a function $F = F(t, G_t)$, according to Ito's lemma, the derivative dF can be represented as:

$$dF = \left(\frac{\delta F}{\delta t} + A\frac{\delta F}{\delta G_t} + \frac{1}{2}B^2\frac{\delta^2 F}{\delta G_t^2}\right)dt + B\frac{\delta F}{\delta G_t}dW_t \tag{3.4}$$

where $\frac{\delta F}{\delta t} + A\frac{\delta F}{\delta G_t} + \frac{1}{2}B^2\frac{\delta^2 F}{\delta G_t^2}$ is the drift rate and $B\frac{\delta F}{\delta G_t}$ is the diffusion rate.

Apply to option value V:
Suppose $V = V(t, S_t)$ where the underlying asset price S_t at time t satisfies GBM, such that:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \tag{3.5}$$

Applying Ito's lemma, the change in option value dV can be calculated by:

$$dV = \left(\frac{\delta V}{\delta t} + \mu S\frac{\delta V}{\delta S} + \frac{1}{2}\sigma^2 S^2\frac{\delta^2 V}{\delta S^2}\right)dt + \sigma S\frac{\delta V}{\delta S}dW_t \tag{3.6}$$

3.3 Deriving Black-Scholes Equation

The foundation of the model is based on many assumptions, to name a few, the underlying asset price follows a log-normal random walk, short selling is allowed, interest rate and volatility of the market are known constants, no transaction costs, no dividend payout on the underlying and no risk-free arbitrage opportunities [9]. Under these assumptions, the model is able to construct a delta-hedged portfolio consisting of a long position in the stock and a short position in the option. The derivation of the Black-Scholes equation below is based on the book written by P. Wilmott (2007)[3].

First, construct a delta-hedged portfolio, with Π denoting the value of the portfolio, which consist of two components, a long option position $V(S,t)$, and a short position in some quantity delta (Δ) of the underlying asset S.

$$\Pi = V(S,t) - \Delta S \tag{3.7}$$

Then across one time step: $t \to t + dt$, $\Pi \to \Pi + d\Pi$, we hold Δ fixed across each time step so Δ is treated as a constant, meaning:

$$d(\Delta S) = \Delta dS \tag{3.8}$$

hence:

$$\begin{aligned} d\Pi &= dV - d(\Delta S) \\ d\Pi &= dV - \Delta dS \end{aligned} \tag{3.9}$$

Using the results from Ito's lemma for dV in Equation 3.6 and substitute into $d\Pi$ gives:

$$\begin{aligned} d\Pi &= \left(\frac{\delta V}{\delta t} + \mu S\frac{\delta V}{\delta S} + \frac{1}{2}\sigma^2 S^2\frac{\delta^2 V}{\delta S^2}\right)dt + \sigma S\frac{\delta V}{\delta S}dW_t - \Delta\left(\mu S dt + \sigma S dW_t\right) \\ &= \left(\frac{\delta V}{\delta t} + \mu S\frac{\delta V}{\delta S} - \Delta\mu S + \frac{1}{2}\sigma^2 S^2\frac{\delta^2 V}{\delta S^2}\right)dt + \sigma S\left(\frac{\delta V}{\delta S} - \Delta\right)dW_t \end{aligned} \tag{3.10}$$

In order to eliminate risk, which is the dW_t term, we set $\Delta = \frac{\delta V}{\delta S}$.

Sub $\Delta = \frac{\delta V}{\delta S}$ back into $d\Pi$ in Equation 3.10 gives:

$$d\Pi = \left(\frac{\delta V}{\delta t} + \frac{1}{2}\sigma^2 S^2\frac{\delta^2 V}{\delta S^2}\right)dt \tag{3.11}$$

By following the no risk-free arbitrage opportunities assumption, the portfolio must earn

at the risk-free interest rate r:

$$d\Pi = r\Pi dt \tag{3.12}$$

Finally equating the two $d\Pi$ equations from Equation 3.11 and Equation 3.12:

$$\begin{aligned} \left(\frac{\delta V}{\delta t} + \frac{1}{2}\sigma^2 S^2 \frac{\delta^2 V}{\delta S^2}\right) dt &= r\Pi dt \\ \frac{\delta V}{\delta t} + \frac{1}{2}\sigma^2 S^2 \frac{\delta^2 V}{\delta S^2} &= r\left(V - S\frac{\delta V}{\delta S}\right) \end{aligned} \tag{3.13}$$

By rearranging, we arrive at the famous 1973 Black-Scholes partial differential equation (PDE):

$$\frac{\delta V}{\delta t} + \frac{1}{2}\sigma^2 S^2 \frac{\delta^2 V}{\delta S^2} + rS\frac{\delta V}{\delta S} - rV = 0 \tag{3.14}$$

3.4 Black-Scholes formula for European Options

In order to obtain the formulas for pricing European options, we must solve the Black-Scholes equation with final conditions depending on the option payoffs. A full derivation can be found in the textbook written by P. Wilmott (2007)[3]. In this thesis, we will simply make use of the resulting analytical formulas which can be solved to calculate the exact price of European options under the Black-Scholes framework.

Call option value:

$$C(S,t) = SN(d_1) - Ee^{-r(T-t)}N(d_2) \tag{3.15}$$

Put option value:

$$P(S,t) = -SN(d_1) + Ee^{-r(T-t)}N(-d_2) \tag{3.16}$$

where $N(.)$ is the cumulative distribution function for the standard normal distribution $\mathcal{N}(0,1)$:

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}\phi^2} d\phi \tag{3.17}$$

and:

$$\begin{aligned} d_1 &= \frac{log\left(\frac{S}{E}\right) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}} \\ d_2 &= \frac{log\left(\frac{S}{E}\right) + \left(r - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}} \end{aligned} \tag{3.18}$$

3.5 Further observations

The Black-Scholes equation contains all the obvious variables and parameters such as the underlying (S), time (t), volatility (σ) and risk-free interest rate (r), but there is no mention of the drift rate (μ). This is because delta-hedging has cancelled out the μ

term at Equation 3.11, meaning the return of the underlying stock is no longer whatever the stock return for that particular share is, as there is no more risks involved since the dW_t term is eliminated. Thus the stock cannot grow at any rate other than the risk-free return, which is why the model only has the risk-free interest rate (r) to represent the growth rate.

In mathematical terms, originally the asset price is evolving according to Equation 3.2. Now under the Black-Scholes framework, the asset price is evolving according to:

$$\frac{dS}{S} = rdt + \sigma dW_t \tag{3.19}$$

By continuously hedging, the fluctuations in the value of a long stock position can be completely balanced out by the corresponding changes in the value of a short option position, leading to a risk-less portfolio and a certain return under the Black-Scholes assumptions.[9]. Although there have been many criticisms of the unrealistic assumptions made by the model, nonetheless, the easy to use closed form solution for pricing the European option proposed by the model, together with its framework for understanding the relationship between the price of an option and the underlying asset were invaluable to the development of quantitative finance. In this thesis, the analytical solutions to pricing options derived from the Black-Scholes model will be used as benchmarks to validate the results obtained from numerical methods, and help improve the accuracy of simulations.

4 Barrier Options

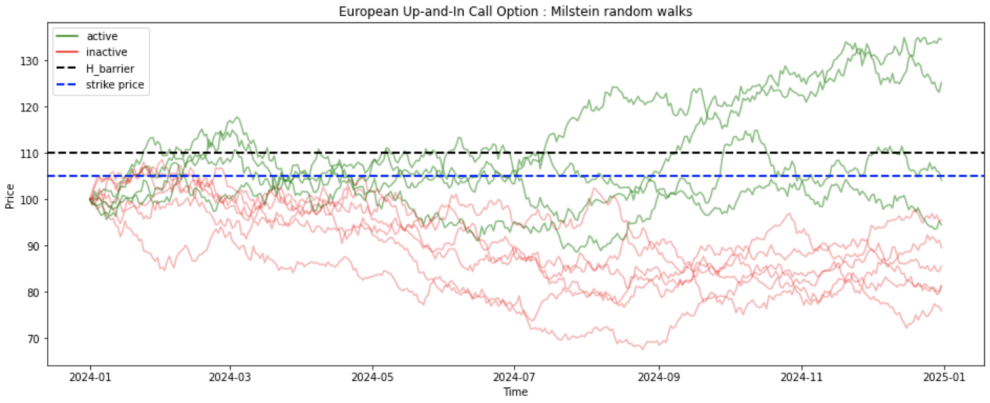
Barrier options represent a highly sought-after category of financial derivatives, demonstrating a remarkable surge in market demand, doubling annually since 1992 [11]. The diversity of available barrier options has also witnessed substantial growth. These exotic options hinge on whether the underlying asset surpasses or falls short of a predetermined price, making them profoundly significant. In the finance world, there is always a continual interest in evading specific price levels or, conversely, reaching designated price levels. Back in 1973, Merton [12] first proposed an analytical solution for calculating European style down-and-out Call options, conditioned on the continuous monitoring of price movement. Later around the year of 1991, further pricing techniques of knock-in types of calls and puts barrier options were proposed by Rich [13], Rubinstein and Reiner [14]. Shortly afterwards, Heynen and Kat [15] and Carr [11] put forward new approaches for pricing more intricate variants of the barrier options. For instance, rainbow barrier options, which is a complex financial derivative that combines features of both barrier options and rainbow (basket) options, with barrier conditions based on the collective performance of a basket of underlying assets [16]. A comprehensive examination of European and American style barrier options is elaborated upon by AtiSahlia et al. [17]. However, the variations listed above is beyond the scope of this thesis, this chapter will focus on the primitive single barrier European style option instead.

As previously mentioned, Barrier options are path-dependent options. The payoff of these contracts is linked to the level of the realised asset path, and specific conditions in the contract are activated if the asset price reaches a certain "barrier", whether it is too high or too low [3]. The barrier can be specified as either "in" or "out," and it can be set at different levels, such as "up" or "down." In general, an "in" barrier option only becomes active or "knocks in" if the price of the underlying asset reaches the barrier level during the life of the option. Conversely, an "out" barrier option becomes void or "knocks out" if the price of the underlying asset reaches the barrier level during the life of the option.

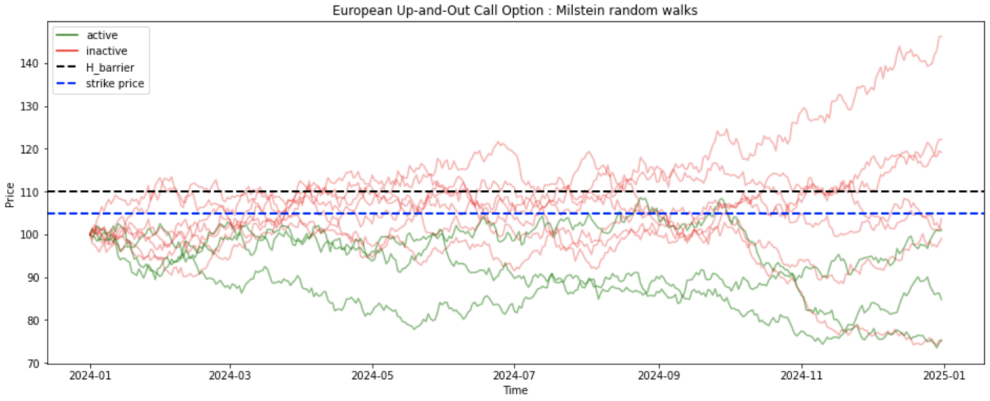
The classification of barrier options can be further extended according to the barrier's position S_b relative to the initial value of the underlying asset price S_0. Specifically, an "up" barrier option has a barrier level set above the current price of the underlying asset, $S_b > S_0$, while a "down" option satisfies $S_b < S_0$.

The mix and match of these features provides a set of Barrier options listed below. In each figure it consist of 10 possible random price movements to help illustrate the mechanisms of Barriers in different option types. Green paths corresponds to situations where the corresponding option contract remains active at expiry. Conversely, the red paths corresponds to situations where the option contract has become nullified, because they have either crossed (or failed to reach) the barrier level:

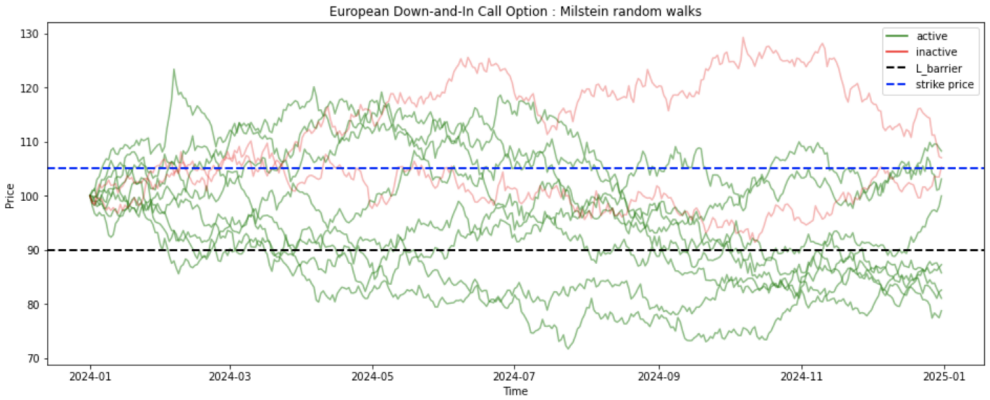
- **up-and-in call option**



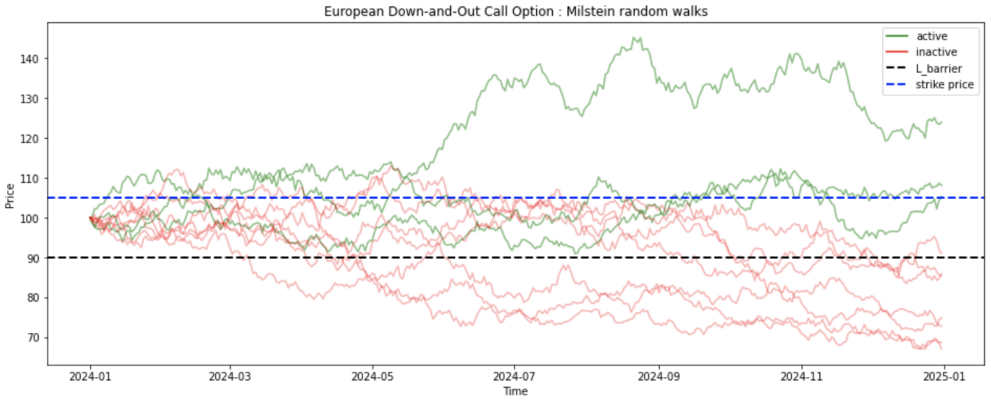
- **up-and-out call option**



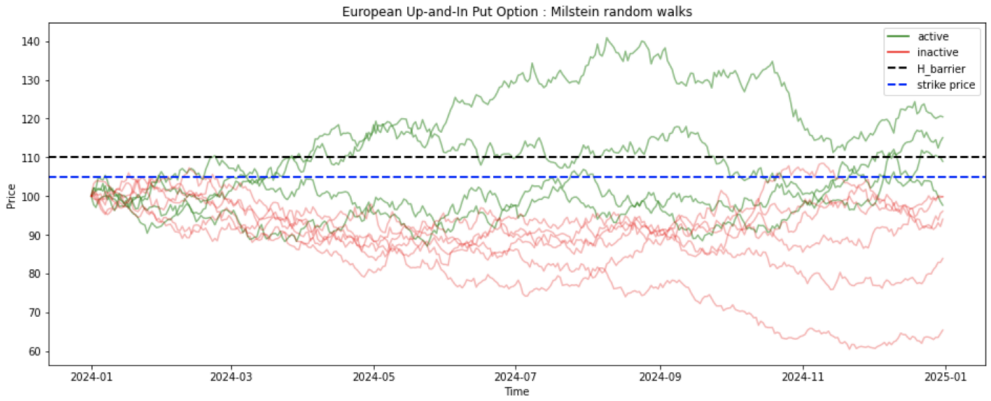
- **down-and-in call option**



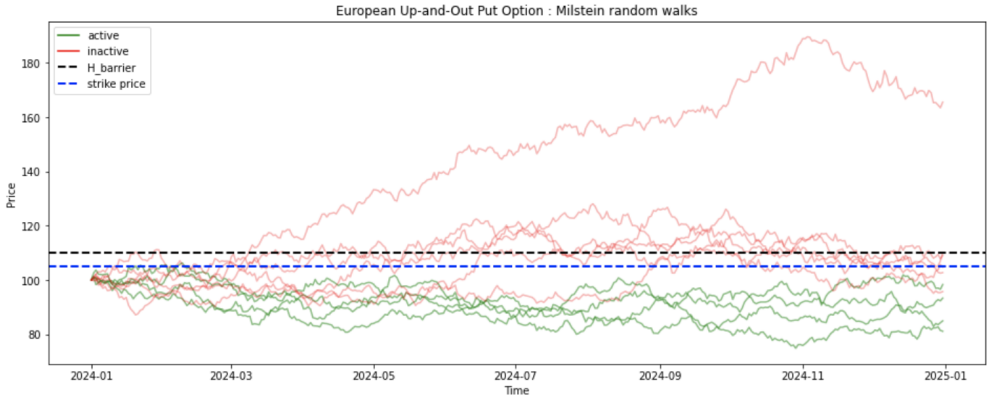
- **down-and-out call option**



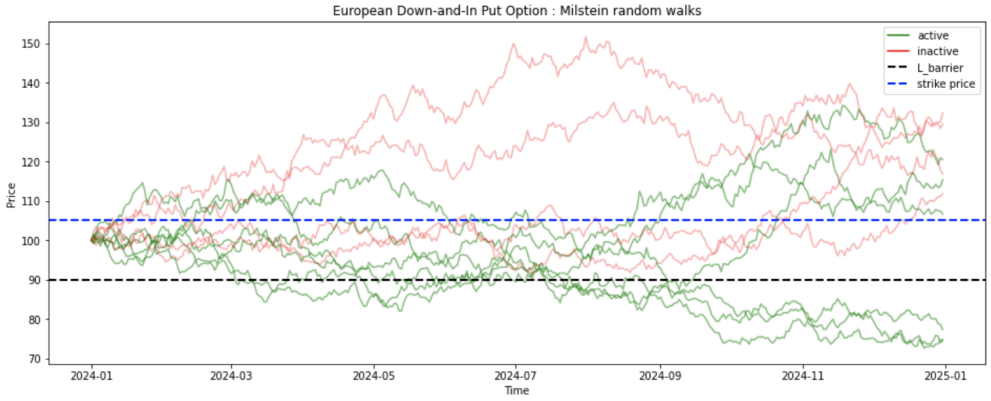
- **up-and-in put option**



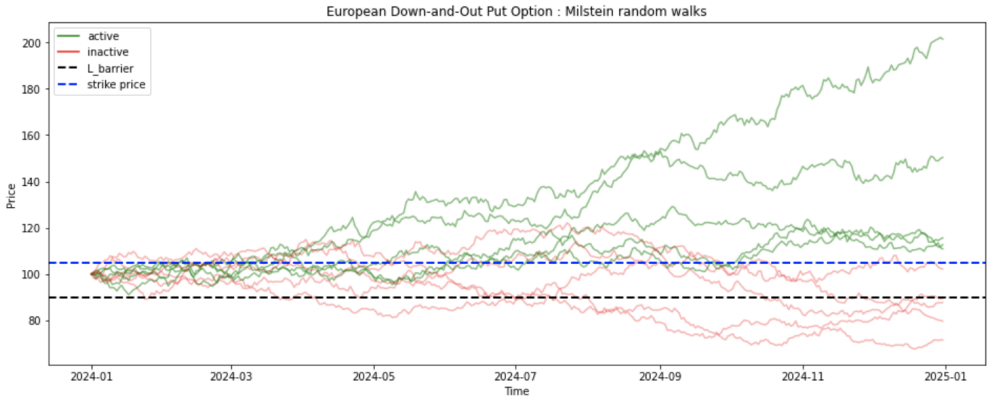
- **up-and-out put option**



- **down-and-in put option**



- **down-and-out put option**



Furthermore, Barrier option is also **weakly** path-dependent, because it works in low dimension. In other words, under the Black-Scholes framework, the value of the contract only depends on two independent variables: the current price level of the underlying asset S, and time to expiry t, which makes it a two dimensional option [3].

Take the up-and-out call option as an example. Before hitting the barrier level, the option works the same way as a plain vanilla call option. As it gives the holder the right (but not the obligation) to buy the underlying asset at the strike price. However, once the underlying asset reaches or surpasses the specified barrier level at any point during the option's lifetime, the option is knocked out and becomes worthless. While the up-and-out call option provides the potential for profit if the price of the underlying asset rises and stays below the barrier level, it also restricts the potential gains if the price surpasses the barrier. This restricting feature naturally makes the up-and-out call option to become cheaper than a regular vanilla call option.

4.1 In-Out parity of Barrier options

Similar to the put-call parity of European vanilla options defined in subsection 2.3, an in-out parity also exists between the value of Barrier options. Given a portfolio comprising one European in-option and one European out-option, both sharing the same barrier,

strike price, and expiration date. The total value of this portfolio is equivalent to that of a corresponding European option with the same strike price and expiration date. This is because, as one option gets knocked-out at the barrier level, the other becomes knocked-in. Thus at expiry, only one of the two barrier options will remain active, and its payoff will be identical to that of the European option [18]. The exact in-out parities are listed below:

$$\begin{aligned} C_{vanilla}(S,t) &= C_{up-and-in}(S,t,S_b) + C_{up-and-out}(S,t,S_b) \\ &= C_{down-and-in}(S,t,S_b) + C_{down-and-out}(S,t,S_b) \end{aligned} \tag{4.1}$$

$$\begin{aligned} P_{vanilla}(S,t) &= P_{up-and-in}(S,t,S_b) + P_{up-and-out}(S,t,S_b) \\ &= P_{down-and-in}(S,t,S_b) + P_{down-and-out}(S,t,S_b) \end{aligned} \tag{4.2}$$

Consequently, once the value of the corresponding in-option is known, the value of an out-option can be readily determined, and vice versa.

4.2 PDE for Barrier Options

In practice, the vast majority of barrier options traded in markets are monitored discretely. This means that they establish specific and fixed intervals for checking the barrier status, often on a daily closing basis [19]. Under this monitoring mode, it is difficult to derive any analytical solutions for pricing the Barrier options, as the model need to take into account the discontinuities in the option's payoff function. However, if the asset price is monitored continuously, meaning if the asset price does hit the barrier level, the option immediately becomes "knocked-in" or "knocked-out", then it is possible to derive analytical formulas by extending the Black-Scholes model with new boundary conditions incorporating the barrier level[4].

According to Hu et al., (2006), as the Barrier options are only weakly path-dependent, the Black-Scholes partial differential equation governing the value of the option is the same as the one defined for European vanilla options in Equation 3.14.

Subject to final conditions:

$$V(S,t) = \begin{cases} (S-E)^+, & \text{knock-out call option;} \\ 0, & \text{knock-in call option;} \\ (E-S)^+, & \text{knock-out put option;} \\ 0, & \text{knock-in put option;} \end{cases}$$

and boundary conditions:

$$V(S_b,t) = \begin{cases} 0, & \text{knock-out option;} \\ V_{vanilla}(S_b,t), & \text{knock-in option;} \end{cases}$$

within the different domains D:

$$D = \begin{cases} \{(S,t)\,|0 \leq S \leq S_b, 0 \leq t \leq T\}, & \text{up option;} \\ \{(S,t)\,|S_b \leq S < \infty, 0 \leq t \leq T\}, & \text{down option;} \end{cases}$$

4.3 Analytical solutions for pricing Barrier options

A full derivation of the formulas is presented in the paper presented by Reiner & Rubinstein (1991) [14]. Presented below are the set of analytical solutions for solving the different types of Barrier options under the Black-Scholes framework, summarised by P. Wilmott (2013) [20]:

Notation:

- $N(.)$**:** the cumulative distribution function for the standard normal distribution $\mathcal{N}(0,1)$ defined in Equation 3.17.
- q**:** the dividend yield on stocks or the foreign interest rate for FX. **Note:** within this thesis, we assume that there is no dividend payout on the underlying, thus this variable is set to $q = 0$ in all computations. It is only included here for completeness.
- S_b**:** the barrier position.
- a **and** b**:**

$$a = \left(\frac{S_b}{S}\right)^{-1+\frac{2(r-q)}{\sigma^2}}, \qquad b = \left(\frac{S_b}{S}\right)^{1+\frac{2(r-q)}{\sigma^2}}$$

- d_* **terms:**

$$
\begin{aligned}
d_1 =& \frac{log\left(\frac{S}{E}\right) + \left(r - q + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}},\\
d_2 =& \frac{log\left(\frac{S}{E}\right) + \left(r - q - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}},\\
d_3 =& \frac{log\left(\frac{S}{S_b}\right) + \left(r - q + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}},\\
d_4 =& \frac{log\left(\frac{S}{S_b}\right) + \left(r - q - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}},\\
d_5 =& \frac{log\left(\frac{S}{S_b}\right) - \left(r - q - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}},\\
d_6 =& \frac{log\left(\frac{S}{S_b}\right) - \left(r - q + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}},\\
d_7 =& \frac{log\left(\frac{SE}{S_b^2}\right) - \left(r - q - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}},\\
d_8 =& \frac{log\left(\frac{SE}{S_b^2}\right) - \left(r - q + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}.
\end{aligned}
$$

Up-and-out call:

$$
\begin{aligned}
C_{up-and-out} =& Se^{-q(T-t)}\left(N\left(d_1\right) - N\left(d_3\right) - b\left(N\left(d_6\right) - N\left(d_8\right)\right)\right)\\
&- Ee^{-r(T-t)}\left(N\left(d_2\right) - N\left(d_4\right) - a\left(N\left(d_5\right) - N\left(d_7\right)\right)\right)
\end{aligned}
\tag{4.3}
$$

Up-and-in call:

$$
\begin{aligned}
C_{up-and-in} =& Se^{-q(T-t)}\left(N\left(d_3\right) + b\left(N\left(d_6\right) - N\left(d_8\right)\right)\right)\\
&- Ee^{-r(T-t)}\left(N\left(d_4\right) + a\left(N\left(d_5\right) - N\left(d_7\right)\right)\right)
\end{aligned}
\tag{4.4}
$$

Down-and-out call:

1. $E > S_b$:

$$
\begin{aligned}
C_{down-and-out} =& Se^{-q(T-t)}\left(N\left(d_1\right) - b\left(1 - N\left(d_8\right)\right)\right)\\
&- Ee^{-r(T-t)}\left(N\left(d_2\right) - a\left(1 - N\left(d_7\right)\right)\right)
\end{aligned}
\tag{4.5}
$$

2. $E < S_b$:

$$
\begin{aligned}
C_{down-and-out} =& Se^{-q(T-t)}\left(N\left(d_3\right) - b\left(1 - N\left(d_6\right)\right)\right)\\
&- Ee^{-r(T-t)}\left(N\left(d_4\right) - a\left(1 - N\left(d_5\right)\right)\right)
\end{aligned}
\tag{4.6}
$$

Down-and-in call:

1. $E > S_b$:

$$\begin{aligned} C_{down-and-in} =& Se^{-q(T-t)}b\left(1-N\left(d_8\right)\right) \\ & - Ee^{-r(T-t)}a\left(1-N\left(d_7\right)\right) \end{aligned} \tag{4.7}$$

2. $E < S_b$:

$$\begin{aligned} C_{down-and-in} =& Se^{-q(T-t)}\left(N\left(d_1\right)-N\left(d_3\right)+b\left(1-N\left(d_6\right)\right)\right) \\ & - Ee^{-r(T-t)}\left(N\left(d_2\right)-N\left(d_4\right)+a\left(1-N\left(d_5\right)\right)\right) \end{aligned} \tag{4.8}$$

Down-and-out put:

$$\begin{aligned} P_{down-and-out} =& - Se^{-q(T-t)}\left(N\left(d_3\right)-N\left(d_1\right)-b\left(N\left(d_8\right)-N\left(d_6\right)\right)\right) \\ & + Ee^{-r(T-t)}\left(N\left(d_4\right)-N\left(d_2\right)-a\left(N\left(d_7\right)-N\left(d_5\right)\right)\right) \end{aligned} \tag{4.9}$$

Down-and-in put:

$$\begin{aligned} P_{down-and-in} =& - Se^{-q(T-t)}\left(1-N\left(d_3\right)+b\left(N\left(d_8\right)-N\left(d_6\right)\right)\right) \\ & + Ee^{-r(T-t)}\left(1-N\left(d_4\right)+a\left(N\left(d_7\right)-N\left(d_5\right)\right)\right) \end{aligned} \tag{4.10}$$

Up-and-out put:

1. $E > S_b$:

$$\begin{aligned} P_{up-and-out} =& - Se^{-q(T-t)}\left(1-N\left(d_3\right)-bN\left(d_6\right)\right) \\ & + Ee^{-r(T-t)}\left(1-N\left(d_4\right)-aN\left(d_5\right)\right) \end{aligned} \tag{4.11}$$

2. $E < S_b$:

$$\begin{aligned} P_{up-and-out} =& - Se^{-q(T-t)}\left(1-N\left(d_3\right)-bN\left(d_8\right)\right) \\ & + Ee^{-r(T-t)}\left(1-N\left(d_2\right)-aN\left(d_7\right)\right) \end{aligned} \tag{4.12}$$

Up-and-in put:

1. $E > S_b$:

$$\begin{aligned} P_{up-and-in} =& - Se^{-q(T-t)}\left(N\left(d_3\right)-N\left(d_1\right)+bN\left(d_6\right)\right) \\ & + Ee^{-r(T-t)}\left(N\left(d_4\right)-N\left(d_2\right)+aN\left(d_5\right)\right) \end{aligned} \tag{4.13}$$

2. $E < S_b$:

$$\begin{aligned} P_{up-and-in} =& - Se^{-q(T-t)}bN\left(d_8\right) \\ & + Ee^{-r(T-t)}aN\left(d_7\right) \end{aligned} \tag{4.14}$$

4.4 Correction formula for Discretely monitored Barrier options

Barrier option pricing models in the previous section work well when the barrier is continuously monitored, meaning knock-in or knock-out events are assumed to happen immediately upon barrier breach. Under this assumption, Merton (1973) [12] was first

able to develop a partial differential equation for pricing knock-out call barrier options. Subsequent work followed by Reiner and Rubinstein (1991) [14], Kunitomo and Ikeda (1992) [21] has contributed to the invention of other analytical solutions defined in the previous section. However, continuous monitoring is not always possible in the real financial market and instead practitioners need to specify how often the barrier is going to be monitored with discrete monitoring. According to Kat and Verdonk (1995) [22], there could be considerable price differences between options with discrete and continuous barrier monitoring, even with daily monitoring. Their research suggested that knock-in options might be cheaper, and knock-out options more expensive, when using discrete monitoring compared to continuous monitoring. This poses challenging issues because there are virtually no analytical solutions available for discretely monitored barrier options due to its complexity and uncertainty from delays in data. Thus practitioners had to continue to use analytical solutions derived from continuous monitoring to price real world discretely monitored barrier options with significant errors.

In order to address this issue, Broadie, Glasserman, and Kou (1997) [23] developed a simple and effective formula for approximating the price of discretely monitored barrier options. This is done by introducing a basic continuity correction to the continuous barrier option formulas, shifting the barrier across each time step to account for discrete monitoring. The adjustment is solely based on the monitoring frequency, asset volatility, and a constant β, approximately equal to 0.5826 [23]. In the original paper it states that

THEOREM 1.1. Let $V_n(S_b)$ be the price of a discretely monitored knock-in or knock-out down call or up put with barrier S_b where n is the monitoring frequency over the period of $[0, T]$. Let $V(S_b)$ be the price of the continuously monitored barrier option. Then

$$V_n(S_b) = V\left(S_b e^{\pm\beta\sigma\sqrt{dt}}\right) + O\left(\frac{1}{\sqrt{n}}\right) \tag{4.15}$$

where $+$ applies if $S_b > S_0$, $-$ applies if $S_b < S_0$, $\beta = -\xi\left(\frac{1}{2}\right)/\sqrt{2\pi} \approx 0.5826$, and ξ is the Riemann zeta function.

Therefore when using the analytical solutions defined in the previous section, the barrier values will take the following new form to improves accuracy of pricing:

$$S_b^* = S_b e^{\pm\beta\sigma\sqrt{dt}} \tag{4.16}$$

5 Asian Options

For Asian options the payoff is based on the average price of the underlying asset over a period of time before expiry, rather than the price at a specific point in time. The average can be calculated over different time intervals, such as the entire life of the option or a specific subset of the option's life [2]. This unique feature results in reduced volatility, leading to a lower cost compared to vanilla European options described in subsection 2.1. The Asian option is said to be **strongly** path-dependent, because it is a high dimensional option. In order to price such option using the Black-Scholes framework, on top of the asset price (S) and time to expiry (t), it needs to keep track of a third independent variable, such as the average price of the underlying asset over a certain period [3].

5.1 PDE for Asian options

5.1.1 state variables

In the general case, we first need to introduce a path-dependent quantity $I(T)$, called the **state variable**, which varies based on the asset's path. This can be expressed as the integral of a certain function $f(S,\tau)$ of the asset over the time period from zero to t:

$$I(t) = \int_0^t f(S,\tau)\, d\tau \tag{5.1}$$

The function $f(S,t)$ can be changed accordingly depending on which aspect of the path-dependency is required. Hence the derivative of the state variable is simply:

$$\begin{aligned} dI =& I(t+dt) - I(t) \\ =& \int_0^{t+dt} f(S,\tau)d\tau - \int_0^t f(S,\tau)\, d\tau \\ =& \left(\int_0^t f(S,\tau)\, d\tau + \int_t^{t+dt} f(S,\tau)d\tau\right) - \int_0^t f(S,\tau)\, d\tau \\ =& \int_t^{t+dt} f(S,\tau)d\tau \\ =& f(S_{t+dt}, t+dt) - f(S_t,t) \\ =& f(S,t)\, dt \end{aligned} \tag{5.2}$$

5.1.2 Derivation

Assuming that the underlying asset evolves according to the same lognormal random walk defined in Equation 3.5. In order to keep track of the state variable in our model, the value of the option need to be redefined as $V(S, I(t), t)$, a function with three variables.

Now construct a delta-hedged portfolio with Π denoting the value of the portfolio, holding one of the path-dependent option and short some quantity Δ of the underlying asset:

$$\Pi = V(S, I, t) - \Delta S \tag{5.3}$$

Across each time step we hold Δ fixed, and by applying Ito's lemma together with results from Equation 5.2, the change in portfolio value is given by:

$$\begin{aligned} d\Pi &= \left(\frac{\delta V}{\delta t} + \mu S\frac{\delta V}{\delta S} + \frac{1}{2}\sigma^2 S^2\frac{\delta^2 V}{\delta S^2}\right)dt + \frac{\delta V}{\delta I}dI + \sigma S\frac{\delta V}{\delta S}dW_t - \Delta\left(\mu S dt + \sigma S dW_t\right) \\ &= \left(\frac{\delta V}{\delta t} + \mu S\frac{\delta V}{\delta S} - \Delta\mu S + \frac{1}{2}\sigma^2 S^2\frac{\delta^2 V}{\delta S^2}\right)dt + f\left(S,t\right)\frac{\delta V}{\delta I}dt + \sigma S\left(\frac{\delta V}{\delta S} - \Delta\right)dW_t \\ &= \left(\frac{\delta V}{\delta t} + \mu S\frac{\delta V}{\delta S} - \Delta\mu S + \frac{1}{2}\sigma^2 S^2\frac{\delta^2 V}{\delta S^2} + f\left(S,t\right)\frac{\delta V}{\delta I}\right)dt + \sigma S\left(\frac{\delta V}{\delta S} - \Delta\right)dW_t \end{aligned} \tag{5.4}$$

Set $\Delta = \frac{\delta V}{\delta S}$ and sub back into Equation 5.4 to eliminate risk:

$$d\Pi = \left(\frac{\delta V}{\delta t} + \frac{1}{2}\sigma^2 S^2\frac{\delta^2 V}{\delta S^2} + f\left(S,t\right)\frac{\delta V}{\delta I}\right)dt \tag{5.5}$$

By following the no risk-free arbitrage opportunities assumption, the portfolio must earn at the risk-free interest rate r:

$$d\Pi = r\Pi dt \tag{5.6}$$

Equating the two equations from Equation 5.5 and Equation 5.6:

$$\begin{aligned} \left(\frac{\delta V}{\delta t} + \frac{1}{2}\sigma^2 S^2\frac{\delta^2 V}{\delta S^2} + f\left(S,t\right)\frac{\delta V}{\delta I}\right)dt &= r\Pi dt \\ \frac{\delta V}{\delta t} + \frac{1}{2}\sigma^2 S^2\frac{\delta^2 V}{\delta S^2} + f\left(S,t\right)\frac{\delta V}{\delta I} &= r\left(V - S\frac{\delta V}{\delta S}\right) \end{aligned} \tag{5.7}$$

Finally by rearranging, the general PDE for an Asian option becomes:

$$\frac{\delta V}{\delta t} + \frac{1}{2}\sigma^2 S^2\frac{\delta^2 V}{\delta S^2} + f\left(S,t\right)\frac{\delta V}{\delta I} + rS\frac{\delta V}{\delta S} - rV = 0 \tag{5.8}$$

5.2 Asian option types

How the average should be calculated in an Asian option can be flexible depending on the buyer's need. This means focusing on defining the function $f(S,t)$ mentioned in Equation 5.1 of the state variable. Below are some commonly used approaches and their corresponding PDEs.

Arithmetic Asian:
set $f(S,t) = S(t)$, the state variable becomes:

$$I = \int_0^t S(\tau)d\tau \tag{5.9}$$

Substitute into Equation 5.8 the resulting PDE is:

$$\frac{\delta V}{\delta t} + \frac{1}{2}\sigma^2 S^2 \frac{\delta^2 V}{\delta S^2} + S\frac{\delta V}{\delta I} + rS\frac{\delta V}{\delta S} - rV = 0 \tag{5.10}$$

Geometric Asian:
set $f(S,t) = log(S(t))$, the state variable becomes:

$$I = \int_0^t log(S(\tau))d\tau \tag{5.11}$$

Substitute into Equation 5.8 the resulting PDE is:

$$\frac{\delta V}{\delta t} + \frac{1}{2}\sigma^2 S^2 \frac{\delta^2 V}{\delta S^2} + log(S)\frac{\delta V}{\delta I} + rS\frac{\delta V}{\delta S} - rV = 0 \tag{5.12}$$

Note the above state variables consist of integrals of the asset price over the averaging period, which makes them **continuously sampled averages**. Alternatively, we could take only data points at specific time intervals, such as summing the closing prices of each day and divide by the total number of days, which are thought to be more reliable, known as **discretely sampled averages**. In real world practice, the latter form of sampling is adopted because data can be unreliable due to external factors such as network delays, and the exact timing of when a trade took place may not be known precisely. Therefore from a legal point of view, it is preferred to use only discrete monitoring, in other words key prices such as the closing price which can be guaranteed to be a genuine traded price, to calculate the averages of Asian options [3]. However, the pricing of Asian options with discretely sampled averages typically requires the use of numerical methods, as closed-form solutions are not available [24].

After defining the average function for calculating the respective average price A, Asian options can be further categorised depending on its payoff function. For example, when compared with the payoff function of a vanilla European Call option [20]:

$$C = max(S - E, 0) \tag{5.13}$$

average strike (floating strike) call:
Replace the strike price E with the average price A, payoff is

$$C = max(S - A, 0) \tag{5.14}$$

average strike (floating strike) put:

Again, replace the strike price E with the average price A, payoff is

$$P = max(A - S, 0) \tag{5.15}$$

average rate (fixed strike) call:
Replace the expiry price S with the average price A, payoff is

$$C = max(A - E, 0) \tag{5.16}$$

average rate (fixed strike) put:
Again, replace the expiry price S with the average price A, payoff is

$$P = max(E - A, 0) \tag{5.17}$$

5.3 Analytical solution for pricing Asian options

As previously mentioned, due to the intricate dependence of Asian option valuation on both the final price at expiration and the underlying asset's price path during the averaging period, only a limited number of closed-form analytical solutions exist. Notably, true closed-form analytical solutions for pricing Asian options are scarce, with most being closed-form approximations. As a consequence, numerical methods like Monte Carlo [25] are the primary approach for pricing them.

Nonetheless, specific conditions permit the availability of closed-form solutions for pricing continuously sampled geometric average rate Asian options. This stems from the lognormal distribution of the geometric average of lognormal random variables [3]. This characteristic is pivotal, allowing mathematicians to employ standard mathematical techniques associated with lognormal distributions for constructing the closed-form solution.

Kemna and Vorst (1990)
One of the earliest research effort in this field is work done by Kemna and Vorst (1990) [26], where they developed Black-Scholes style expressions for estimating the value of geometric averaging Asian call and put options, subject to two conditions:

1. The volatility must be replaced with $\sigma/\sqrt{3}$.
2. The dividend yield must be replaced with $D + \sigma^2/6$.

The estimating solutions are given by:

$$C \approx Se^{-r(b-r)(T-t)}N\left(d_1\right) - Ke^{-r(T-t)}N\left(d_2\right) \tag{5.18}$$

and

$$P \approx -Se^{(b-r)(T-t)}N\left(-d_1\right) + Ke^{(T-t)}N\left(-d_2\right) \tag{5.19}$$

where,

$$\begin{aligned} d_1 =& \frac{ln\left(\frac{S}{K}\right) + \left(b + \frac{\sigma_A^2}{2}\right)T}{\sigma_A\sqrt{T}} \\ d_2 =& \frac{ln\left(\frac{S}{K}\right) + \left(b - \frac{\sigma_A^2}{2}\right)T}{\sigma_A\sqrt{T}} \\ =& d_1 - \sigma_A\sqrt{T} \end{aligned}$$

and

$\sigma_A = \frac{\sigma}{\sqrt{3}}$ is the adjusted volatility, with σ being the observed volatility

$b = \frac{1}{2}\left(r - D - \frac{\sigma^2}{6}\right)$ as the adjusted dividend payout, which is assumed to be 0 in this thesis.

Values obtained from Kemna and Vorst's method will be used as a benchmark in this thesis to compare with the results obtained from Monte Carlo method.

Continuously Monitored Geometric Averaging Options

There are also other closed form formulas for continuously monitored geometric averaging options where the full derivation of these formulas can be found in the book written by Kwok (2001) [4]. Below are two examples formulas summarised by Wilmott (2013) [20].

Geometric average rate call:

$$C = e^{-r(T-t)}\left(Gexp\left(\frac{\left(r - D - \frac{\sigma^2}{2}\right)(T-t)^2}{2T} + \frac{\sigma^2(T-t)^3}{6T^2}\right)N(d_1) - EN(d_2)\right) \tag{5.20}$$

Geometric average rate put:

$$P = e^{-r(T-t)}\left(EN(-d_2) - Gexp\left(\frac{\left(r - D - \frac{\sigma^2}{2}\right)(T-t)^2}{2T} + \frac{\sigma^2(T-t)^3}{6T^2}\right)N(d_1)\right) \tag{5.21}$$

where,

$$
\begin{aligned}
I =& \int_0^t log\left(S\left(\tau\right)\right) d\tau, \\
G =& e^{\frac{I}{T}} S^{\frac{(T-t)}{T}}, \\
d_1 =& \frac{Tlog\left(\frac{G}{E}\right) + \left(r - D - \frac{\sigma^2}{2}\right)\frac{(T-t)^2}{2} + \sigma^2\frac{(T-t)^3}{3T}}{\sigma\sqrt{\frac{(T-t)^3}{3}}}, \\
d_2 =& \frac{Tlog\left(\frac{G}{E}\right) + \left(r - D - \frac{\sigma^2}{2}\right)\frac{(T-t)^2}{2}}{\sigma\sqrt{\frac{(T-t)^3}{3}}}
\end{aligned}
\tag{5.22}
$$

and D is the dividend payout of the underlying

Aside from the technical details, Asian options are popular in the market due to their ability to lower the likelihood of market manipulation close to the expiry date, and provide more hedging opportunities for businesses with a series of exposures [27]. For example businesses involving thinly traded commodities such as rare earth metals and agricultural products.

6 Monte Carlo Simulations

Complex mathematical problems often lack closed-form solutions, particularly when dealing with exotic options that possess unique features, as discussed in previous chapters. In such cases, numerical methods offer an approach to estimate the solution through approximations and iterative processes. [28]. This chapter will be based on understanding the concepts of the Monte Carlo method, which is a type of numerical method and explore its application to exotic option pricing.

During World War II, John von Neumann and Stanislaw Ulam collaborated to develop the Monte Carlo Method as a means of enhancing decision making in situations where outcomes were uncertain. They have named this method after a well-known casino destination, Monaco, as probability plays a central role in the simulation process, much like in games of roulette [29]. The Monte Carlo technique essentially calculates the expected value of estimates by conducting random sampling of a specific random variable [30]. This expected value relies on the solution to a stochastic differential equation (SDE). The initial proposal to employ the Monte Carlo method for valuing European options was put forth by Boyle in 1977 [31]. To obtain a thorough understanding of the Monte Carlo techniques applied in option pricing, one may consult authoritative references, such as those by Boyle, Broadie, and Glasserman (1997)[32], which delve into methods for reducing Monte Carlo errors by applying different variance reduction methods, as investigated in the studies by Glasserman (2004)[33] and later extended upon by Asmussen and Glynn (2007)[34]. Furthermore, the book written by Kloeden and Platen (1992)[35] explains in detail how numerical solutions can be used for solving stochastic differential equations in general. Similar materials on the topic can also be found the book by Milstein and Tretyakov (2004)[36].

6.1 Basic concepts

MC simulation is based on the idea of using random numbers to solve complex mathematical problems. The method involves simulating a large number of possible outcomes, each with a different set of randomly generated inputs. By repeating this process many times, the simulation produces a distribution of possible outcomes, allowing for the calculation of probabilities and expected values. Based on the law of large numbers, the accuracy of MC methods can be improved by simply increasing the number of simulations because the estimated results should converge to the real value [33].

Suppose there is some arbitrary function $g(X)$ where X is a random variable, with an unknown mean and variance, which we denote as $\mathbb{E}[g(X)] = \mu$ and $var\,[g(X)] = \sigma^2$. In order to estimate the value of the mean a, we can draw M independent random sample values $X_1, X_2, X_3, ..., X_M$ from the same distribution as X using a probability density function $f(x)$, and calculate an estimated value called sample mean μ_* by:

$$\mu_* = \frac{1}{M}\sum_{i=1}^{M} g(X_i) \tag{6.1}$$

Due to the law of large numbers, as we increase our sample points in the estimation, it follows that:

$$\frac{1}{M}\sum_{i=1}^{M} g(X_i) \to \mathbb{E}[g(X)], \text{ or } \mu_* \to \mu, \text{ with probability 1, as } M \to \infty$$

Thus our sample mean μ_* gradually converges to the real mean μ given sufficiently large number of samples. For the variance, since $var[g(X)] = \mathbb{E}\Big[(g(X) - \mathbb{E}[g(X)])^2\Big]$, we can estimate an unbiased sample variance value σ_*^2 by:

$$\sigma_*^2 = \frac{1}{M-1}\sum_{i=1}^{M} (g(X_i) - \mu_*)^2$$

The approximations made by repeatedly drawing random samples from the same distribution as the target random variable X is indeed a typical use of the Monte Carlo method. Subsequently, by the Central Limit Theorem, we could say that $\mu_* - \mu$ behaves similar to a random variable with distribution $N\left(0, \frac{\sigma^2}{M}\right)$. The **standard error** resulted from this approximation is its standard deviation, more commonly referred to as $\varepsilon = \frac{\sigma}{\sqrt{M}}$. This will be an important indicator to measure the accuracy of the simulated result. Thus as n increases, the variance becomes smaller and the error of approximation reduces [37].

The implementation of the MC method is very straightforward to code, and we can make adjustments to how the payoffs should be calculated for different exotic options. For example when pricing Barrier Options, we can examine whether each realisation has hit the barrier and decide whether the contract should be nullified or not. For Asian options on the other hand, we can keep track of the full path of each realisation and calculate the required averages to be used in the final payoffs. However the disadvantages of the MC method are also very apparent. In order to get something close to the accurate answer, tens of thousands of realisations need to be run, which is time consuming [3]. Furthermore, the Greeks in the Black-Scholes equation are calculated by taking the partial derivatives of the option price with respect to some parameters, such as asset price, time, volatility and interest rate. In MC simulations, the partial derivatives are estimated by making small changes to the simulated inputs and taking the average, which can be computationally expensive and inaccurate [38]. This makes it harder to get accurate estimates of the Greeks.

6.2 Monte Carlo recipe

The MC method is simple and yet very powerful for pricing European style options. The concept readily carries over to exotic and path-dependent contracts. The key steps required are simply:

1. Simulate the risk-neutral random walk starting at today's value over the required time frame, which gives one realisation of the price path of the underlying asset.
2. Calculate the payoff of the option for this realisation.
   (a) Barrier option:
   (b) Sub-item 2
3. Repeat many more realisations of the asset's price movement over the same time frame.
4. Estimate the average payoff of all the realisations.
5. Calculate the present value discounting the average payoff and that will be the option's value today.

In mathematical terms, when valuing the price of options using Monte Carlo method, we simulate the underlying under the risk-neutral measure and discount the expected payoff depending on the option type. The general option's value can be written in the form:

$$V(S,t) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}[Payoff(S')] \tag{6.2}$$

where $\mathbb{Q}$ represents the risk-neutral density, and

$$\mathbb{E}^{\mathbb{Q}}[Payoff(S')] = \int_0^{\infty} \tilde{p}\left(S,t;S',T\right) P\left(S'\right) dS'$$

with $\tilde{p}\left(S,t;S',T\right)$ representing the probability density function.

6.3 Generating sample paths

Following the recipe above, the first step is to simulate sample paths of the underlying price movement. For vanilla European options, this step is not necessary, because under the Black-Scholes framework, the closed form solution only requires the price at maturity to calculate an exact value of the option by discounting the value of the final payoff. How the price has evolved across the time period is irrelevant. However, for pricing Barrier and Asian options, their payoffs are path-dependent, hence the whole sample path between current and maturity date is required.

In order to simulate the price movement, we assume that the underlying asset follows a GBM defined in Equation 3.2, but with the risk-free interest rate as its growth rate, written as:

$$dS_t = rS_t dt + \sigma S_t dW_t \tag{6.3}$$

There are many different methods for approximating the stochastic process described by the GBM above, this thesis will focus on three approaches: Euler Scheme, Forward Euler-Maruyama method and the Milstein method. The full derivation of each method can be found in the paper by Rouah (2011) [39].

6.3.1 Euler Scheme

For this method, we first assume a very special case, the value of the option contract $V(S)$ changes by the function $V(S) = log(S)$. Also notice, $V(S)$ is one dimensional and the function value only depends on the asset price S, and independent of time. Hence we have the following:

$$\frac{\delta V}{\delta S} = \frac{1}{S}, \quad \frac{\delta^2 V}{\delta S^2} = -\frac{1}{S^2}, \quad \frac{\delta V}{\delta t} = 0$$

Apply Ito's lemma defined in Equation 3.6 to work out dV:

$$\begin{aligned} dV = d\left(logS\right) &= \left(\mu S\left(\frac{1}{S}\right) + \frac{1}{2}\sigma^2 S^2\left(-\frac{1}{S^2}\right)\right)dt + \left(\sigma S\left(\frac{1}{S}\right)\right)dW_t \\ &= \left(\mu - \frac{1}{2}\sigma^2\right)dt + \sigma dW_t \end{aligned}$$

Let $W_t \sim \phi\sqrt{t}$, where $\phi \sim N(0,1)$ is a standard normal random variable. Integrate over $[t, t+\delta t]$ and by using Euler Scheme, also known as the Euler discretisation method, find the solution in discrete time stepping form:

$$\begin{aligned} \int_t^{t+\delta t} d\left(logS\right) &= \int_t^{t+\delta t}\left(\mu - \frac{1}{2}\sigma^2\right)d\tau + \int_t^{t+\delta t}\sigma dW_\tau \\ log\left(S_{t+\delta t}\right) - log\left(S_t\right) &= \left(\mu - \frac{1}{2}\sigma^2\right)\delta t + \sigma\left(W_{t+\delta t} - W_t\right) \\ log\left(S_{t+\delta t}\right) &= log\left(S_t\right) + \left(\mu - \frac{1}{2}\sigma^2\right)\delta t + \sigma\phi\sqrt{\delta t} \end{aligned}$$

exponentiating both sides gives the following expression:

$$S_{t+\delta t} = S_t e^{\left(\mu - \frac{1}{2}\sigma^2\right)\delta t + \sigma\phi\sqrt{\delta t}} \tag{6.4}$$

which is an exact solution that can be used to simulate asset path.

6.3.2 Forward Euler-Maruyama method

Consider a stochastic process of the form:

$$dS_t = \mu(S_t, t)dt + \sigma(S_t, t)dW_t \tag{6.5}$$

then simulate the asset price S_t over the time interval $[0, T]$ by discretising the time interval from $0 = t_1 < t_2 < ... < t_m = T$, such that $t_i - t_{i-1} = dt$. Thus when we integrate dS_t over the interval $[t, t + \delta t]$, the equation becomes:

$$S_{t+\delta t} = S_t + \int_t^{t+\delta t} \mu(S_\tau, \tau)d\tau + \int_t^{t+\delta t} \sigma(S_\tau, \tau)dW_\tau \tag{6.6}$$

Applying Euler scheme gives:

$$S_{t+\delta t} = S_t + \mu(S_t, t)dt + \sigma(S_t, t)\phi\sqrt{dt} \tag{6.7}$$

Thus apply Euler-Maruyama to our GBM in Equation 6.3, the resulting forward Euler-Maruyama method is:

$$\begin{aligned} S_{t+\delta t} =& S_t + rS_t\delta t + \sigma S_t\phi\sqrt{\delta t} \\ =& S_t\left(1 + r\delta t + \sigma\phi\sqrt{\delta t}\right) \end{aligned} \tag{6.8}$$

6.3.3 Milstein Method

Take the exact solution obtained in Equation 6.4, apply Taylor expansion to the exponential term:

$$e^{\left(\mu - \frac{1}{2}\sigma^2\right)\delta t + \sigma\phi\sqrt{\delta t}} \sim 1 + \left(r - \frac{1}{2}\sigma^2\right)\delta t + \sigma\phi\sqrt{\delta t} + \frac{1}{2}\sigma^2\phi^2\delta t \tag{6.9}$$

gives:

$$S_{t+\delta t} \sim S_t\left(1 + r\delta t + \sigma\phi\sqrt{\delta t} + \frac{1}{2}\sigma^2\left(\phi^2 - 1\right)\delta t + ...\right) \tag{6.10}$$

which we can use to simulate the asset path. Furthermore, if we compare with the Euler-Maruyama method in Equation 6.8, we see that there is an extra term $\frac{1}{2}\sigma^2\left(\phi^2 - 1\right)\delta t$. The term

$$\frac{1}{2}\left(\phi^2 - 1\right)\delta t \tag{6.11}$$

is called the Milstein correction term.

6.4 Confidence interval

As well as measuring the standard error ε resulted from the simulations, another way of determining the reliability of the result is to look at its confidence interval. Since MC method is based on probability, where the underlying is generated from a pool of normally distributed samples, the end results in theory may have large variances, and

we cannot bound the error exactly. Instead we could impose a confidence interval, which bounds the error within a 95% range.

Given that for a random variable Y, such that $(Y - \mu)/\sigma \sim N(0, 1)$, its confidence interval is defined as [37]:

$$\mathbb{P}(\mu - 1.96\sigma \leq Y \leq \mu + 1.96\sigma) = 0.95 \tag{6.12}$$

When using MC, the option value V is expected to have a distribution of $N\left(0, \frac{\sigma^2}{M}\right)$. Thus substituting into Equation 6.12:

$$\mathbb{P}\left(-1.96\sqrt{\frac{\sigma^2}{M}} \leq V \leq 1.96\sqrt{\frac{\sigma^2}{M}}\right) = 0.95 \tag{6.13}$$

From the equation, we observe that the width of the confidence interval is inversely proportional to $\sqrt{M}$, hence as we increase the number of simulations M, the confidence interval should becomes smaller. In fact in order to shrink the interval by a factor of 10, the MC method requires 100 times as many samples, which can be computationally expensive. Another approach to shrink the interval is to reduce variance, covered in the next section.

6.5 Variance Reduction

The Monte Carlo method is inefficient because as observed, the standard error

$$\varepsilon = \frac{\sigma}{\sqrt{M}} \tag{6.14}$$

is proportional to $\sqrt{var(X_i)}$, and requires large number of simulations to reduce its variance, which can be computationally expensive. Therefore it order to minimise the error and help the simulations to converge faster, variance reduction techniques have been invented. This thesis will make use of the antithetic variate method.

6.5.1 antithetic variate

In this technique, we compute two approximations for the option value utilising a single set of random numbers. This is achieved by drawing a set of random samples from the normal distribution to generate a realization of the asset price path, along with the associated option payoff and its present value. Subsequently, the same set of random numbers is utilised, but with their signs reversed, effectively replacing the variable ϕ with its negative counterpart, $-\phi$. Another realization is then simulated using this set of negative counterpart, and the option payoff and its present value are calculated accordingly [20]. Then, the option's value is computed from averaging the results of these two present values. Through iterative application of this process, a precise and dependable estimate for the option value can be achieved [3].

The effectiveness of this technique stems from the symmetry inherent in the Normal distribution, if $\phi^{(n)} \sim N(0,1)$, then $-\phi^{(n)}$ also has a standard Normal distribution. This is a property which the antithetic variables were based on [37]. Hence, the utilization of antithetic variables enables the generation of pairs of sample paths that exhibit mirror-like behavior, resulting in a reduction of variance and leading to more accurate estimations of the option value.

In the case of pricing path-dependent options, if we update each price increment with a $N(0,1)$ random variable ϕ_j coming from the *i.i.d.* sequence $\{\phi_0, \phi_1, \phi_2, ...\phi_{n-1}\}$ represented by Φ_1. Also the negative variate $\{-\phi_0, -\phi_1, -\phi_2, ... - \phi_{n-1}\}$ represented by Φ_2. Denote the payoff function as $U(.)$. The estimated payoff for the antithetic variate method becomes:

$$\tilde{Payoff} = \frac{1}{n}\sum_{i=1}^{n}\frac{U(\phi_1) + U(\phi_2)}{2} \tag{6.15}$$

The variance of the antithetic method can be calculated by:

$$var\left[\frac{U(\Phi_1) + U(\Phi_2)}{2}\right] = \frac{1}{4}\left(var(U(\Phi_1)) + var(U(\Phi_2)) + 2cov(U(\Phi_1), U(\Phi_2))\right) \tag{6.16}$$

which will be smaller than the variance of standard method $var\left[U(\Phi_1)\right]$, because since $\Phi_2 = -\Phi_1$, their covariance $cov(U(\Phi_1), U(\Phi_2))$ will be negative.

The distribution of the pairs $(\Phi_1, \Phi 2)$ with size of n samples each, exhibits more regularity compared to a collection of $2n$ independent samples, where the sample mean over the antithetic pairs consistently matches the population mean of 0. As a result, the data set demonstrates lower variance.

7 Results, Analysis and Evaluation

This chapter presents a comprehensive series of computational results employing the Monte Carlo method for pricing Barrier and Asian options. A thorough analysis of the efficacy and precision of these outcomes will be provided.

To help differentiate the results, for asset paths which did not make use of variance reduction techniques, they are referred to as plain realisations or crude Monte Carlo. "Euler Scheme", "Euler Maruyama" and "Milstein" columns each holds the option value obtained with the respective random walk models. "Error" columns are referred to the standard error of Monte Carlo method. "Time" columns refers to the runtime of the corresponding simulations. To simplify calculation, assuming all option contracts starts on the first day of year 2024, and ends on the last day, with each day being a trading day, thus number of monitoring frequency or time steps $N = 365$, with M denoting the number of simulations.

7.1 Barrier Options

Example: up-and-in call option
Consider pricing an up-and-in call option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, S_u = 110, S_u^* = 110.6772, E = 105$, $t_0 = 01/01/2024$, $T = 31/12/2024$, $N = 365$.

Using a closed formula defined in Equation 4.3, we are able to obtain the exact price of the Barrier option in Table 3, under the assumption of continuous monitoring, with the corrected up Barrier value of $S_u^* = 110.6772$.

Exact value
7.1055

Table 3: Exact value of up-and-in Call Barrier option

In Table 4 is a set of results obtained from Monte Carlo method. In this case, models of the three different random walks with plain realisations were used, showing their respective standard errors and runtimes. To ensure a fair comparison between the models, across each time step, the underlying movement has been updated by the same random variable ϕ_i. It can be observed that as number of simulations M increases, the estimated option price decreases from approximately 7.3 down to 7.09, getting closer and closer to the exact value.

Table 5 displays the estimated values obtained using antithetic variates method. The random walk models have also used the same set of random variables as with the plain realisations in Table 4 to ensure a fair comparison.

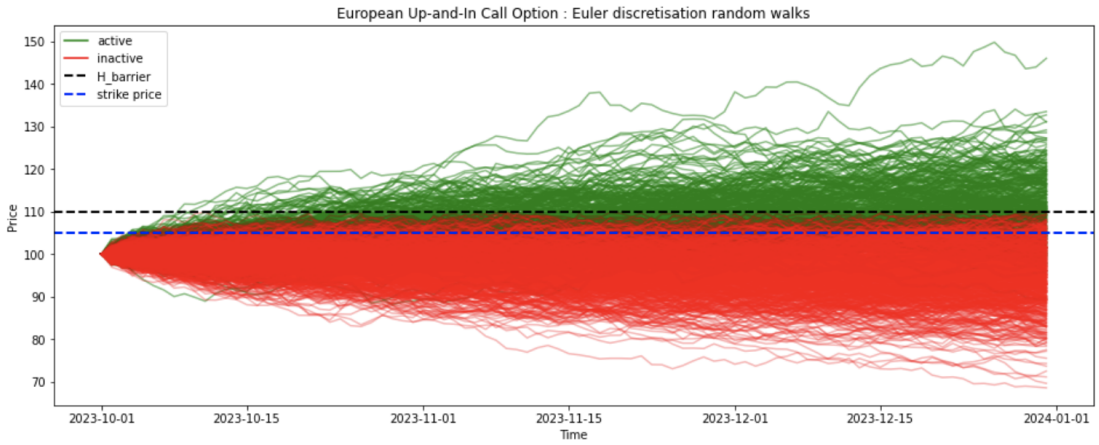


Figure 1: Monte Carlo simulation of up-and-in Call option

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	7.3361	1.17796	0.0068	7.3124	1.17361	0.0016	7.3353	1.17784	0.0024
250	8.0450	0.88356	0.0035	8.0445	0.88335	0.0012	8.0442	0.88347	0.0020
500	6.9126	0.56032	0.0081	6.9152	0.56018	0.0065	6.9119	0.56026	0.0062
750	7.5408	0.45783	0.0142	7.5512	0.45807	0.0040	7.5400	0.45778	0.0051
1000	7.7562	0.41264	0.0162	7.7626	0.41227	0.0057	7.7554	0.41260	0.0066
2500	6.8556	0.24776	0.0389	6.8512	0.24775	0.0146	6.8549	0.24774	0.0221
5000	7.2991	0.18025	0.1466	7.2972	0.18010	0.0317	7.2984	0.18023	0.0554
7500	7.0519	0.14458	0.4341	7.0512	0.14449	0.0503	7.0512	0.14457	0.0655
10000	7.2341	0.12742	0.1918	7.2336	0.12737	0.0825	7.2334	0.12740	0.0874
25000	7.0314	0.07930	0.5213	7.0310	0.07927	0.2310	7.0307	0.07929	0.2801
50000	7.1689	0.05640	0.9893	7.1685	0.05638	0.4349	7.1682	0.05640	0.6582
75000	7.1440	0.04570	1.2655	7.1441	0.04568	0.6715	7.1433	0.04569	0.9131
100000	7.0941	0.03938	1.6225	7.0940	0.03936	0.9272	7.0934	0.03938	1.3184

Table 4: Pricing of up-and-in Call option with crude Monte Carlo

By comparing the figures from the two tables, it appears that for the same random walk model, using antithetic variates method has helped the estimated price to be more stable and converges to the true value more quickly. For example, consider the Milstein model. When using plain realisations, it first obtained a close estimate at $M = 7500$, with a value of 7.0512. Whereas when using the antithetic variates reduction technique, a better approximate value of 7.1011 was achieved at $M = 5000$. Similar conclusions can be drawn for the other models, which proves that variance reduction technique does indeed allow Monte Carlo method to converge faster.

Another observation from the results is that there are only minute differences between the estimated values by the three random walk models when the same set of random variables were used. Thus a key finding in this case is when using Monte Carlo method to price options, the choice of the random walk model may have little effect on the final outcome, and perhaps practitioners should focus more on applying variance reduction techniques.

In Figure 2, Figure 3 and Figure 4, a plot of the confidence level for each random walk

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	6.3359	0.64786	0.0026	6.3272	0.64573	0.0010	6.3353	0.64781	0.0013
250	7.6119	0.50358	0.0036	7.6092	0.50341	0.0023	7.6111	0.50353	0.0036
500	7.1962	0.33642	0.0104	7.1957	0.33611	0.0063	7.1955	0.33639	0.0104
750	7.1380	0.26230	0.0113	7.1430	0.26218	0.0078	7.1373	0.26227	0.0104
1000	7.2509	0.23101	0.0143	7.2554	0.23080	0.0100	7.2502	0.23099	0.0150
2500	6.9221	0.14252	0.0395	6.9193	0.14251	0.0264	6.9214	0.14250	0.0444
5000	7.1018	0.10341	0.0851	7.1003	0.10335	0.0554	7.1011	0.10340	0.0850
7500	7.1884	0.08518	0.1297	7.1876	0.08512	0.0956	7.1877	0.08517	0.1355
10000	7.1517	0.07375	0.1572	7.1512	0.07369	0.1109	7.1510	0.07375	0.1671
25000	7.0373	0.04645	0.3866	7.0369	0.04643	0.2722	7.0365	0.04645	0.5347
50000	7.1493	0.03268	1.3931	7.1482	0.03266	1.0032	7.1486	0.03268	2.1447
75000	7.1079	0.02666	2.3588	7.1079	0.02664	1.2882	7.1072	0.02666	4.0066
100000	7.0940	0.02297	2.9149	7.0935	0.02295	3.1759	7.0933	0.02296	4.6564

Table 5: Pricing of up-and-in Call option with antithetic variates

model is displayed. The grey shaded areas indicate the 95% confidence interval of the crude Monte Carlo method with plain realisations, with the black dots representing the estimated values. The orange shaded area represent 95% confidence interval of Monte Carlo with antithetic variate technique, and the red dots are the respective estimated option price. From the plots, it is clear that antithetic variate is a very effective method of reducing variance, as the bounds of its confidence interval is much smaller than that of crude Monte Carlo method. For all three random walk models, the exact value from closed form solution successfully lies within the bounds of their confidence interval. This visualisation further proves that the three random walk models are almost equally as good for simulating the underlying movement.

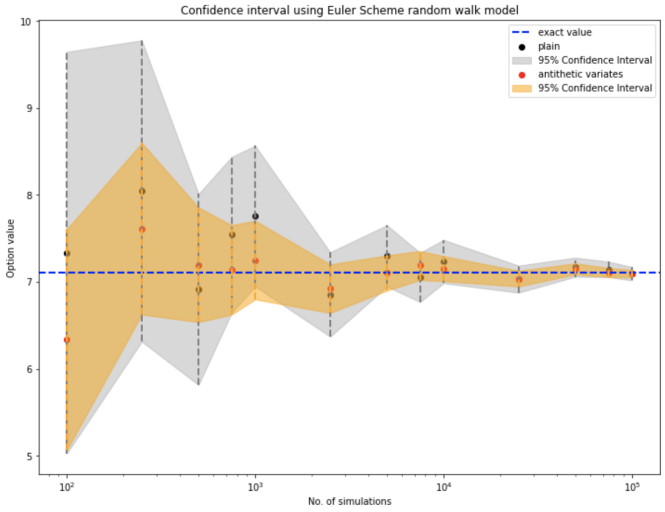


Figure 2: Confidence Interval of Euler Scheme model for up-and-in Barrier option

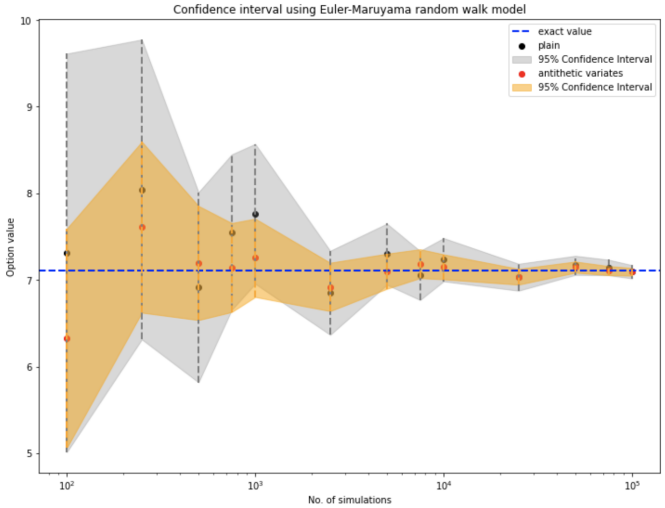


Figure 3: Confidence Interval of Euler Maruyama model for up-and-in Barrier option

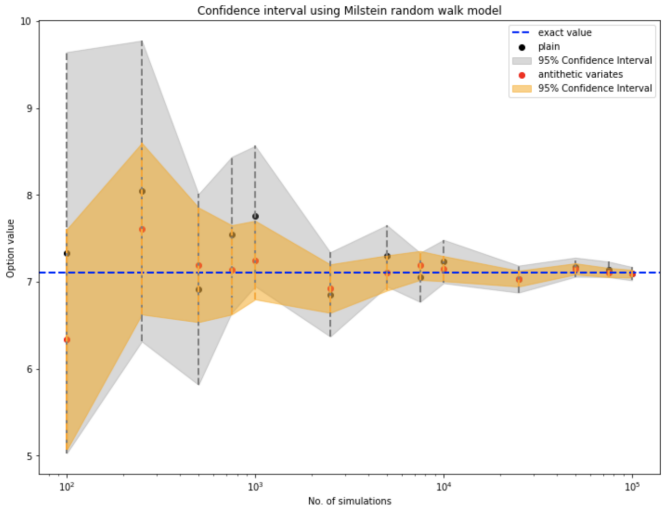


Figure 4: Confidence Interval of Milstein model for up-and-in Barrier option

Similar observations can be concluded from the simulation results for other 7 types of barrier options, which are available in the appendix under subsection A.1.

7.2 Asian Options

Example: geometric average rate Call option - average of full length of contract

This example shows the pricing of an Geometric average rate Call option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, E = 105$, $T_0 = 01/01/2024$, $T_1 = 01/01/2024$, $T_2 = 31/12/2024$ and $N = 365$, where T_0 is the contract start date, T_1 is

the start date of the period from which to calculate the average price, T_2 is the expiration date, where the averaging period also ends. Thus in this example, average taken is the average price across the whole year, or the full lifetime of the contract.

This example is chosen so that we can make use of the exact closed form solution defined in Equation 5.18, and the exact value is shown in Table 6.

Exact value
2.9849

Table 6: Exact value of geometric average rate call option

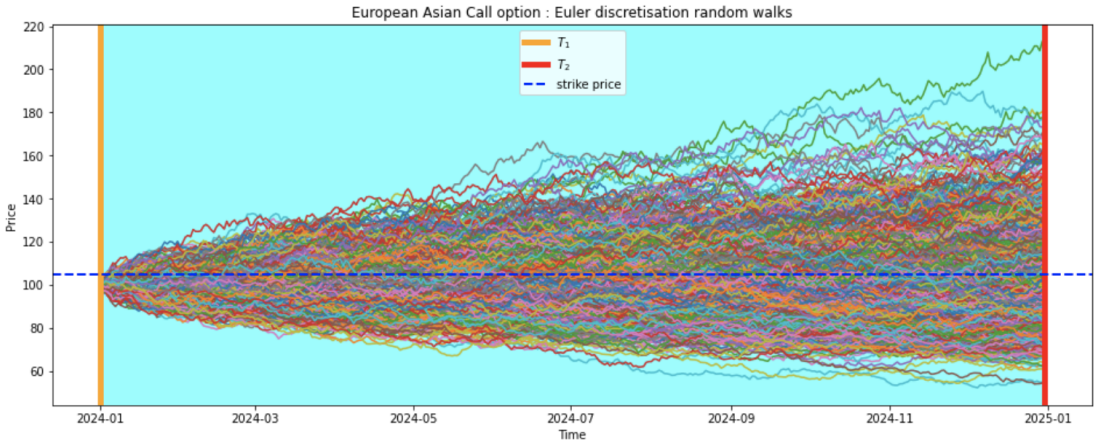


Figure 5: Monte Carlo simulation of geometric average rate Call option

Figure 5 is a plot of the Monte Carlo simulations using Euler Scheme random walk model. The orange band labelled as T_1 denotes the start date of the averaging period. The red band labelled as T_2 denotes the end date of the averaging period. The highlighted area in cyan colour is to help with visualising the effective time interval for calculating the geometric average price.

Table 7 shows the option value estimated by crude Monte Carlo method. It appears that with sufficient number of simulations, the estimated value does also converge to the exact value with closed form solution. Once again, values obtained from different random walk models show very slight differences, which suggests that for the pricing of geometric average rate call option, the choice of random walk has little impact on the final outcome.

Table 8 on the other hand displays the estimated value with antithetic variate technique applied. It seems that with $M = 750$, the simulation has already converged to some relatively accurate value of 2.9037, 2.9017 and 2.9034 from the three random walk models. Whereas for crude Monte Carlo, similar accuracy was achieved when $M = 1000$.

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	2.1525	0.47836	0.0035	2.1531	0.47860	0.0014	2.1523	0.47831	0.0016
250	2.7379	0.34885	0.0053	2.7322	0.34837	0.0017	2.7376	0.34882	0.0043
500	3.1371	0.27773	0.0119	3.1342	0.27763	0.0064	3.1367	0.27770	0.0051
750	2.8139	0.20103	0.0135	2.8122	0.20080	0.0071	2.8135	0.20100	0.0083
1000	2.8906	0.17842	0.0166	2.8903	0.17843	0.0075	2.8903	0.17840	0.0085
2500	3.2076	0.12140	0.0438	3.2073	0.12134	0.0218	3.2072	0.12139	0.0275
5000	3.0177	0.08419	0.1018	3.0171	0.08415	0.0503	3.0173	0.08419	0.0559
7500	2.9533	0.06519	0.1910	2.9532	0.06516	0.0675	2.9530	0.06518	0.0765
10000	2.9803	0.05811	0.1825	2.9796	0.05807	0.0930	2.9799	0.05810	0.1459
25000	2.9928	0.03701	0.4715	2.9921	0.03699	0.2367	2.9924	0.03701	0.2948
50000	3.0103	0.02625	1.0332	3.0100	0.02624	0.6747	3.0099	0.02625	0.9024
75000	2.9819	0.02139	1.5308	2.9816	0.02137	1.0966	2.9815	0.02138	1.3826
100000	2.9827	0.01852	2.0763	2.9820	0.01851	1.3152	2.9824	0.01852	1.8602

Table 7: Pricing of geometric average rate call option with crude Monte Carlo

Meanwhile the runtime for crude Monte Carlo and antithetic variate methods are relatively similar. This observation agrees with the fact that antithetic variate does indeed improve the efficiency of Monte Carlo method.

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	2.7454	0.30910	0.0029	2.7471	0.30954	0.0021	2.7451	0.30907	0.0038
250	3.5352	0.25368	0.0065	3.5310	0.25318	0.0048	3.5347	0.25366	0.0045
500	3.1747	0.16898	0.0102	3.1749	0.16888	0.0077	3.1743	0.16896	0.0094
750	2.9037	0.12693	0.0166	2.9017	0.12679	0.0112	2.9034	0.12692	0.0135
1000	2.9173	0.11129	0.0182	2.9174	0.11120	0.0138	2.9170	0.11128	0.0178
2500	3.0791	0.07167	0.0562	3.0788	0.07164	0.0428	3.0788	0.07167	0.0526
5000	3.0068	0.05115	0.1729	3.0065	0.05113	0.2296	3.0064	0.05114	0.1178
7500	3.0014	0.04049	0.1621	3.0010	0.04046	0.1169	3.0011	0.04048	0.1643
10000	3.0205	0.03581	0.1971	3.0198	0.03578	0.1433	3.0201	0.03581	0.2113
25000	3.0035	0.02251	0.6134	3.0030	0.02249	0.4824	3.0032	0.02251	0.6539
50000	2.9923	0.01597	1.9151	2.9917	0.01596	1.3537	2.9919	0.01597	1.9797
75000	2.9971	0.01305	3.1236	2.9967	0.01304	2.2420	2.9967	0.01305	3.9595
100000	2.9837	0.01127	3.7420	2.9830	0.01126	2.6156	2.9833	0.01127	5.3264

Table 8: Pricing of geometric average rate call option with antithetic variate

The plots in 18(a), 18(b) and Figure 6 are the confidence intervals of different random walk models. Take Figure 6 as an example, initial estimates deviates from the exact value with large margin. As the number of simulations increase, the estimates quickly converges to the exact value, which is always within the bounds of the confidence intervals. This observation suggest that Monte Carlo method is indeed a reliable method for estimating this type of Asian option.

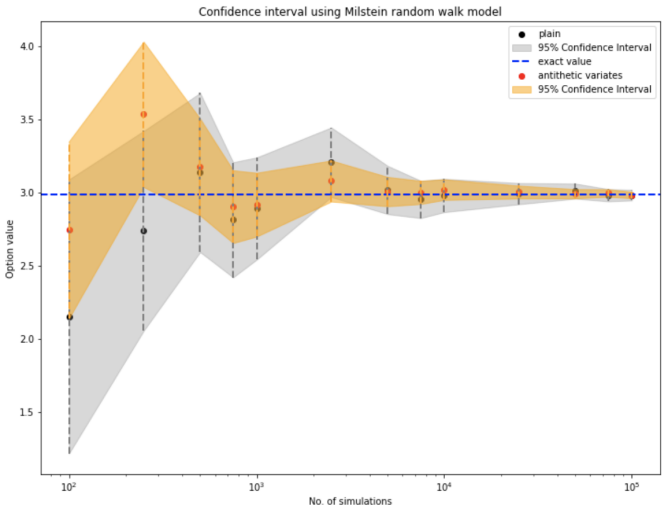


Figure 6: Confidence Interval of Milstein model for geometric average rate call option - average of full contract length

Value against Volatility sensitivity analysis

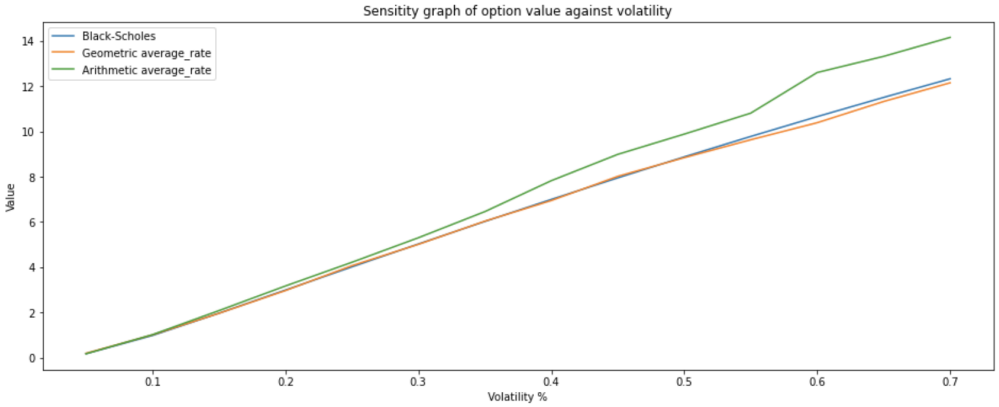


Figure 7: Option value against Volatility - average of full length of contract

Figure 7 is a graph showing the option values of geometric average rate Call option obtained from the closed-form Black-Scholes equation (in blue), Monte Carlo estimate of geometric average rate (in orange) and Monte Carlo estimate of arithmetic average rate (in green). From the graph it shows that in general as the volatility of the underlying increases so is the option value. The value between geometric average of Monte Carlo method and the Black-Scholes solution is almost identical, independent of volatility, which agrees with the result this article by Wiklund (2012)[40]. Whereas the arithmetic average of Monte Carlo method shows an overestimate of option value, with growing

differences as the volatility increases. This observation aligns with the underlying mathematics, because for arithmetic average rate Asian option, the payoff is based on the arithmetic mean of prices over the averaging period. This means that extreme values, both high and low, have a greater impact on the final calculated average, which in turn affects the option's payoff. Therefore, extreme values can lead to higher average prices and the option's potential payout is larger, which contributes to its higher cost. Whereas geometric average rate Asian option is based on the geometric mean of prices over the averaging period. Thus the geometric mean is less sensitive to extreme values and is generally lower than the arithmetic mean.

Example: geometric average rate Put option - average of full length of contract

This example shows the pricing of an Geometric average rate Put option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, E = 105$, $T_0 = 01/01/2024$, $T_1 = 01/01/2024$, $T_2 = 31/12/2024$ and $N = 365$.

Exact solution calculated by Equation 5.19 is:

Exact value
6.6983

Table 9: Exact value of geometric average rate put option

Table 10 and Table 11 are the estimated option prices by crude Monte Carlo and antithetic variates methods.

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	5.6733	0.64791	0.0027	5.6710	0.64796	0.0009	5.6731	0.64787	0.0011
250	6.9465	0.50376	0.0271	6.9492	0.50385	0.0025	6.9462	0.50373	0.0035
500	6.4207	0.33145	0.0224	6.4168	0.33143	0.0110	6.4204	0.33143	0.0057
750	6.5147	0.26809	0.0321	6.5148	0.26824	0.0073	6.5143	0.26808	0.0070
1000	6.7221	0.23106	0.0210	6.7233	0.23116	0.0089	6.7218	0.23104	0.0087
2500	6.6769	0.14695	0.0650	6.6769	0.14701	0.0217	6.6765	0.14694	0.0252
5000	6.8575	0.10632	0.0968	6.8573	0.10636	0.0449	6.8572	0.10631	0.0566
7500	6.5878	0.08612	0.1227	6.5872	0.08614	0.0583	6.5875	0.08612	0.0692
10000	6.6474	0.07378	0.1597	6.6463	0.07380	0.0747	6.6470	0.07377	0.0937
25000	6.7159	0.04724	0.8527	6.7155	0.04725	0.2997	6.7156	0.04724	0.4143
50000	6.7602	0.03358	1.2758	6.7596	0.03359	0.7112	6.7598	0.03358	0.8786
75000	6.7255	0.02715	1.4123	6.7250	0.02716	0.9016	6.7252	0.02715	1.1551
100000	6.6538	0.02346	1.7243	6.6534	0.02347	1.1644	6.6534	0.02346	1.6403

Table 10: Pricing of geometric average rate put option with crude Monte Carlo

Figure 8 is the confidence interval of using Milstein model, which shows the convergence to exact value as number of simulations increase. Confidence interval plots of Euler scheme and Euler-Maruyama models can be found in 20(a) and 20(b).

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	6.0838	0.20477	0.0521	6.0832	0.20495	0.0092	6.0835	0.20475	0.0048
250	7.0345	0.17655	0.0077	7.0364	0.17658	0.0052	7.0341	0.17654	0.0103
500	6.6462	0.11633	0.0104	6.6434	0.11648	0.0137	6.6458	0.11632	0.0108
750	6.7672	0.09497	0.0147	6.7667	0.09517	0.0108	6.7668	0.09496	0.0137
1000	6.6142	0.07983	0.0195	6.6140	0.07994	0.0141	6.6138	0.07983	0.0186
2500	6.7092	0.05398	0.0493	6.7094	0.05405	0.0384	6.7089	0.05398	0.0507
5000	6.7875	0.03806	0.1176	6.7871	0.03811	0.1197	6.7872	0.03806	0.2304
7500	6.7351	0.03079	0.1316	6.7349	0.03082	0.1117	6.7348	0.03078	0.1612
10000	6.6799	0.02630	0.2098	6.6789	0.02633	0.1605	6.6795	0.02629	0.2248
25000	6.7151	0.01686	0.6279	6.7144	0.01688	0.5142	6.7147	0.01686	0.9874
50000	6.7286	0.01201	1.9494	6.7280	0.01203	1.4980	6.7282	0.01201	2.7467
75000	6.7042	0.00973	3.2659	6.7036	0.00974	2.0900	6.7038	0.00972	4.2171
100000	6.7102	0.00844	3.7849	6.7098	0.00845	4.4285	6.7099	0.00844	4.0936

Table 11: Pricing of geometric average rate put option with antithetic variates

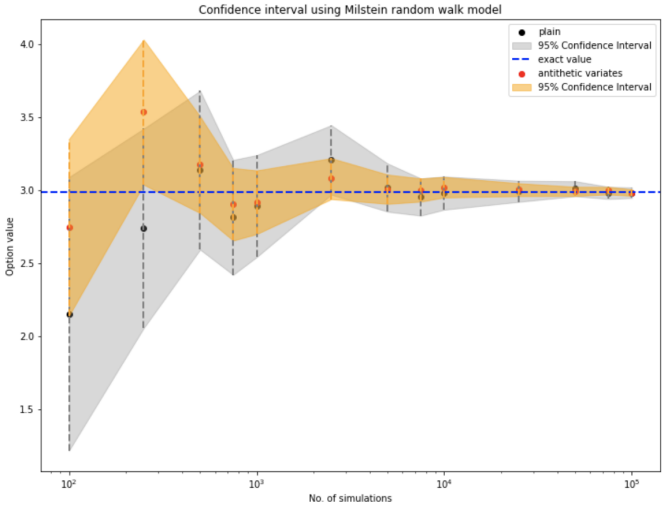


Figure 8: Confidence Interval of Milstein model for geometric average rate put option - average of full contract length

Example: geometric average rate Call option - Average of 30 days before expiry
This example shows the pricing of an Geometric average rate Put option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, E = 105$, $T_0 = 01/01/2024$, $T_1 = 01/12/2024$, $T_2 = 31/12/2024$ and $N = 365$. In this case, the average price is calculated with prices in the last 31 days of the contract.

This example presents an option price identical to the one showcased in example Table 6 when employing the exact closed-form solution as described in Equation 5.18. This is because the closed-form solution does not consider a changing averaging period other than averaging throughout the entire contract duration. Thus we can make use of Equation 3.15, the closed form solution for calculating vanilla European Call option to obtain another price for comparison.

Exact value	vanilla European Call
2.9849	7.1281

Table 12: Exact value of geometric average rate call option and vanilla European call option

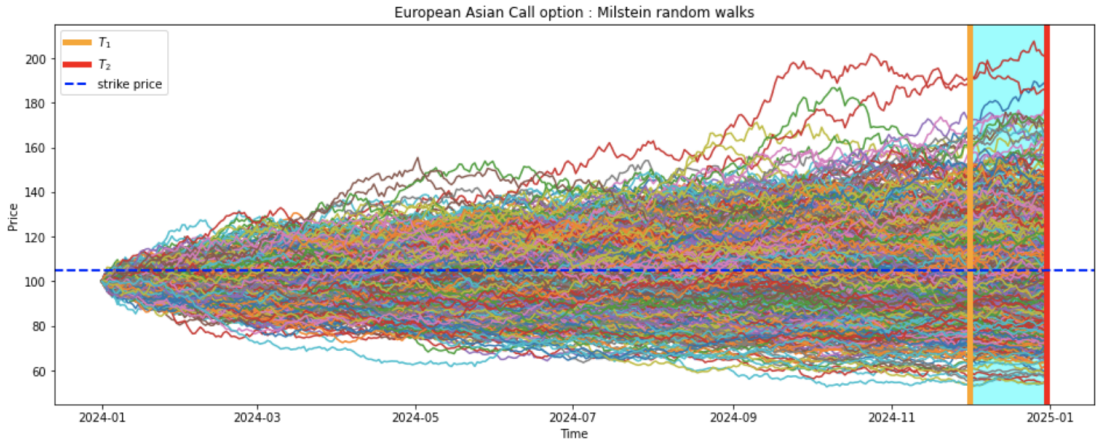


Figure 9: Monte Carlo simulation of geometric average rate Call option with average price of 30 days before expiry

Based on the findings presented in Table 14, the Monte Carlo estimation yields an approximate price of 3.43. This value is nearer to the precise figure derived from the analytical solution, 2.9849, and is notably distant from the value for a standard European Call option, 7.1281. This observation aligns with the common understanding that exotic options tend to have lower prices compared to vanilla options.

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	3.3111	0.60469	0.0055	3.3087	0.60497	0.0025	3.3106	0.60463	0.0042
250	3.4521	0.38269	0.0060	3.4488	0.38235	0.0017	3.4517	0.38266	0.0050
500	3.4757	0.30365	0.0094	3.4791	0.30392	0.0042	3.4753	0.30362	0.0055
750	3.5669	0.23297	0.0155	3.5637	0.23271	0.0086	3.5665	0.23295	0.0069
1000	3.6138	0.20926	0.0176	3.6137	0.20927	0.0082	3.6133	0.20923	0.0097
2500	3.6480	0.13795	0.0578	3.6460	0.13780	0.0236	3.6476	0.13793	0.0285
5000	3.3937	0.09225	0.1607	3.3928	0.09218	0.0550	3.3933	0.09224	0.1288
7500	3.4827	0.07682	0.1470	3.4828	0.07678	0.0641	3.4823	0.07681	0.0902
10000	3.4615	0.06576	0.2071	3.4613	0.06573	0.0973	3.4610	0.06575	0.1554
25000	3.4226	0.04154	0.6607	3.4226	0.04152	0.2606	3.4222	0.04154	0.3408
50000	3.4382	0.02944	1.6105	3.4379	0.02943	0.7043	3.4378	0.02944	0.9080
75000	3.3881	0.02384	1.6569	3.3873	0.02383	0.9250	3.3877	0.02384	1.2925
100000	3.4418	0.02083	2.2045	3.4413	0.02082	1.3594	3.4414	0.02083	1.6672

Table 13: Pricing of geometric average rate call option with crude Monte Carlo, average price of last 30 days

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	3.2191	0.35522	0.0044	3.2202	0.35568	0.0050	3.2187	0.35518	0.0044
250	3.3450	0.23496	0.0114	3.3442	0.23508	0.0047	3.3447	0.23493	0.0042
500	3.5288	0.18092	0.0097	3.5328	0.18093	0.0071	3.5284	0.18091	0.0085
750	3.3029	0.13480	0.0138	3.3038	0.13473	0.0099	3.3025	0.13479	0.0121
1000	3.6747	0.13013	0.0186	3.6757	0.13008	0.0137	3.6743	0.13011	0.0183
2500	3.3472	0.08001	0.0525	3.3467	0.07992	0.0404	3.3468	0.08001	0.0736
5000	3.3857	0.05542	0.1734	3.3851	0.05538	0.0861	3.3853	0.05542	0.1306
7500	3.4527	0.04593	0.1716	3.4525	0.04590	0.1707	3.4523	0.04593	0.2257
10000	3.4252	0.03954	0.2145	3.4248	0.03951	0.1617	3.4248	0.03953	0.3015
25000	3.4389	0.02511	0.6157	3.4384	0.02510	0.4586	3.4385	0.02511	0.7828
50000	3.4287	0.01777	1.8512	3.4280	0.01776	1.2390	3.4283	0.01777	2.0998
75000	3.4222	0.01442	3.2032	3.4214	0.01441	3.4427	3.4217	0.01442	4.9393
100000	3.4336	0.01252	3.4040	3.4332	0.01252	2.6169	3.4332	0.01252	6.5162

Table 14: Pricing of geometric average rate call option with antithetic variates, average price of last 30 days

Figure 10 illustrates the confidence interval obtained through the Monte Carlo approach for this dynamic averaging timeframe. The values tend to converge towards an estimation that lies above the exact value derived from the analytic solution (represented by the blue dashed line), yet below the value of a vanilla call option (represented by the green dashed line). Notably, none of the exact solutions fell within this interval. Consequently, when dealing with Asian options characterised by varying user-defined averaging periods, relying on the Monte Carlo method is considered a more dependable pricing strategy.

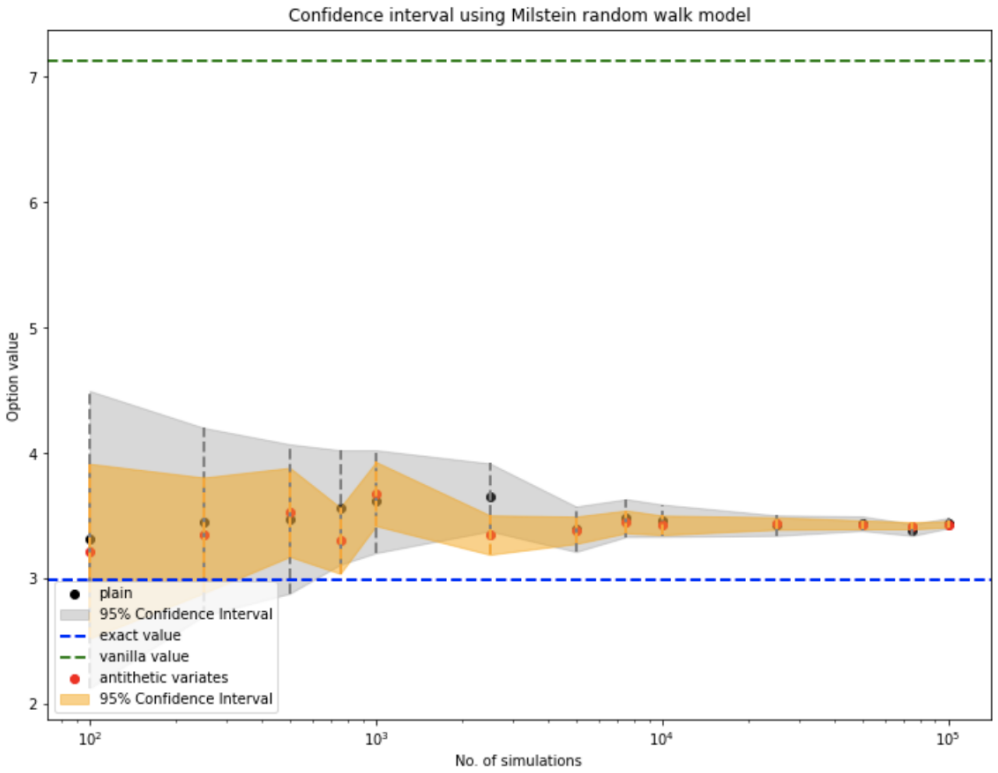


Figure 10: Confidence Interval of Milstein model for geometric average rate call option with average price of 30 days before expiry

More simulation results for estimation of Asian option with an averaging period of 60 days before expiry can be found in the Appendix under subsection A.2. In general, for geometric average rate Call and Put options, the estimated values from Monte Carlo method does converge to the exact price calculated from the analytical solutions. For arithmetic average rate Call options, the estimated price tend to be higher than estimate for geometric average rate option.

8 Legal, Social, Ethical and Professional issues

My work strictly adheres to the British Computer Society Code of Conduct, reflecting its principles of inclusivity ("You make IT for everyone"), continuous improvement ("Show what you know, learn what you don't"), ethical professionalism ("Respect the organisation or individual you work for"), and fostering growth within the IT community ("Keep IT real. Keep IT professional. Pass IT on") [41].

Legal Issues:
The development of this thesis has been undertaken with strict adherence to legal considerations, following the British Computer society (BCS) Code of Conduct. Every aspect of the research process, from data collection to analysis, has been conducted in full compliance with relevant laws and regulations. This commitment to legality ensures the integrity of the research outcomes.

Social Issues:
The social implications of this study are paramount. By focusing on the pricing of exotic options and evaluating the effectiveness of Monte Carlo simulation, this research contributes to the broader financial landscape. Its findings can potentially impact market participants, regulators, and investors, thereby fostering a more informed and efficient financial environment.

Ethical Issues:
Ethical considerations have been a cornerstone of this thesis. The thorough review of professional standards and the unwavering commitment to conducting rigorous and transparent analysis exemplify ethical conduct. The research process has been driven by a sense of responsibility, avoiding discrimination and respecting the rights of all stakeholders.

Professional Issues:
This thesis has been shaped by a deep commitment to professional excellence. From the meticulous assessment of the project's scope to continuous learning and development, the pursuit of professional competence has been at the forefront. By embracing the antithetic variates technique for variance reduction and applying the principles of the British Computer Society Code of Conduct, this research sets a standard for future finance professionals. The collaborative environment fostered throughout the research journey embodies the spirit of professional development and mutual support within the field.

9 Conclusion

In conclusion, the findings of this thesis underscore the effectiveness of the Monte Carlo method in accurately pricing various types of Barrier and Asian options. The Monte Carlo approach demonstrates its robustness by consistently converging to the exact values derived from analytical solutions as the number of simulations increases, typically stabilizing at around 5000 simulations for Barrier options. A similar trend is observed with Asian options, which achieve convergence with a smaller number of simulations, typically ranging from 750 to 1000. This intriguing pattern suggests that the Monte Carlo method exhibits superior efficacy in valuing Asian options compared to Barrier options.

In addition to the aforementioned findings, a significant insight arises from the examination of different random walk models, specifically the Euler Scheme, Euler-Maruyama, and Milstein's method, which collectively have shown minimal impact on enhancing the accuracy of the Monte Carlo simulation. The convergence rates for the estimated prices using these models are remarkably consistent, implying that the choice of random walk model under the Black-Scholes framework may not be the most influential factor in refining the precision of the Monte Carlo method.

Furthermore, it is crucial to emphasise the significant impact of the antithetic variate technique in reducing the variance of the Monte Carlo simulations, resulting in notable improvements in convergence speed. Specifically, the application of this method demonstrated its effectiveness in accelerating the convergence process by approximately 1.5 times when estimating Barrier options, and by a factor of about 1.3 when pricing Asian options. This results agrees with the findings presented in the paper by C. N. De Ponte (2013)[42] and the paper by B. Wang and L. Wang (2011)[43], where the use of antithetic variate reduces the bounds of the confidence interval significantly compared with standard Monte Carlo. At the same time, the computational time for using antithetic variates is approximately 1.5 times longer than that of the standard Monte Carlo method due to the need for calculations involving twice as large a sample size. These findings highlights the practical importance of the antithetic variate technique in enhancing the overall efficiency and precision of option pricing within the Monte Carlo framework.

A valuable takeaway for future research is the suggestion that professionals and researchers might benefit more from focusing on exploring and implementing variance reduction techniques rather than investing extensive effort in developing new mathematical models of random walks solely for basic Monte Carlo testing within the Black-Scholes framework. While the groundwork presented in this thesis relies on the Black-Scholes model, it is crucial to acknowledge its underlying assumption that volatility and risk-free interest rates are constant and known. To align Monte Carlo simulation more effectively with real-world trading scenarios, it is worthwhile to explore the development of new random walk models that do not rely on fixed constants but instead incorporate contin-

uously evolving volatility and risk-free interest rates of the underlying which the option is conditioned on.

References

[1] J. Fernando, "Derivatives: Types, considerations, and pros and cons, investopedia." https://www.investopedia.com/terms/d/derivative.asp, January 2023. Accessed: April 16, 2023.

[2] F. Weert, *Exotic Options Trading*. Chichester, England: John Wiley & Sons, 2008.

[3] P. Wilmott, *Paul Wilmott introduces quantitative finance*. John Wiley & Sons, 2007.

[4] Y.-K. Kwok, *Mathematical models of Financial Derivatives*. Springer Berlin, Heidelberg, 2008.

[5] J. Hull, *Options futures and other derivatives*. Pearson Education India, 2003.

[6] Z. Janková, "Drawbacks and limitations of black-scholes model for options pricing.," *Journal of Financial Studies & Research*, pp. 1–7, 2018.

[7] P. Zhang, *An introduction to exotic options*. European Financial Management, 1995.

[8] V. Henderson, "Black–scholes model.," *Wiley StatsRef: Statistics Reference Online.*, 2014.

[9] F. Black and M. Scholes, "Ethe pricing of options and corporate liabilities.," *Journal of political economy*, vol. 81, no. 3, pp. 637–654, 1973.

[10] P. Wilmott, S. Howison, and J. Dewynne, *The mathematics of financial derivatives: a student introduction*. Cambridge university press, 1995.

[11] P. Carr, "Two extensions to barrier option valuation," *Applied Mathematical Finance*, pp. 173–209, 1995.

[12] R. Merton, "Theory of rational option pricing," *The Bell Journal of economics and management science*, pp. 141–183, 1973.

[13] D. Rich, "The mathematical foundations of barrier option-pricing theory," *Advances in Futures and Options Research*, pp. 267– 311, 1991.

[14] E. Reiner and M. Rubinstein, "Breaking down the barriers.," *Risk*, vol. 4, no. 8, pp. 28–35, 1991.

[15] H. K. R. Heynen, "Partial barrier options," *Journal of Financial Engineering*, pp. 253–274, 1994.

[16] E. Banks, *The credit risk of complex derivatives*. Springer, 2016.

[17] F. AitSahlia, L. Imhof, and T. Lai, "Fast and accurate valuation of american barrier op- tions," *Journal of Computational Finance*, pp. 129–145, 2003.

[18] R. Hu, N. Huang, C. Zhao, and B. Lee, "The pricing for a class of barrier options.," *NONLINEAR ANALYSIS FORUM*, vol. 11, no. 1, p. 33, 2006.

[19] S. G. Kou, "On pricing of discrete barrier options.," *Statistica Sinica*, pp. 955–964, 2003.

[20] P. Wilmott, *Paul Wilmott on quantitative finance*. John Wiley & Sons, 2013.

[21] N. Kunitomo and M. Ikeda, "Pricing options with curved boundaries," *Mathematical finance*, pp. 275–298, 1992.

[22] H. Kat and L. Verdonk, "Tree surgery.," *Cass Business School Research Paper.*, 1995.

[23] M. Broadie, P. Glasserman, and S. Kou, "A continuity correction for discrete barrier options," *Mathematical Finance*, pp. 325–349, 1997.

[24] V. Cardellini, A. Fanfarillo, and S. Filippone, "Heterogeneous caf-based load balancing on intel xeon phi," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 702–711, 2016.

[25] E. W., "Asian option pricing and volatility.," 2012.

[26] A. Kemna and A. Vorst, "A pricing method for options based on average asset values," *Journal of Banking & Finance*, pp. 113–129, 1990.

[27] G. Fusai and A. Meucci, "Pricing discretely monitored asian options under lévy processes.," *Journal of Banking & Finance*, vol. 32, no. 10, pp. 2076–2088, 2008.

[28] J. F. EP., *An Introduction to Numerical Methods and Analysis*. Wiley, 2013.

[29] W. Kenton, "Monte carlo simulation: History, how it works, and 4 key steps, investopedia." https://www.investopedia.com/terms/m/montecarlosimulation.asp, March 2023. Accessed: April 17, 2023.

[30] E. Atanassov and I. Dimov, "What monte carlo models can do and cannot do efficiently?," *Applied Mathematical Modelling*, pp. 1477–1500, 2008.

[31] P. P. Boyle, "Options: A monte carlo approach," *Journal of financial economics*, vol. 4, no. 3, pp. 323–338, 1977.

[32] P. Boyle, M. Broadie, and P. Glasserman, "Monte carlo methods for security pricing," *Journal of economic dynamics and control*, vol. 21, no. 8-9, pp. 1267–1321, 1997.

[33] P. Glasserman, *Monte Carlo methods in financial engineering*. New York: springer, 2004.

[34] S. Asmussen and P. W. Glynn, *Stochastic simulation: algorithms and analysis*, vol. 57. Springer, 2007.

[35] P. E. Kloeden, E. Platen, P. E. Kloeden, and E. Platen, *Stochastic differential equations*. Springer, 1992.

[36] G. N. Milstein and M. V. Tretyakov, *Stochastic numerics for mathematical physics*, vol. 39. Springer, 2004.

[37] D. J. HIGHAM, *An Introduction to Financial Option Valuation*. Cambridge University Press, 2004.

[38] G. Gracianti, "Computing greeks by finite difference using monte carlo simulation and variance reduction techniques.," *BIMIPA*, vol. 25, no. 1, pp. 80–93, 2018.

[39] F. Rouah, "Euler and milstein discretization.," *Documento de trabajo, Sapient Global Markets, Estados Unidos.*, 2011.

[40] E. Wiklund, "Asian option pricing and volatility," 2012.

[41] T. C. I. f. I. BCS, "Code of conduct for bcs members." https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/, June 2022. Accessed: August 1, 2023.

[42] C. N. De Ponte, *Pricing barrier options with numerical methods*. PhD thesis, North-West University, 2013.

[43] B. Wang and L. Wang, "Pricing barrier options using monte carlo methods," 2011.

A Appendix

A.1 Barrier Option simulation results

A.1.1 up-and-out call option

Consider pricing an up-and-out call option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, S_u = 110, S_u^* = 110.6772, E = 105$, $t_0 = 01/01/2024$, $T = 31/12/2024$, $N = 365$.

Exact value
0.0225

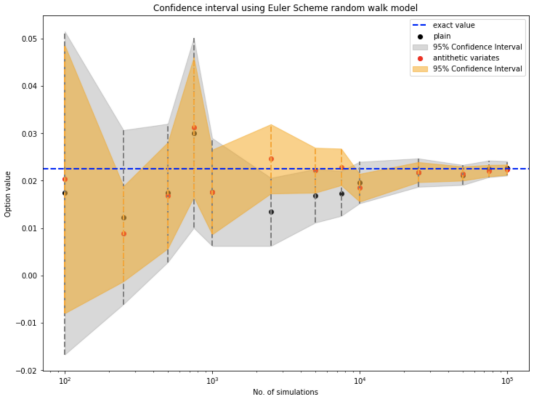
Table 15: Exact value of up-and-out Call Barrier option

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	0.0174	0.01740	0.0106	0.0178	0.01779	0.0017	0.0174	0.01740	0.0116
250	0.0123	0.00938	0.0038	0.0133	0.00988	0.0010	0.0123	0.00937	0.0020
500	0.0174	0.00745	0.0079	0.0181	0.00797	0.0032	0.0174	0.00745	0.0030
750	0.0301	0.01024	0.0109	0.0314	0.01061	0.0034	0.0301	0.01024	0.0043
1000	0.0176	0.00581	0.0167	0.0193	0.00611	0.0050	0.0176	0.00581	0.0062
2500	0.0134	0.00367	0.0312	0.0147	0.00399	0.0130	0.0134	0.00367	0.0165
5000	0.0168	0.00289	0.0743	0.0165	0.00289	0.0296	0.0168	0.00289	0.0383
7500	0.0173	0.00242	0.2883	0.0179	0.00250	0.0542	0.0173	0.00242	0.0548
10000	0.0196	0.00224	0.1806	0.0209	0.00235	0.0543	0.0196	0.00224	0.0692
25000	0.0217	0.00153	0.4662	0.0214	0.00152	0.1473	0.0217	0.00153	0.1990
50000	0.0212	0.00108	0.8759	0.0206	0.00105	0.4195	0.0212	0.00108	0.7145
75000	0.0225	0.00090	1.0722	0.0224	0.00090	0.6138	0.0225	0.00090	0.8472
100000	0.0226	0.00078	1.4123	0.0226	0.00077	0.8016	0.0226	0.00078	1.0614

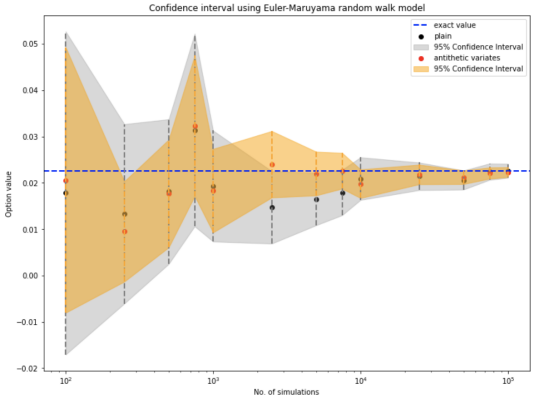
Table 16: Pricing of up-and-out Call option with crude Monte Carlo

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	0.0203	0.01443	0.0043	0.0206	0.01461	0.0044	0.0203	0.01442	0.0023
250	0.0088	0.00513	0.0041	0.0095	0.00552	0.0025	0.0087	0.00512	0.0032
500	0.0169	0.00572	0.0071	0.0177	0.00594	0.0045	0.0169	0.00572	0.0077
750	0.0313	0.00753	0.0093	0.0323	0.00774	0.0060	0.0313	0.00753	0.0089
1000	0.0176	0.00456	0.0129	0.0183	0.00460	0.0100	0.0176	0.00456	0.0120
2500	0.0246	0.00373	0.0355	0.0240	0.00366	0.0236	0.0246	0.00373	0.0341
5000	0.0222	0.00242	0.0760	0.0220	0.00241	0.0491	0.0222	0.00242	0.0709
7500	0.0229	0.00197	0.1129	0.0226	0.00197	0.0822	0.0229	0.00197	0.1421
10000	0.0185	0.00150	0.1321	0.0198	0.00158	0.0956	0.0185	0.00150	0.1623
25000	0.0218	0.00108	0.3724	0.0218	0.00108	0.2615	0.0218	0.00108	0.4495
50000	0.0215	0.00076	1.3048	0.0212	0.00075	0.8544	0.0215	0.00076	1.3182
75000	0.0221	0.00062	1.9646	0.0221	0.00062	1.3362	0.0221	0.00062	4.0066
100000	0.0223	0.00055	2.2721	0.0223	0.00054	1.4929	0.0223	0.00055	4.8946

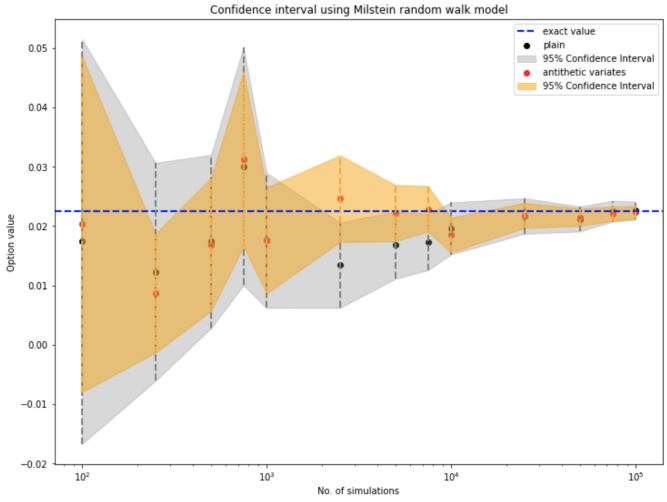
Table 17: Pricing of up-and-out Call option with antithetic variates



(a) Euler Scheme model



(b) Euler-Maruyama model



(c) Milstein model

Figure 11: Confidence Interval for up-and-out Call Barrier option

A.1.2 down-and-in call option

Consider pricing an down-and-in call option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, S_d = 90, S_d^* = 89.4493, E = 105,\ t_0 = 01/01/2024,\ T = 31/12/2024,\ N = 365$.

Exact value
0.9376

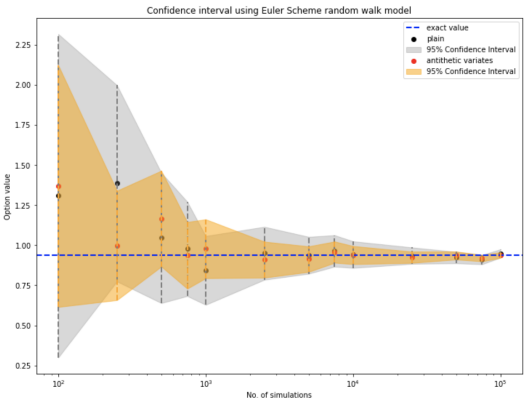
Table 18: Exact value of down-and-in Call Barrier option

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	1.3081	0.51458	0.0087	1.3091	0.51464	0.0007	1.3080	0.51452	0.0022
250	1.3858	0.31220	0.0066	1.3847	0.31224	0.0009	1.3856	0.31217	0.0025
500	1.0452	0.20735	0.0089	1.0432	0.20628	0.0027	1.0451	0.20733	0.0063
750	0.9769	0.15007	0.0124	0.9458	0.14530	0.0039	0.9768	0.15005	0.0043
1000	0.8425	0.10947	0.0153	0.8371	0.10924	0.0049	0.8424	0.10946	0.0057
2500	0.9495	0.08425	0.0396	0.9484	0.08451	0.0137	0.9494	0.08425	0.0169
5000	0.9371	0.05865	0.0896	0.9432	0.05867	0.0277	0.9356	0.05863	0.0379
7500	0.9642	0.05001	0.2382	0.9626	0.04990	0.0519	0.9623	0.04997	0.0684
10000	0.9411	0.04243	0.1906	0.9397	0.04204	0.0709	0.9388	0.04237	0.1143
25000	0.9349	0.02611	0.4689	0.9316	0.02604	0.1986	0.9348	0.02611	0.3333
50000	0.9229	0.01824	0.9563	0.9226	0.01825	0.4355	0.9228	0.01824	0.7446
75000	0.9100	0.01489	1.4657	0.9075	0.01485	0.7340	0.9099	0.01489	0.8705
100000	0.9485	0.01320	1.6802	0.9504	0.01324	1.0968	0.9481	0.01319	1.4685

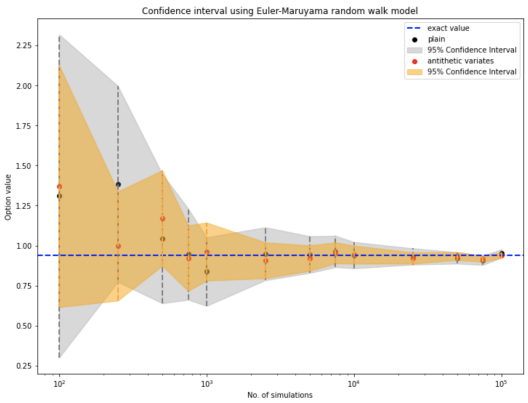
Table 19: Pricing of down-and-in Call option with crude Monte Carlo

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	1.3684	0.38446	0.0016	1.3697	0.38506	0.0010	1.3682	0.38442	0.0015
250	0.9985	0.17400	0.0070	0.9964	0.17388	0.0087	0.9984	0.17399	0.0055
500	1.1655	0.15243	0.0069	1.1713	0.15211	0.0044	1.1654	0.15241	0.0061
750	0.9366	0.10606	0.0098	0.9210	0.10449	0.0076	0.9365	0.10605	0.0096
1000	0.9777	0.09372	0.0130	0.9620	0.09249	0.0080	0.9776	0.09371	0.0121
2500	0.9097	0.05729	0.0382	0.9074	0.05733	0.0268	0.9096	0.05729	0.0413
5000	0.9130	0.04065	0.0733	0.9209	0.04077	0.0490	0.9122	0.04064	0.0821
7500	0.9568	0.03381	0.1402	0.9539	0.03370	0.0883	0.9558	0.03380	0.1408
10000	0.9381	0.02882	0.1922	0.9430	0.02900	0.1276	0.9369	0.02880	0.1920
25000	0.9248	0.01800	0.4955	0.9218	0.01796	0.3565	0.9246	0.01800	0.5171
50000	0.9357	0.01271	1.5345	0.9337	0.01271	1.2968	0.9352	0.01271	2.2198
75000	0.9193	0.01033	2.3078	0.9174	0.01031	1.5413	0.9192	0.01033	3.8506
100000	0.9385	0.00903	3.1570	0.9394	0.00905	1.7659	0.9382	0.00903	3.9074

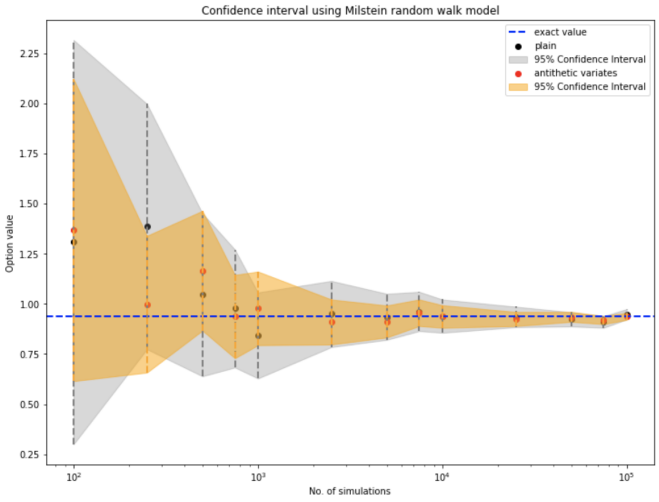
Table 20: Pricing of down-and-in Call option with antithetic variates



(a) Euler Scheme model



(b) Euler-Maruyama model



(c) Milstein model

Figure 12: Confidence Interval for down-and-in Call Barrier option

A.1.3 down-and-out call option

Consider pricing an down-and-out call option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, S_d = 90, S_d^* = 89.4493, E = 105$, $t_0 = 01/01/2024$, $T = 31/12/2024$, $N = 365$.

Exact value
6.1905

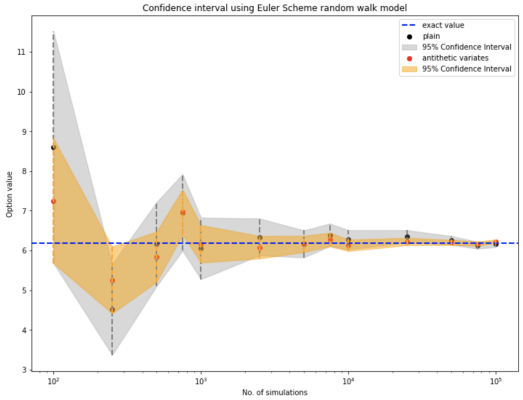
Table 21: Exact value of down-and-out Call Barrier option

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	8.5930	1.49015	0.0033	8.5700	1.48542	0.0016	8.5921	1.48998	0.0022
250	4.5115	0.58470	0.0037	4.5198	0.58518	0.0013	4.5110	0.58465	0.0017
500	6.1562	0.54002	0.0234	6.1425	0.53979	0.0032	6.1557	0.53997	0.0063
750	6.9554	0.49232	0.0150	6.9513	0.49225	0.0046	6.9548	0.49227	0.0042
1000	6.0490	0.39550	0.0136	6.0479	0.39539	0.0048	6.0484	0.39546	0.0056
2500	6.3381	0.23746	0.0514	6.3255	0.23725	0.0197	6.3374	0.23743	0.0247
5000	6.1615	0.17430	0.1822	6.1578	0.17419	0.0409	6.1608	0.17429	0.0539
7500	6.3912	0.14506	0.1217	6.3865	0.14501	0.0448	6.3905	0.14505	0.0622
10000	6.2665	0.12379	0.1731	6.2717	0.12376	0.0666	6.2659	0.12377	0.0848
25000	6.3535	0.07925	0.4166	6.3517	0.07918	0.1680	6.3528	0.07924	0.2523
50000	6.2507	0.05574	0.7496	6.2527	0.05572	0.4569	6.2501	0.05573	0.7012
75000	6.1291	0.04477	1.3554	6.1318	0.04475	0.7472	6.1285	0.04476	1.0520
100000	6.1601	0.03886	1.8240	6.1584	0.03884	0.9693	6.1595	0.03885	1.4918

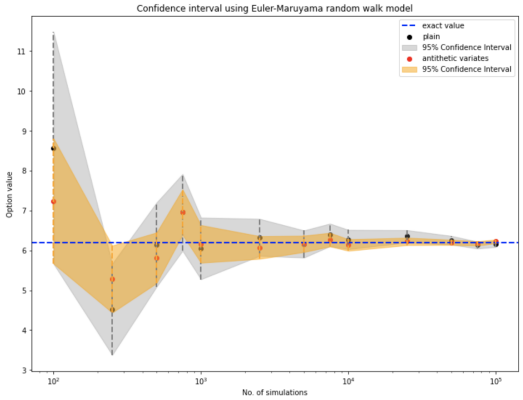
Table 22: Pricing of down-and-out Call option with crude Monte Carlo

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	7.2457	0.80378	0.0026	7.2360	0.80217	0.0011	7.2450	0.80369	0.0020
250	5.2536	0.42985	0.0035	5.2783	0.42946	0.0032	5.2531	0.42981	0.0033
500	5.8372	0.32425	0.0081	5.8118	0.32466	0.0057	5.8367	0.32422	0.0066
750	6.9615	0.29197	0.0116	6.9592	0.29189	0.0058	6.9609	0.29194	0.0088
1000	6.1634	0.24133	0.0129	6.1602	0.24110	0.0079	6.1628	0.24130	0.0127
2500	6.0789	0.14542	0.0579	6.0712	0.14540	0.0258	6.0783	0.14541	0.0401
5000	6.1614	0.10502	0.1131	6.1619	0.10496	0.2398	6.1608	0.10501	0.1718
7500	6.2725	0.08778	0.1284	6.2684	0.08775	0.0875	6.2719	0.08777	0.1343
10000	6.1278	0.07399	0.1853	6.1333	0.07394	0.1252	6.1272	0.07398	0.1859
25000	6.2226	0.04777	0.4606	6.2231	0.04774	0.3385	6.2228	0.04777	0.5752
50000	6.1998	0.03372	1.5296	6.2013	0.03370	0.9271	6.1992	0.03371	1.7245
75000	6.1696	0.02744	2.4780	6.1712	0.02742	1.4590	6.1690	0.02744	2.0575
100000	6.2262	0.02387	3.3218	6.2262	0.02386	2.0122	6.2256	0.02387	5.2886

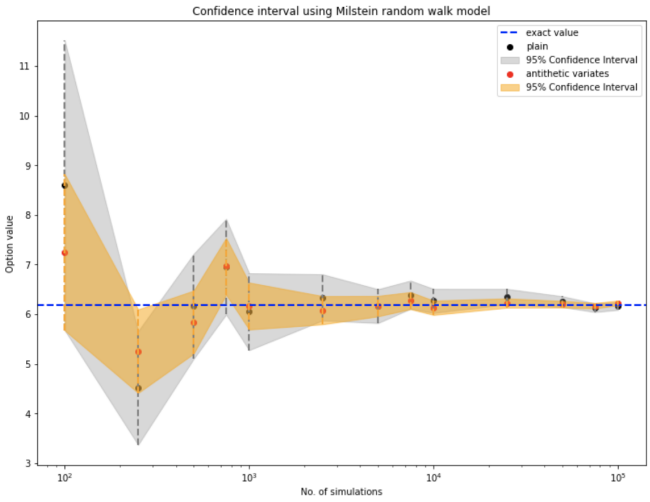
Table 23: Pricing of down-and-out Call option with antithetic variates



(a) Euler Scheme model



(b) Euler-Maruyama model



(c) Milstein model

Figure 13: Confidence Interval for down-and-out Call Barrier option

A.1.4 up-and-in put option

Consider pricing an up-and-in Put option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, S_u = 110, S_u^* = 110.6772, E = 105$, $t_0 = 01/01/2024$, $T = 31/12/2024$, $N = 365$.

Exact value
2.2663

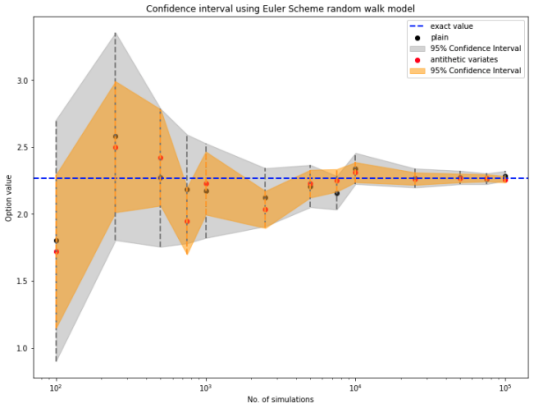
Table 24: Exact value of up-and-in Put Barrier option

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	1.8017	0.46013	0.0022	1.8005	0.46059	0.0008	1.8016	0.46009	0.0014
250	2.5793	0.39543	0.0037	2.5846	0.39636	0.0009	2.5791	0.39540	0.0016
500	2.2702	0.26297	0.0152	2.2204	0.26169	0.0058	2.2701	0.26295	0.0059
750	2.1868	0.20737	0.0134	2.1893	0.20745	0.0055	2.1867	0.20736	0.0051
1000	2.1745	0.18032	0.0160	2.1904	0.18064	0.0058	2.1743	0.18031	0.0070
2500	2.1247	0.11078	0.0400	2.1154	0.11076	0.0155	2.1245	0.11077	0.0207
5000	2.2084	0.08048	0.0991	2.2195	0.08061	0.0424	2.2083	0.08047	0.0591
7500	2.1580	0.06386	0.1185	2.1666	0.06403	0.0440	2.1578	0.06386	0.0549
10000	2.3396	0.05934	0.1611	2.3537	0.05952	0.0681	2.3394	0.05934	0.1239
25000	2.2686	0.03608	0.4747	2.2688	0.03610	0.1649	2.2684	0.03608	0.2288
50000	2.2725	0.02573	0.9990	2.2759	0.02576	0.4922	2.2718	0.02573	0.7857
75000	2.2629	0.02090	1.4553	2.2657	0.02091	1.0388	2.2627	0.02090	1.2158
100000	2.2851	0.01821	1.4827	2.2854	0.01822	0.7406	2.2849	0.01821	1.1131

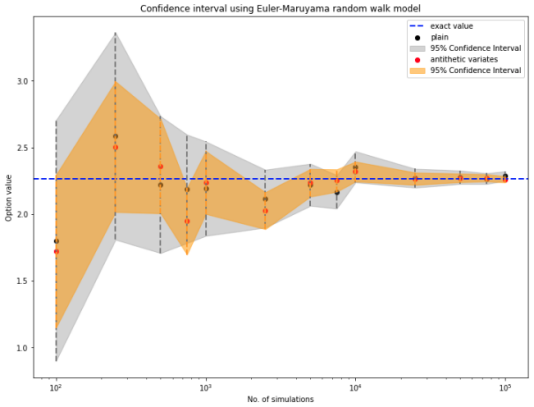
Table 25: Pricing of up-and-in Put option with crude Monte Carlo

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	1.7200	0.29238	0.0037	1.7202	0.29253	0.0020	1.7199	0.29236	0.0031
250	2.5014	0.24960	0.0052	2.5047	0.24988	0.0025	2.5012	0.24958	0.0037
500	2.4216	0.18360	0.0080	2.3598	0.18089	0.0109	2.3851	0.18092	0.0078
750	1.9483	0.12756	0.0115	1.9465	0.12749	0.0074	1.9482	0.12755	0.0116
1000	2.2306	0.12000	0.0148	2.2368	0.12021	0.0101	2.2304	0.11999	0.0148
2500	2.0333	0.07074	0.0410	2.0252	0.07074	0.0290	2.0331	0.07074	0.0480
5000	2.2262	0.05285	0.0897	2.2347	0.05292	0.0623	2.2261	0.05284	0.3855
7500	2.2505	0.04285	0.1183	2.2513	0.04286	0.0921	2.2503	0.04284	0.3390
10000	2.3115	0.03814	0.2239	2.3184	0.03818	0.1548	2.3114	0.03814	0.2246
25000	2.2635	0.02373	0.4123	2.2650	0.02374	0.2891	2.2633	0.02373	0.4415
50000	2.2686	0.01678	2.0145	2.2714	0.01679	0.9884	2.2682	0.01677	1.5960
75000	2.2698	0.01374	2.6176	2.2718	0.01375	1.4712	2.2697	0.01374	3.7293
100000	2.2586	0.01185	3.3587	2.2599	0.01186	5.9694	2.2584	0.01185	5.3110

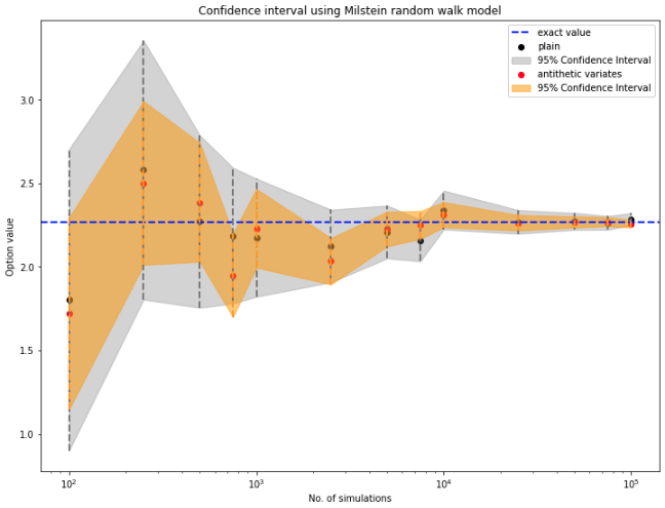
Table 26: Pricing of up-and-in Put option with antithetic variates



(a) Euler Scheme model



(b) Euler-Maruyama model



(c) Milstein model

Figure 14: Confidence Interval for up-and-in Put Barrier option

A.1.5 up-and-out put option

Consider pricing an up-and-out Put option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, S_u = 110, S_u^* = 110.6772, E = 105$, $t_0 = 01/01/2024$, $T = 31/12/2024$, $N = 365$.

Exact value
6.7585

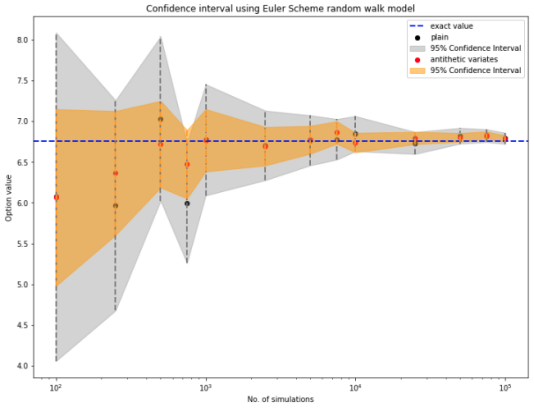
Table 27: Exact value of up-and-out Put Barrier option

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	6.0723	1.02866	0.0032	6.0843	1.02963	0.0009	6.0720	1.02860	0.0013
250	5.9660	0.65823	0.0056	5.9650	0.65866	0.0015	5.9656	0.65819	0.0020
500	7.0308	0.51460	0.0250	6.9951	0.51431	0.0049	7.0303	0.51457	0.0051
750	5.9941	0.37625	0.0256	5.9891	0.37614	0.0058	5.9937	0.37622	0.0057
1000	6.7702	0.34797	0.0183	6.7870	0.34807	0.0059	6.7698	0.34794	0.0077
2500	6.7020	0.21780	0.0500	6.6987	0.21780	0.0242	6.7015	0.21779	0.0274
5000	6.7649	0.15660	0.0992	6.7593	0.15663	0.0354	6.7678	0.15660	0.0419
7500	6.7771	0.12702	0.0993	6.7706	0.12703	0.0365	6.7766	0.12701	0.0487
10000	6.8461	0.11016	0.1674	6.8332	0.11009	0.0629	6.8456	0.11015	0.1176
25000	6.7318	0.06908	0.4699	6.7319	0.06908	0.2284	6.7314	0.06907	0.2605
50000	6.8203	0.04937	1.2906	6.8197	0.04937	0.5166	6.8199	0.04936	0.7084
75000	6.8215	0.04030	1.1801	6.8191	0.04030	0.6211	6.8211	0.04030	1.0139
100000	6.7888	0.03480	1.8204	6.7841	0.03480	1.0087	6.7884	0.03480	1.6176

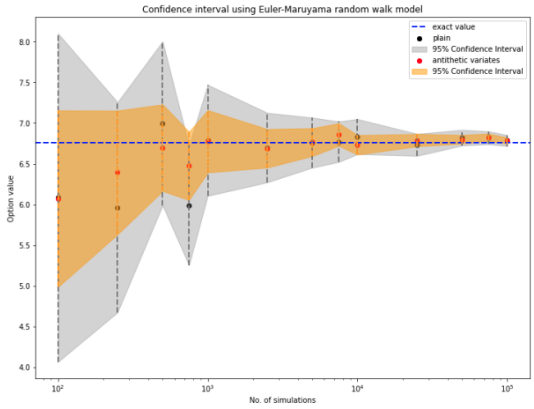
Table 28: Pricing of up-and-out Put option with crude Monte Carlo

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	6.0626	0.55303	0.0032	6.0705	0.55309	0.0027	6.0621	0.55299	0.0021
250	6.3636	0.38794	0.0057	6.3924	0.38753	0.0023	6.3632	0.38791	0.0042
500	6.7168	0.27101	0.0085	6.6972	0.27131	0.0054	6.7164	0.27099	0.0075
750	6.4716	0.21397	0.0119	6.4738	0.21390	0.0077	6.4712	0.21395	0.0117
1000	6.7647	0.19534	0.0165	6.7752	0.19528	0.0112	6.7642	0.19533	0.0176
2500	6.6924	0.12038	0.0457	6.6893	0.12051	0.0393	6.6919	0.12037	0.0588
5000	6.7711	0.08710	0.1103	6.7655	0.08714	0.2820	6.7722	0.08709	0.1728
7500	6.8603	0.07062	0.0982	6.8584	0.07065	0.0708	6.8599	0.07061	0.2165
10000	6.7370	0.06125	0.2169	6.7316	0.06125	0.1183	6.7365	0.06124	0.1809
25000	6.7930	0.03860	0.3865	6.7896	0.03861	0.2697	6.7925	0.03859	0.4272
50000	6.8012	0.02737	1.6332	6.8000	0.02738	0.9059	6.8006	0.02737	2.1895
75000	6.8290	0.02234	2.7872	6.8261	0.02235	3.0167	6.8286	0.02234	2.9938
100000	6.7862	0.01935	4.0862	6.7829	0.01935	4.8213	6.7857	0.01934	6.3862

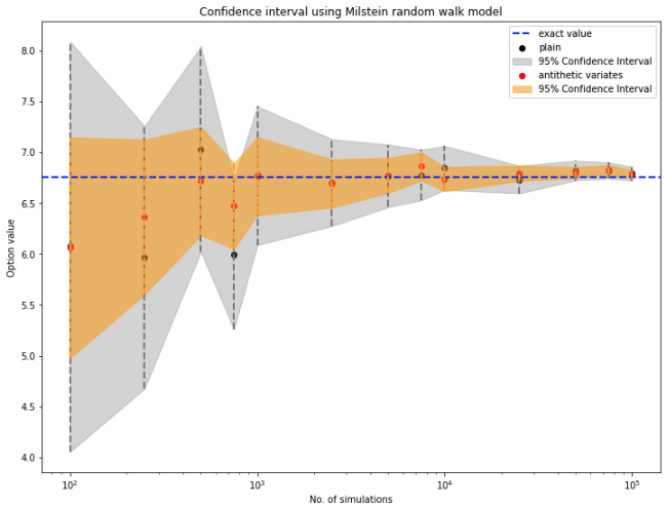
Table 29: Pricing of up-and-out Put option with antithetic variates



(a) Euler Scheme model



(b) Euler-Maruyama model



(c) Milstein model

Figure 15: Confidence Interval for up-and-out Put Barrier option

A.1.6 down-and-in put option

Consider pricing an down-and-in Put option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, S_d = 90, S_d^* = 89.4493, E = 105$, $t_0 = 01/01/2024$, $T = 31/12/2024$, $N = 365$.

Exact value
8.4428

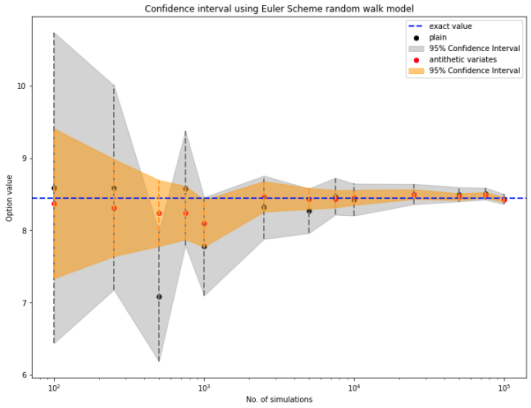
Table 30: Exact value of down-and-in Put Barrier option

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	8.5873	1.09697	0.0025	8.6048	1.09728	0.0008	8.5868	1.09691	0.0038
250	8.5927	0.72241	0.0079	8.5913	0.72247	0.0024	8.5921	0.72236	0.0040
500	7.0858	0.45952	0.0080	7.0820	0.45936	0.0032	7.0853	0.45949	0.0038
750	8.5818	0.40545	0.0118	8.5677	0.40580	0.0043	8.5813	0.40542	0.0059
1000	7.7768	0.34722	0.0228	7.7804	0.34737	0.0068	7.7762	0.34720	0.0078
2500	8.3169	0.22364	0.0382	8.3081	0.22379	0.0154	8.3163	0.22363	0.0211
5000	8.2694	0.15764	0.1043	8.2581	0.15770	0.0384	8.2689	0.15763	0.0413
7500	8.4700	0.13035	0.1379	8.4691	0.13039	0.0487	8.4694	0.13034	0.0659
10000	8.4240	0.11285	0.1590	8.4279	0.11287	0.0832	8.4234	0.11285	0.1115
25000	8.5007	0.07182	0.4001	8.5042	0.07184	0.1665	8.5001	0.07182	0.2201
50000	8.4977	0.05064	1.1614	8.4981	0.05065	0.6583	8.4970	0.05064	0.7079
75000	8.5068	0.04163	1.1240	8.5052	0.04164	0.6171	8.5062	0.04163	1.1500
100000	8.4315	0.03571	1.7302	8.4304	0.03572	0.9249	8.4308	0.03571	1.1247

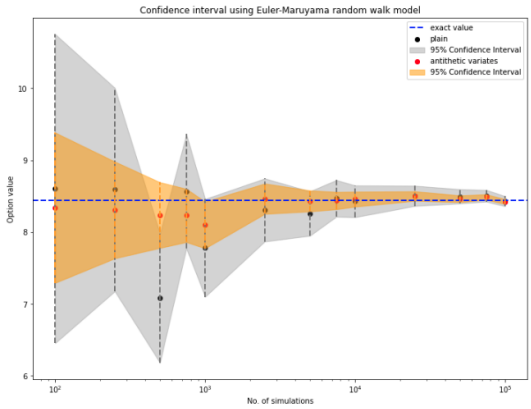
Table 31: Pricing of down-and-in Put option with crude Monte Carlo

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	8.3720	0.52886	0.0032	8.3394	0.53352	0.0015	8.3715	0.52882	0.0019
250	8.3133	0.34312	0.0062	8.3079	0.34337	0.0034	8.3127	0.34310	0.0079
500	8.2375	0.23300	0.0135	8.2369	0.23303	0.0086	8.2369	0.23298	0.0102
750	8.2423	0.18866	0.0107	8.2314	0.18915	0.0074	8.2418	0.18864	0.0108
1000	8.0971	0.16574	0.0156	8.0991	0.16582	0.0106	8.0966	0.16573	0.0176
2500	8.4674	0.10760	0.0442	8.4647	0.10773	0.0336	8.4668	0.10759	0.0509
5000	8.4394	0.07450	0.0819	8.4339	0.07462	0.0582	8.4388	0.07450	0.1300
7500	8.4357	0.06153	0.1493	8.4386	0.06153	0.0865	8.4351	0.06153	0.1299
10000	8.4547	0.05327	0.2016	8.4580	0.05328	0.1170	8.4542	0.05327	0.2469
25000	8.4996	0.03369	0.4196	8.5016	0.03370	0.2925	8.4991	0.03369	0.4421
50000	8.4641	0.02383	1.5774	8.4636	0.02384	0.8585	8.4634	0.02383	1.6730
75000	8.4882	0.01956	2.4268	8.4879	0.01958	1.4817	8.4876	0.01956	4.1676
100000	8.4259	0.01680	2.9631	8.4252	0.01681	1.8027	8.4252	0.01680	7.5124

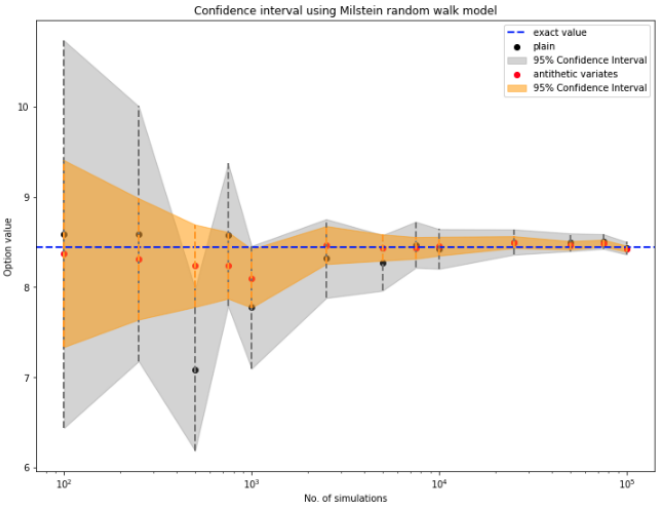
Table 32: Pricing of down-and-in Put option with antithetic variates



(a) Euler Scheme model



(b) Euler-Maruyama model



(c) Milstein model

Figure 16: Confidence Interval for down-and-in Put Barrier option

A.1.7 down-and-out put option

Consider pricing an down-and-out Put option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, S_d = 90, S_d^* = 89.4493, E = 105$, $t_0 = 01/01/2024$, $T = 31/12/2024$, $N = 365$.

Exact value
0.5821

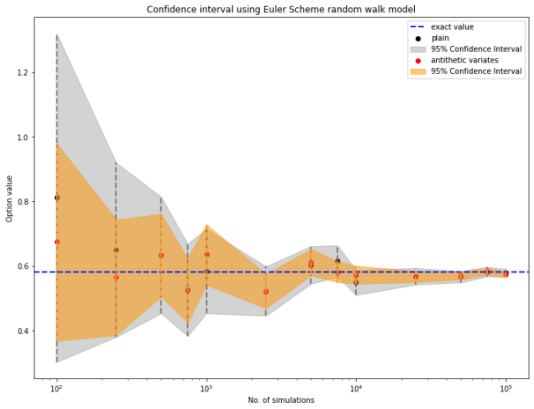
Table 33: Exact value of down-and-out Put Barrier option

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	0.8117	0.25956	0.0039	0.8099	0.25950	0.0011	0.8116	0.25954	0.0012
250	0.6501	0.13760	0.0054	0.6506	0.13758	0.0013	0.6501	0.13759	0.0019
500	0.6337	0.09191	0.0086	0.6592	0.09541	0.0033	0.6337	0.09190	0.0037
750	0.5259	0.07281	0.0193	0.5320	0.07270	0.0216	0.5258	0.07280	0.0089
1000	0.5840	0.06664	0.0222	0.5821	0.06691	0.0065	0.5840	0.06664	0.0085
2500	0.5224	0.03894	0.0410	0.5295	0.03934	0.0157	0.5224	0.03894	0.0229
5000	0.6031	0.02973	0.0792	0.5945	0.02945	0.0320	0.6031	0.02973	0.0419
7500	0.6154	0.02487	0.1815	0.6208	0.02506	0.0474	0.6154	0.02487	0.0621
10000	0.5488	0.01975	0.1670	0.5564	0.01994	0.0784	0.5488	0.01975	0.1577
25000	0.5682	0.01297	0.3897	0.5710	0.01302	0.1620	0.5681	0.01297	0.2297
50000	0.5667	0.00905	1.2492	0.5645	0.00903	0.5550	0.5669	0.00905	1.0944
75000	0.5824	0.00756	1.2751	0.5830	0.00756	0.7010	0.5825	0.00756	0.9671
100000	0.5782	0.00650	2.1363	0.5772	0.00649	1.2348	0.5782	0.00650	1.6928

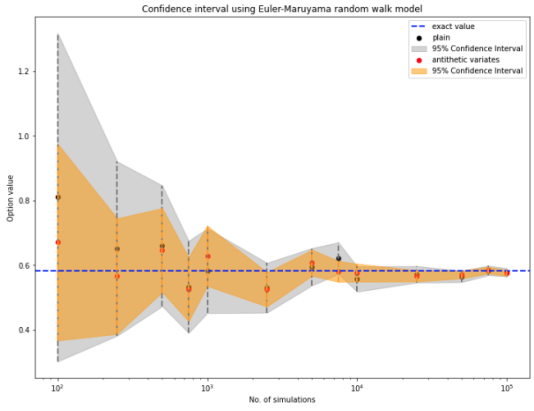
Table 34: Pricing of down-and-out Put option with crude Monte Carlo

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	0.6742	0.15597	0.0023	0.6709	0.15540	0.0011	0.6741	0.15596	0.0013
250	0.5646	0.09115	0.0042	0.5654	0.09118	0.0025	0.5645	0.09115	0.0475
500	0.6338	0.06511	0.0082	0.6457	0.06621	0.0056	0.6338	0.06511	0.0114
750	0.5238	0.05054	0.0129	0.5249	0.05042	0.0075	0.5237	0.05054	0.0110
1000	0.6354	0.04772	0.0163	0.6280	0.04746	0.0111	0.6354	0.04772	0.0165
2500	0.5224	0.02697	0.0509	0.5245	0.02708	0.0318	0.5223	0.02697	0.0564
5000	0.6118	0.02061	0.0840	0.6068	0.02050	0.0656	0.6118	0.02060	0.1490
7500	0.5820	0.01655	0.1249	0.5800	0.01652	0.1533	0.5820	0.01655	0.1334
10000	0.5727	0.01406	0.1731	0.5760	0.01409	0.1334	0.5726	0.01406	0.1810
25000	0.5677	0.00894	0.5019	0.5663	0.00892	0.3052	0.5679	0.00894	0.5347
50000	0.5710	0.00631	2.1178	0.5707	0.00631	1.1247	0.5711	0.00631	1.8064
75000	0.5834	0.00524	2.2538	0.5837	0.00524	2.5004	0.5835	0.00524	4.2383
100000	0.5751	0.00449	3.6806	0.5753	0.00449	4.5464	0.5752	0.00449	7.0053

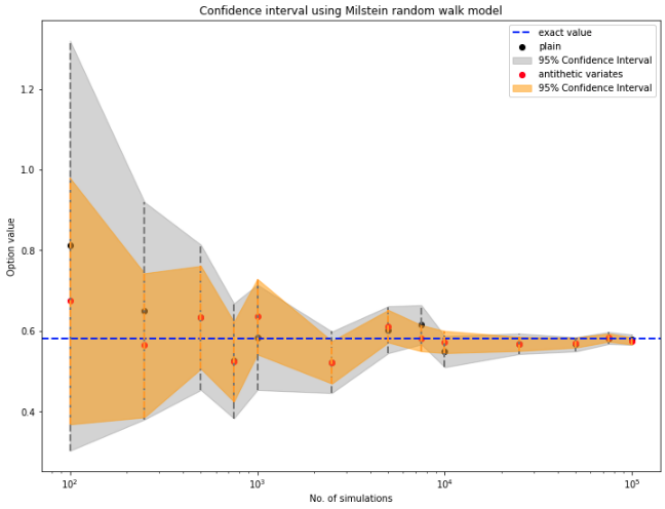
Table 35: Pricing of down-and-out Put option with antithetic variates



(a) Euler Scheme model



(b) Euler-Maruyama model



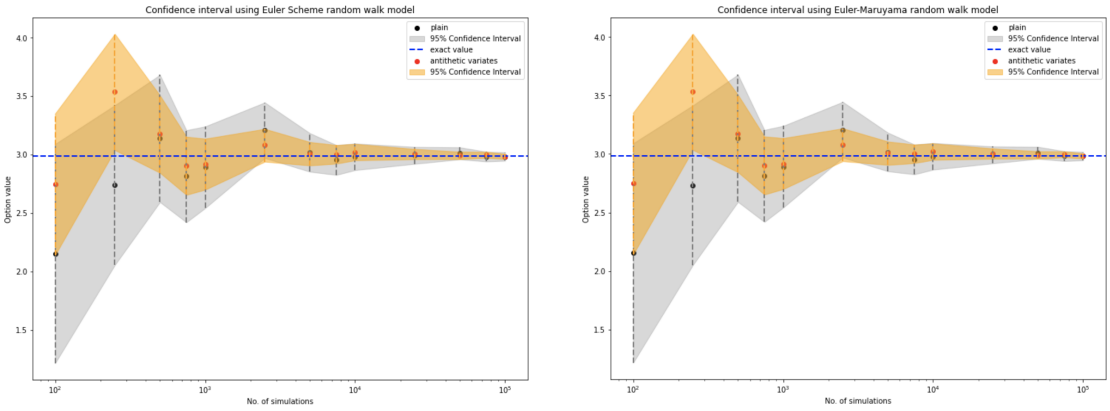
(c) Milstein model

Figure 17: Confidence Interval for down-and-out Put Barrier option

A.2 Asian Option simulation results

A.2.1 Geometric average rate Call option - Average of full length of contract

Analysis of results were given in subsection 7.2. Below are the confidence interval plots of Monte Carlo using Euler Scheme and Euler-Maruyama models.



(a) Euler Scheme model

(b) Euler-Maruyama model

Figure 18: Confidence Interval for geometric average rate call option - average of full contract length

A.2.2 Arithmetic average rate Call option - Average of full length of contract

Consider pricing of an Arithmetic average rate Call option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, E = 105$, $T_0 = 01/01/2024$, $T_1 = 01/01/2024$, $T_2 = 31/12/2024$ and $N = 365$.

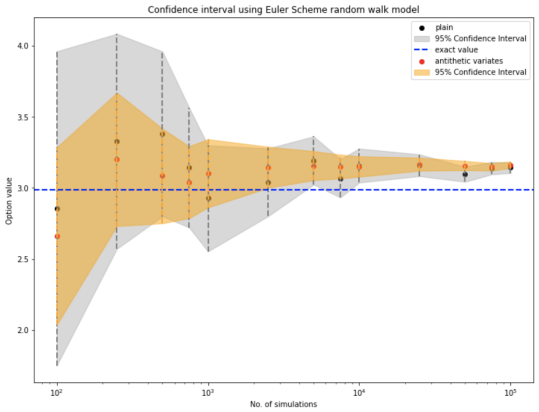
M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	2.8543	0.56416	0.0033	2.8470	0.56365	0.0009	2.8539	0.56408	0.0014
250	3.3272	0.38595	0.0073	3.3229	0.38557	0.0013	3.3268	0.38591	0.0020
500	3.3794	0.29604	0.0087	3.3798	0.29570	0.0033	3.3790	0.29600	0.0042
750	3.1431	0.21542	0.0127	3.1407	0.21525	0.0043	3.1427	0.21540	0.0059
1000	2.9260	0.19047	0.0173	2.9243	0.19010	0.0082	2.9256	0.19045	0.0153
2500	3.0391	0.12187	0.0354	3.0396	0.12183	0.0129	3.0387	0.12185	0.0162
5000	3.1924	0.08656	0.0725	3.1917	0.08649	0.0278	3.1920	0.08655	0.0349
7500	3.0670	0.06899	0.1145	3.0657	0.06894	0.1098	3.0666	0.06898	0.2246
10000	3.1565	0.06115	0.1717	3.1558	0.06112	0.0643	3.1561	0.06114	0.0919
25000	3.1588	0.03914	0.4533	3.1583	0.03911	0.1889	3.1584	0.03913	0.2486
50000	3.0954	0.02725	0.8762	3.0950	0.02723	0.4959	3.0950	0.02725	0.6977
75000	3.1378	0.02223	1.2530	3.1374	0.02221	0.7299	3.1374	0.02223	1.0308
100000	3.1417	0.01938	1.5302	3.1409	0.01937	1.0317	3.1413	0.01938	1.2786

Table 36: Pricing of arithmetic average rate call option - average of full length of contract with crude Monte Carlo

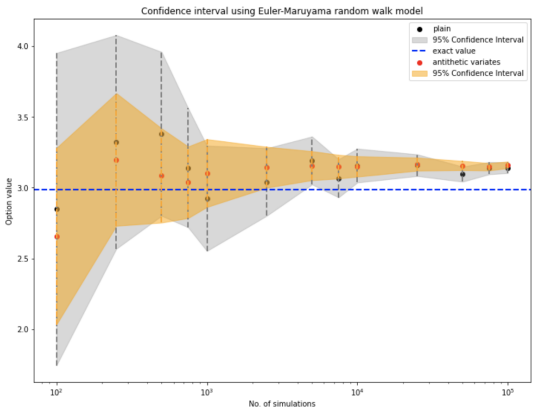
The exact value used in the confidence plot is the value derived from the closed form solution of geometric average rate call option, defined in Table 6. Arithmetic average rate tend to have higher estimated price than geometric average rate.

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	2.6604	0.31847	0.0032	2.6561	0.31846	0.0024	2.6600	0.31844	0.0021
250	3.2001	0.23956	0.0044	3.1987	0.23931	0.0023	3.1997	0.23953	0.0037
500	3.0835	0.17010	0.0077	3.0854	0.16994	0.0055	3.0832	0.17008	0.0084
750	3.0395	0.12926	0.0123	3.0374	0.12914	0.0080	3.0391	0.12925	0.0129
1000	3.1016	0.12256	0.0176	3.1024	0.12239	0.0092	3.1012	0.12255	0.0130
2500	3.1446	0.07378	0.0337	3.1440	0.07373	0.0234	3.1442	0.07377	0.0354
5000	3.1559	0.05229	0.0738	3.1549	0.05224	0.0477	3.1555	0.05228	0.0701
7500	3.1497	0.04287	0.1739	3.1479	0.04283	0.0958	3.1493	0.04286	0.1280
10000	3.1497	0.03682	0.1779	3.1495	0.03680	0.1180	3.1493	0.03681	0.1662
25000	3.1665	0.02365	0.4560	3.1658	0.02363	0.2866	3.1661	0.02365	0.4423
50000	3.1564	0.01670	1.5564	3.1558	0.01669	0.8507	3.1560	0.01670	1.6175
75000	3.1473	0.01355	2.3424	3.1469	0.01354	1.5844	3.1469	0.01355	4.3189
100000	3.1618	0.01176	2.7513	3.1611	0.01175	1.7433	3.1614	0.01176	6.0113

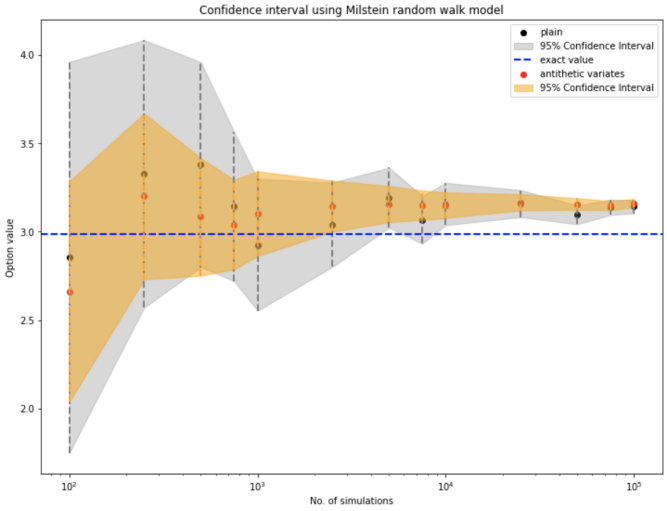
Table 37: Pricing of arithmetic average rate Call option - average of full length of contract with antithetic variates



(a) Euler Scheme model



(b) Euler-Maruyama model

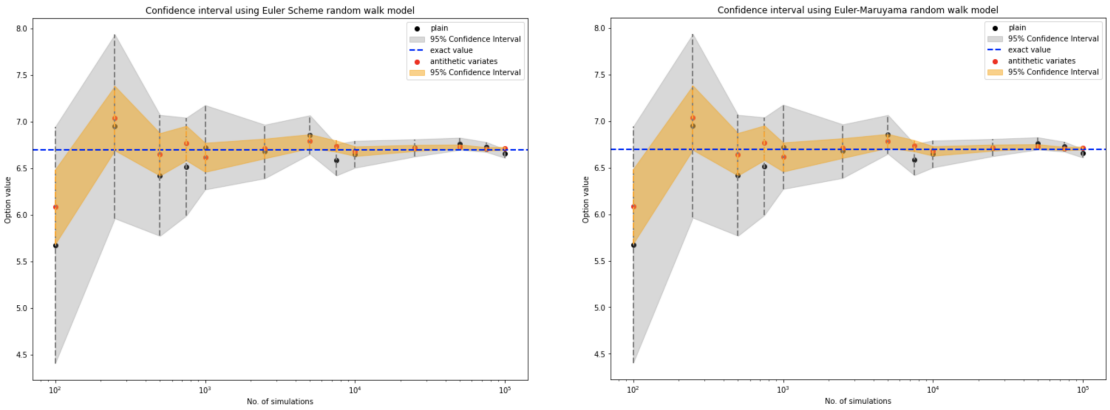


(c) Milstein model

Figure 19: Confidence Interval for arithmetic average rate call option - average of full contract length

A.2.3 Geometric average rate Put option - Average of full length of contract

Analysis of results were given in Example 7.2. Below are the confidence interval plots of Monte Carlo using Euler Scheme and Euler-Maruyama models.



(a) Euler Scheme model (b) Euler-Maruyama model

Figure 20: Confidence Interval for geometric average rate Put option - average of full contract length

A.2.4 Arithmetic average rate Put option - Average of full length of contract

Consider pricing of an Arithmetic average rate Put option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, E = 105$, $T_0 = 01/01/2024$, $T_1 = 01/01/2024$, $T_2 = 31/12/2024$ and $N = 365$.

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	5.8008	0.60548	0.0037	5.8041	0.60585	0.0005	5.8006	0.60544	0.0013
250	6.5108	0.43874	0.0037	6.5072	0.43843	0.0012	6.5105	0.43871	0.0017
500	7.0059	0.34077	0.0114	7.0050	0.34093	0.0044	7.0055	0.34075	0.0036
750	6.7733	0.27651	0.0115	6.7723	0.27658	0.0038	6.7730	0.27649	0.0043
1000	6.6852	0.23668	0.0170	6.6812	0.23668	0.0053	6.6848	0.23666	0.0059
2500	6.4805	0.14561	0.0361	6.4794	0.14564	0.0134	6.4802	0.14560	0.0177
5000	6.4902	0.10254	0.1167	6.4892	0.10256	0.0804	6.4899	0.10253	0.2736
7500	6.5544	0.08409	0.1188	6.5538	0.08410	0.0395	6.5540	0.08408	0.0528
10000	6.5311	0.07262	0.1671	6.5306	0.07265	0.0612	6.5307	0.07262	0.0843
25000	6.5767	0.04601	0.5906	6.5757	0.04602	0.3381	6.5764	0.04601	0.2644
50000	6.5299	0.03249	0.9198	6.5294	0.03250	0.5049	6.5295	0.03248	0.7443
75000	6.5269	0.02665	1.2088	6.5264	0.02666	0.6160	6.5266	0.02665	0.8735
100000	6.5503	0.02312	1.7358	6.5492	0.02312	0.8748	6.5500	0.02311	1.4129

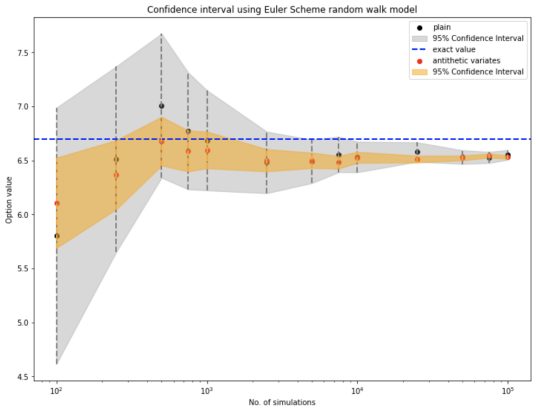
Table 38: Pricing of arithmetic average rate Put option - average of full length of contract with crude Monte Carlo

The exact value used in the confidence plot is the value derived from the closed form

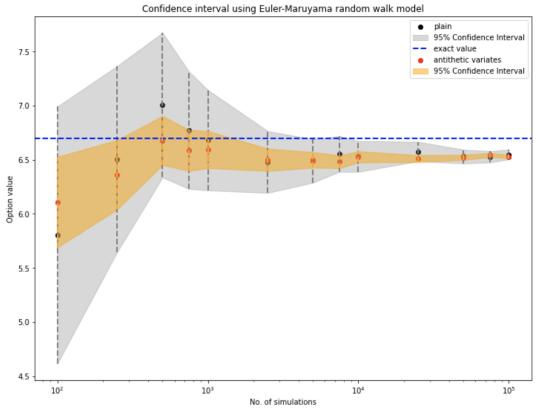
solution of geometric average rate put option, defined in Table 9. In this case, arithmetic average rate tend to have lower estimated price than geometric average rate.

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	6.1066	0.21325	0.0032	6.1081	0.21341	0.0020	6.1063	0.21323	0.0023
250	6.3666	0.16350	0.0040	6.3619	0.16362	0.0022	6.3662	0.16349	0.0077
500	6.6773	0.11546	0.0094	6.6779	0.11567	0.0059	6.6770	0.11545	0.0068
750	6.5866	0.09682	0.0095	6.5855	0.09689	0.0061	6.5863	0.09681	0.0090
1000	6.5961	0.08721	0.0131	6.5939	0.08734	0.0083	6.5958	0.08720	0.0122
2500	6.5007	0.05275	0.0448	6.4996	0.05279	0.0316	6.5004	0.05274	0.0517
5000	6.4995	0.03673	0.1321	6.4986	0.03676	0.0880	6.4992	0.03672	0.1090
7500	6.4820	0.03021	0.1126	6.4816	0.03022	0.0730	6.4817	0.03020	0.1145
10000	6.5275	0.02619	0.1611	6.5266	0.02623	0.1122	6.5271	0.02619	0.1652
25000	6.5113	0.01657	0.4226	6.5105	0.01659	0.3531	6.5110	0.01657	0.6132
50000	6.5221	0.01174	1.9082	6.5216	0.01175	0.9127	6.5218	0.01174	2.9949
75000	6.5437	0.00962	1.9982	6.5434	0.00963	3.2983	6.5434	0.00962	2.8279
100000	6.5296	0.00836	3.4080	6.5286	0.00837	2.2608	6.5293	0.00835	5.5015

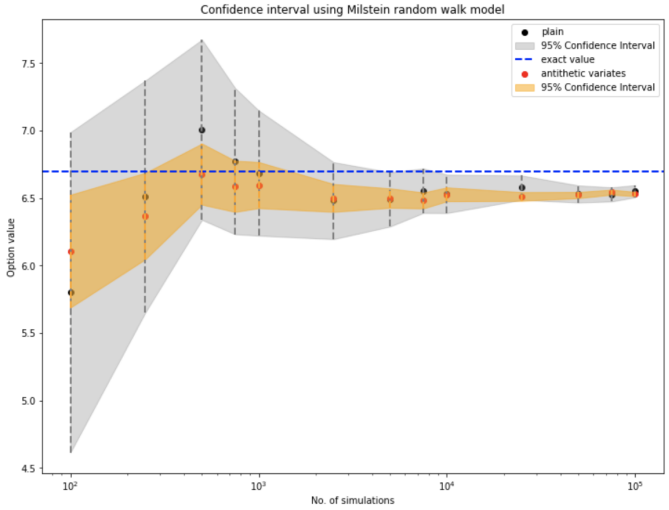
Table 39: Pricing of arithmetic average rate Put option - average of full length of contract with antithetic variates



(a) Euler Scheme model



(b) Euler-Maruyama model

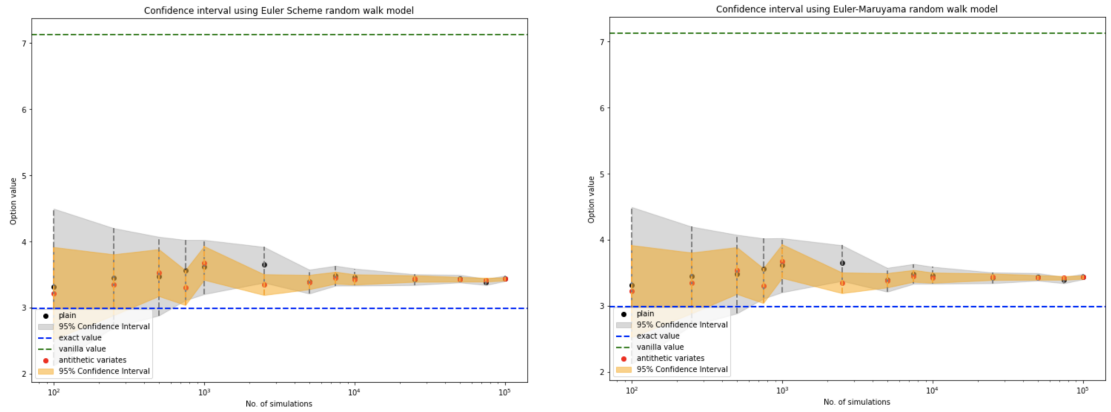


(c) Milstein model

Figure 21: Confidence Interval for arithmetic average rate Put option - average of full contract length

A.2.5 Geometric average rate Call option - Average of 30 days before expiry

Analysis of results were given in example Figure 7.2. Below are the confidence interval plots of Monte Carlo using Euler Scheme and Euler-Maruyama models.



(a) Euler Scheme model (b) Euler-Maruyama model

Figure 22: Confidence Interval for geometric average rate call option - average of 30 days before expiry

A.2.6 Geometric average rate Call option - Average of 60 days before expiry

Consider pricing of an Geometric average rate Call option with the following parameters: $S_0 = 100, \sigma = 0.2, r = 0.03, E = 105$, $T_0 = 01/01/2024$, $T_1 = 01/11/2024$, $T_2 = 31/12/2024$ and $N = 365$.

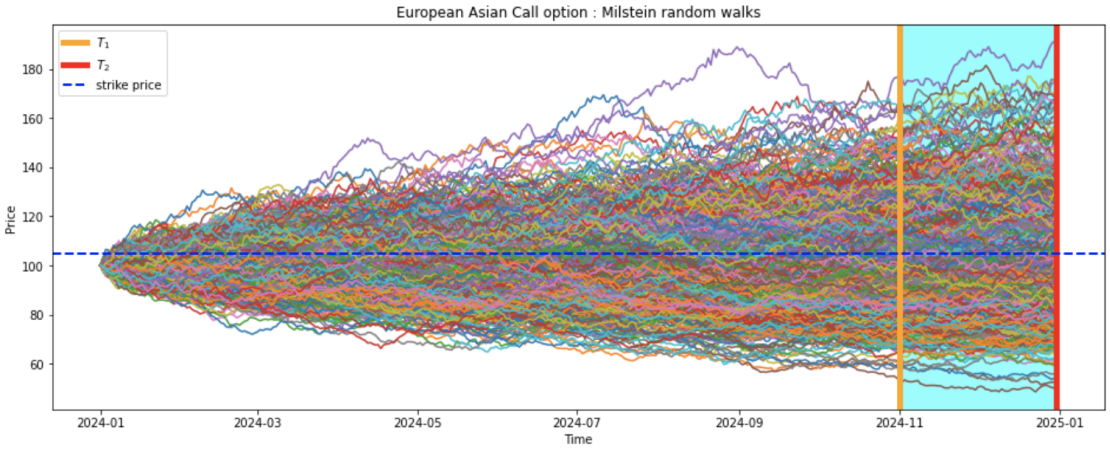


Figure 23: Monte Carlo simulation of geometric average rate Call option - average of 60 days before expiry

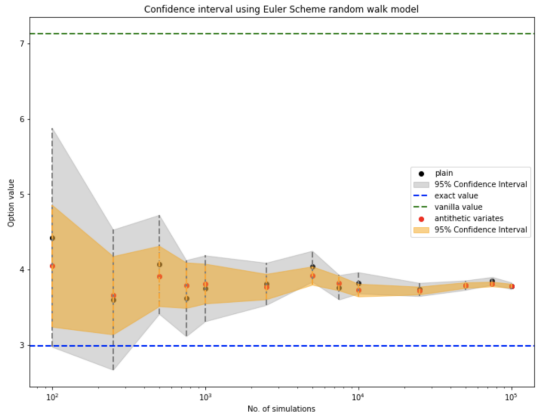
M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	4.4252	0.74066	0.0031	4.4250	0.74040	0.0011	4.4247	0.74058	0.0014
250	3.5992	0.47445	0.0138	3.5984	0.47449	0.0023	3.5988	0.47439	0.0027
500	4.0674	0.33515	0.0093	4.0683	0.33504	0.0042	4.0670	0.33512	0.0044
750	3.6194	0.25807	0.0142	3.6140	0.25769	0.0059	3.6190	0.25804	0.0062
1000	3.7500	0.22291	0.0180	3.7477	0.22282	0.0097	3.7495	0.22289	0.0103
2500	3.8089	0.14303	0.0477	3.8089	0.14295	0.0203	3.8085	0.14302	0.0227
5000	4.0424	0.10491	0.0812	4.0415	0.10486	0.0388	4.0419	0.10490	0.0470
7500	3.7635	0.08287	0.1217	3.7632	0.08282	0.0612	3.7631	0.08286	0.0786
10000	3.8225	0.07230	0.1719	3.8219	0.07226	0.0846	3.8220	0.07229	0.1045
25000	3.7348	0.04478	0.5347	3.7345	0.04476	0.2517	3.7344	0.04478	0.3146
50000	3.7915	0.03216	1.3532	3.7909	0.03214	0.6450	3.7910	0.03215	0.9077
75000	3.8496	0.02634	1.6670	3.8487	0.02633	1.1578	3.8491	0.02634	1.2094
100000	3.7842	0.02263	1.7679	3.7838	0.02262	1.1267	3.7838	0.02263	1.6168

Table 40: Pricing of geometric average rate Call option - average of 60 days before expiry with crude Monte Carlo

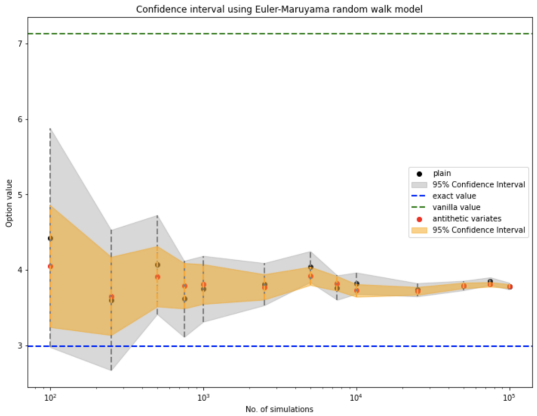
The exact value used in the confidence plot is the value derived from the closed form solution of geometric average rate put option, defined in Table 9. In this case, arithmetic average rate tend to have lower estimated price than geometric average rate.

M	Euler scheme	Error	Time	Euler Maruyama	Error	Time	Milstein	Error	Time
100	4.0472	0.41211	0.0019	4.0472	0.41202	0.0014	4.0467	0.41207	0.0025
250	3.6579	0.26495	0.0097	3.6529	0.26484	0.0050	3.6575	0.26492	0.0048
500	3.9136	0.20438	0.0091	3.9133	0.20427	0.0066	3.9132	0.20436	0.0085
750	3.7911	0.15413	0.0133	3.7875	0.15400	0.0098	3.7906	0.15411	0.0130
1000	3.8127	0.13449	0.0227	3.8106	0.13445	0.0174	3.8122	0.13448	0.0205
2500	3.7729	0.08597	0.0469	3.7725	0.08591	0.0357	3.7724	0.08596	0.0463
5000	3.9194	0.06208	0.1359	3.9184	0.06204	0.2465	3.9190	0.06207	0.1316
7500	3.8186	0.04968	0.1526	3.8185	0.04965	0.1148	3.8181	0.04968	0.1572
10000	3.7264	0.04300	0.2061	3.7258	0.04298	0.1573	3.7260	0.04300	0.2474
25000	3.7201	0.02675	0.6569	3.7193	0.02673	0.4257	3.7196	0.02675	0.7129
50000	3.7924	0.01928	2.6845	3.7919	0.01927	1.2358	3.7920	0.01928	1.8824
75000	3.8066	0.01568	2.9771	3.8058	0.01567	2.0634	3.8061	0.01568	4.8951
100000	3.7836	0.01354	3.5006	3.7830	0.01353	2.5046	3.7831	0.01354	6.8503

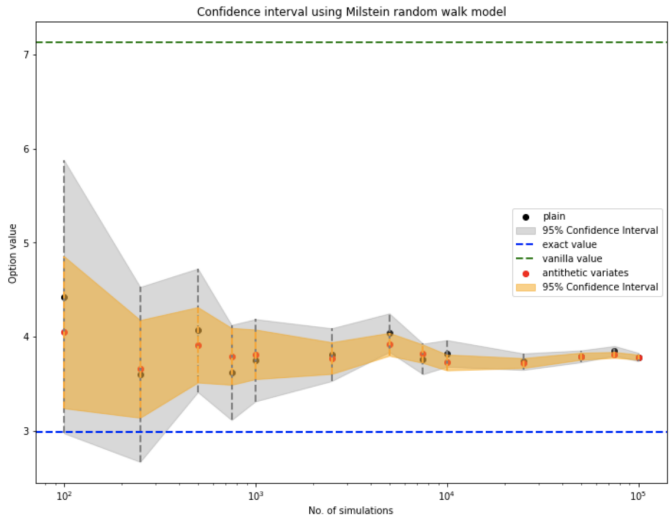
Table 41: Pricing of geometric average rate Call option - average of 60 days before expiry with antithetic variates



(a) Euler Scheme model



(b) Euler-Maruyama model



(c) Milstein model

Figure 24: Confidence Interval for geometric average rate Call option - average of 60 days before expiry

B Python Code

B.1 Random Walk Base Class

```
class BaseSimulator:
    def __init__(self, model_name, S0, K, r, sigma, T, N, M, T0, T2, phis=[]):
        self.model_name = model_name
        self.S0 = S0  # initial price
        self.K = K    # strike price
        self.r = r    # risk-free interest rate
        self.sigma = sigma # volatility
        self.T = T    # time to maturity in years
        self.N = N    # number of time steps
        self.dt = T/N # time step size
        self.M = M    # number of simulations
        self.T0 = T0  # start date
        self.T2 = T2  # maturity date
        self.realisations = None    # full path of realisations
        self.av_realisations = None # antithetic variate paths
        self.expiry_prices = None # expiry prices
        self.phis = phis # sequence of noise
        self.option_price = None # estimated option price
        self.payoffs = None # payoffs of each simulation
        self.antithetic_variates = False
        self.av_option_price = None # estimated option price with antithetic variate
        self.av_payoffs = None

    def get_option_price(self, option_type="c"):
        """
        calculate payoffs and dicounted option price

        Parameters:
            option_type (str): c -> call option, p -> put option
        """
        self.expiry_prices = self.realisations[-1]
        payoffs = (self.expiry_prices - self.K) if (option_type=="c")\
                                                else (self.K - self.expiry_prices)
        self.payoffs = maximum(payoffs, 0)

        if (self.antithetic_variates==True):
            av_expiry_prices = self.av_realisations[-1]
            av_payoffs = (av_expiry_prices - self.K) if (option_type=="c")\
                                                     else (self.K - av_expiry_prices)
            av_payoffs = maximum(av_payoffs, 0)
            self.av_payoffs = 0.5 * (self.payoffs + av_payoffs)
```

```
            self.av_option_price = np.exp(-self.r*self.T) * sum(self.av_payoffs)/self.M
            return self.av_option_price
        else:
            self.option_price = exp(-self.r*self.T) * sum(self.payoffs)/self.M
            return self.option_price

    def sample_path_model(self):
        # function for defining how asset sample path should be generated
        raise NotImplementedError("Subclass must implement abstract method")

    def generate_sample_paths(self, antithetic_variates=False):
        """
        function for generating random walks/realisations

        Parameters:
            antithetic_variates (bool): activate antithetic variate method
        """
        # initialise arrays of random variables
        if (len(self.phis) == 0):
            self.phis = np.random.normal(0, 1, size=(self.N, self.M))

        # generate sample paths with the specified path generating model
        self.antithetic_variates = antithetic_variates
        self.sample_path_model()

    def plot_n_realisations(self, n=0, display_mode=0):
        """
        Parameters:
            n (int): number of sample paths to plot
            display_mode (int): 0 -> plot standard realisations
                                1 -> plot realisations of antithetic varaites
                                2 -> plot both realisations
        """

        n = n if n > 0 else self.M
        timeline = pd.date_range(start=self.T0,end=self.T2)
        if (display_mode==1):
            plt.plot(timeline, self.av_realisations[:, :n])
        elif (display_mode==2):
            plt.plot(timeline, self.realisations[:, :n], color='blue')
            plt.plot(timeline, self.av_realisations[:, :n], color='orange')
            plt.plot(self.T0, self.S0, color='blue', label="standard")
            plt.plot(self.T0, self.S0, color='orange', label="antithetic variate")
            plt.legend()
```

```
        else:
            plt.plot(timeline, self.realisations[:, :n])

        plt.xlabel('Time')
        plt.ylabel('Value')
        plt.title(f'Simulated Time Series using {self.model_name} method')
        plt.show()
```

B.2 Euler Scheme Class

```
class Euler_discretisation_fast_simulator(BaseSimulator):
    def sample_path_model(self):
        drift = (self.r - 0.5*self.sigma**2)*self.dt
        diffusion = self.sigma*np.sqrt(self.dt)*self.phis
        delta_St = drift + diffusion
        self.realisations = self.S0*np.cumprod(np.exp(delta_St), axis=0)
        # prepend initial price S0 to the realisations
        self.realisations = np.concatenate(
            (np.full(shape=(1, self.M), fill_value=self.S0), self.realisations)
        )

        # use negative counterpart of the phis to generate antithetic variates
        if (self.antithetic_variates):
            delta_St_av = drift - diffusion
            self.av_realisations = self.S0*np.cumprod(np.exp(delta_St_av), axis=0)
            # prepend initial price S0 to the realisations
            self.av_realisations = np.concatenate(
                (np.full(shape=(1, self.M), fill_value=self.S0), self.av_realisations)
            )
```

B.3 Euler-Maruyama Class

```
class Euler_Maruyama_fast_simulator(BaseSimulator):
    def sample_path_model(self):
        delta_St = 1 + self.r*self.dt + self.sigma*np.sqrt(self.dt)*self.phis
        self.realisations = self.S0*np.cumprod(delta_St, axis=0)
        # prepend initial price S0 to the realisations
        self.realisations = np.concatenate(
            (np.full(shape=(1, self.M), fill_value=self.S0), self.realisations)
        )

        # use negative counterpart of the phis to generate antithetic variates
```

```
        if (self.antithetic_variates):
            delta_St_av = 1 + self.r*self.dt + self.sigma*np.sqrt(self.dt)*(-self.phis)
            self.av_realisations = self.S0*np.cumprod(delta_St_av, axis=0)
            # prepend initial price S0 to the realisations
            self.av_realisations = np.concatenate(
                (np.full(shape=(1, self.M), fill_value=self.S0), self.av_realisations)
            )
```

B.4 Milstein Class

```
class Milstein_fast_simulator(BaseSimulator):
    def sample_path_model(self):
        delta_St = 1 + self.r*self.dt +
                   self.sigma*np.sqrt(self.dt)*self.phis +
                   0.5*(self.sigma**2)*(np.square(self.phis)-1)*self.dt
        self.realisations = self.S0*np.cumprod(delta_St, axis=0)
        # prepend initial price S0 to the realisations
        self.realisations = np.concatenate(
            (np.full(shape=(1, self.M), fill_value=self.S0), self.realisations)
        )

        # use negative counterpart of the phis to generate antithetic variates
        if (self.antithetic_variates):
            delta_St_av = 1 + self.r*self.dt - self.sigma*np.sqrt(self.dt)*self.phis +
            0.5*(self.sigma**2)*(np.square(self.phis)-1)*self.dt
            self.av_realisations = self.S0*np.cumprod(delta_St_av, axis=0)
            # prepend initial price S0 to the realisations
            self.av_realisations = np.concatenate(
                (np.full(shape=(1, self.M), fill_value=self.S0), self.av_realisations)
            )
```

B.5 Monte Carlo Class

```
class MC_simulator_optimised:
    def __init__(self, S0, K, r, sigma, T, N, M, T0, T2, simulator, phis=[]):
        self.S0 = S0  # initial asset price
        self.K = K    # strike price
        self.r = r    # risk-free interest rate
        self.sigma = sigma # volatility
        self.T = T    # Final time
        self.N = N    # number of time steps
        self.M = M    # number of simulations
```

```
        self.T0 = T0  # start date
        self.T2 = T2  # maturity date
        self.dt = T/N # time step size
        self.simulator  = simulator # asset path generator
        self.phis = phis # sequence of noise
        self.mil_simulator = None
        self.em_simulator = None
        self.ed_simulator = None
        self.antithetic_variates = False

    def standard_error(self, payoffs, option_price):
        # sample standard deviation
        std_dev = np.sqrt(np.sum((exp(-self.r*self.T)*payoffs -\
                  option_price)**2) / (self.M-1))

        # standard error
        SE = std_dev/np.sqrt(self.M)
        return SE

    def get_European_option_price(self, option_type):
        # price the option
        option_price = self.simulator.get_option_price(option_type)
        SE = 0
        if (self.antithetic_variates):
            SE = self.standard_error(self.simulator.av_payoffs,
                 self.simulator.av_option_price)
        else:
            SE = self.standard_error(self.simulator.payoffs,
                 self.simulator.option_price)
        return (option_price, SE)

    def run_monte_carlo(self, antithetic_variates):
        self.antithetic_variates = antithetic_variates
        self.simulator.generate_sample_paths(antithetic_variates)

    def update_model(self, simulator_model):
        self.simulator = simulator_model

    def comparing_ith_realisations(self, simulators, ith_rls):
        timeline = pd.date_range(start=self.T0,end=self.T2)
        plt.figure(figsize=(13, 6))

        for simulator in simulators:
            plt.plot(timeline, simulator.realisations[:,ith_rls],\
```

```
            label=simulator.model_name)
        plt.xlabel('Time')
        plt.ylabel('Value')
        plt.title('Comparing Simulated Time Series of three random walk models')
        plt.legend()
        plt.show()

    def plot_n_realisations(self, simulators, n=0):
        fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(13, 6))
        n = n if n > 0 else self.M
        timeline = pd.date_range(start=self.T0,end=self.T2)

        # plots Euler discretisation realisations
        axes[0, 0].plot(timeline, simulators[0].realisations[:, :n])
        axes[0, 0].set_xlabel('Time')
        axes[0, 0].set_ylabel('Value')
        axes[0, 0].set_title('Simulated Time Series using Euler discretisation method')

        # plots Euler-Maruyama realisations
        axes[0, 1].plot(timeline, simulators[1].realisations[:, :n])
        axes[0, 1].set_xlabel('Time')
        axes[0, 1].set_ylabel('Value')
        axes[0, 1].set_title('Simulated Time Series using Euler-Maruyama method')

        # plots Milstein realisations
        axes[1, 0].plot(timeline, simulators[2].realisations[:, :n])
        axes[1, 0].set_xlabel('Time')
        axes[1, 0].set_ylabel('Value')
        axes[1, 0].set_title('Simulated Time Series using Milstein method')

        axes[1, 1].axis('off')
        fig.tight_layout()
        plt.show()
```

B.6 Barrier Option Class

```
class BarrierOption(MC_simulator_optimised):
    def __init__(self, S0, K, r, sigma, T, N, M, T0, T2,
                 simulator, barrier_type, H_bound, L_bound):
        super().__init__(S0, K, r, sigma, T, N, M, T0, T2, simulator)
        self.barrier_type = barrier_type # int: corresponds to the indexes listed above
        self.H_bound = H_bound   # upper barrier
        self.L_bound = L_bound   # lower barrier
```

```
        self.mask_1 = None
        self.mask_2 = None
        self.title = 'European Up-and-Out Call Option'
        self.main_type = "Call" if (barrier_type < 5) else "Put"
        self.barrier_price = 0
        self.barrier_payoffs = None

    def calculate_option_price(self):
        """
        barrier_type < 5  : call option => ST - K
        barrier_type >= 5 : put option  => K - ST
        """
        S_expiry = self.simulator.realisations[-1,:].copy()
        S_expiry[self.mask_1] = self.K
        payoffs = (S_expiry - self.K) if (self.barrier_type < 5)\
                                      else (self.K - S_expiry)
        payoffs = np.maximum(0, payoffs)

        if (self.antithetic_variates):
            S_expiry_2 = self.simulator.av_realisations[-1,:].copy()
            S_expiry_2[self.mask_2] = self.K
            payoffs_2 = (S_expiry_2 - self.K) if (self.barrier_type < 5)\
                                              else (self.K - S_expiry_2)
            payoffs_2 = np.maximum(0, payoffs_2)
            av_payoffs = 0.5*(payoffs + payoffs_2)
            option_price = np.exp(-self.r*self.T)*np.sum(av_payoffs)/self.M
            self.barrier_payoffs = av_payoffs
            return option_price
        else:
            option_price = np.exp(-self.r*self.T)*np.sum(payoffs)/self.M
            self.barrier_payoffs = payoffs
            return option_price

    def up_and_in(self):
        self.title = 'European Up-and-In ' +\
                     ('Call Option' if (self.barrier_type < 5) else 'Put Option')
        # change realisation values to 0 based on indexing mask
        mask = np.any(self.simulator.realisations >= self.H_bound, axis=0)
        # reversing mask so realisations below barrier are nullified
        self.mask_1 = ~mask

        if (self.antithetic_variates):
            # change realisation values to 0 based on indexing mask
            mask_2 = np.any(self.simulator.av_realisations >= self.H_bound, axis=0)
```

```
            # reversing mask so realisations below barrier are nullified
            self.mask_2 = ~mask_2
        return self.calculate_option_price()

    def up_and_out(self):
        self.title = 'European Up-and-Out ' +\
                     ('Call Option' if (self.barrier_type < 5) else 'Put Option')
        # change realisation values to 0 based on indexing mask
        self.mask_1 = np.any(self.simulator.realisations >= self.H_bound, axis=0)

        if (self.antithetic_variates):
            # change realisation values to 0 based on indexing mask
            self.mask_2 = np.any(self.simulator.av_realisations >= self.H_bound, axis=0)
        return self.calculate_option_price()

    def down_and_in(self):
        self.title = 'European Down-and-In ' +\
                     ('Call Option' if (self.barrier_type < 5) else 'Put Option')
        # change realisation values to 0 based on indexing mask
        mask = np.any(self.simulator.realisations <= self.L_bound, axis=0)
        # reversing mask so realisations above barrier are nullified
        self.mask_1 = ~mask

        if (self.antithetic_variates):
            # change realisation values to 0 based on indexing mask
            mask_2 = np.any(self.simulator.av_realisations <= self.L_bound, axis=0)
            # reversing mask so realisations below barrier are nullified
            self.mask_2 = ~mask_2
        return self.calculate_option_price()

    def down_and_out(self):
        self.title = 'European Down-and-Out ' +\
                     ('Call Option' if (self.barrier_type < 5) else 'Put Option')
        # change realisation values to 0 based on indexing mask
        self.mask_1 = np.any(self.simulator.realisations <= self.L_bound, axis=0)

        if (self.antithetic_variates):
            # change realisation values to 0 based on indexing mask
            self.mask_2 = np.any(self.simulator.av_realisations <= self.L_bound, axis=0)
        return self.calculate_option_price()

    def get_Barrier_price(self):
        # price option based on barriers
        select_barrier = {
```

```
            1 : self.up_and_in,
            2 : self.up_and_out,
            3 : self.down_and_in,
            4 : self.down_and_out,
            5 : self.up_and_in,
            6 : self.up_and_out,
            7 : self.down_and_in,
            8 : self.down_and_out
        }
        self.barrier_price = select_barrier.get(self.barrier_type, self.up_and_in)()
        SE = self.standard_error(self.barrier_payoffs, self.barrier_price)
        return (self.barrier_price, SE)

    def plot_barrier_paths(self):
        plt.figure(figsize=(16, 6))
        timeline = pd.date_range(start=self.T0,end=self.T2)
        random_walk = self.simulator.av_realisations if self.antithetic_variates \
                      else self.simulator.realisations
        mask = self.mask_2 if self.antithetic_variates else self.mask_1
        # plot realisations
        plt.plot(timeline, random_walk[:,~mask],'g', alpha=0.5)
        plt.plot(self.T0, self.S0, 'g', label='active')
        plt.plot(timeline, random_walk[:,mask],'r', alpha=0.3)
        plt.plot(self.T0, self.S0, 'r', label='inactive')

        # plot barriers
        if (self.barrier_type < 3 or (self.barrier_type > 4 and self.barrier_type < 7)):
            plt.axhline(y=self.H_bound, color='k', linestyle='--',
                        linewidth=2.0, label='H_barrier')
        else:
            plt.axhline(y=self.L_bound, color='k', linestyle='--',
                        linewidth=2.0, label='L_barrier')

        plt.axhline(y=self.K, linestyle='--', color='blue',linewidth=2.0,
                    label='strike price')
        plt.xlabel('Time')
        plt.ylabel('Price')
        _title = self.title + " : " + self.simulator.model_name + " random walks"
        plt.title(_title)
        plt.legend()
        plt.show()
```

B.7 Asian Option Class

```
class AsianOption(MC_simulator_optimised):
    def __init__(self, S0, K, r, sigma, T, N, M, T0, T2, T1, simulator):
        super().__init__(S0, K, r, sigma, T, N, M, T0, T2, simulator)
        self.T1 = T1 # start date of average price period
        self.em_mask = None # mask for Euler-Maruyama
        self.m_mask = None  # mask for Milstein
        self.title = 'European Asian'
        self.main_type = "Call"
        self.asian_price = None
        self.avg_prices = None
        self.asian_payoffs = None

    def arithmetic_avg(self, realisations):
        return np.mean(realisations, axis=0)

    def geometric_avg(self, realisations):
        return np.exp(np.mean(np.log(realisations), axis=0))

    def calculate_average_strike(self, start_point, option_type, avg_type):
        rls = self.simulator.realisations[-start_point:, :]
        avg_prices = self.arithmetic_avg(rls) if (avg_type=="arithmetic")\
                                              else self.geometric_avg(rls)
        S_expiry = self.simulator.realisations[-1,:]

        payoffs = (S_expiry - avg_prices) if (option_type=="c")\
                                          else (avg_prices - S_expiry)
        payoffs = maximum(payoffs, 0)

        if (self.antithetic_variates):
            rls_2 = self.simulator.av_realisations[-start_point:, :]
            avg_prices_2 = self.arithmetic_avg(rls_2) if (avg_type=="arithmetic")\
                                                      else self.geometric_avg(rls_2)
            S_expiry_2 = self.simulator.av_realisations[-1,:]

            payoffs_2 = (S_expiry_2 - avg_prices_2) if (option_type=="c")\
                                                    else (avg_prices_2 - S_expiry_2)
            payoffs_2 = maximum(payoffs_2, 0)
            av_payoffs = 0.5 * (payoffs + payoffs_2)
            option_price = np.exp(-self.r*self.T) * np.sum(av_payoffs)/self.M
            self.asian_payoffs = av_payoffs
            return option_price
        else:
```

```
            option_price = exp(-self.r*self.T) * np.sum(payoffs)/self.M
            self.asian_payoffs = payoffs
            return option_price

    def calculate_average_rate(self, start_point, option_type, avg_type):
        rls = self.simulator.realisations[-start_point:, :]
        avg_prices = self.arithmetic_avg(rls) if (avg_type=="arithmetic")\
                                              else self.geometric_avg(rls)

        payoffs = (avg_prices - self.K) if (option_type=="c") else (self.K - avg_prices)
        payoffs = maximum(payoffs, 0)

        if (self.antithetic_variates):
            rls_2 = self.simulator.av_realisations[-start_point:, :]
            avg_prices_2 = self.arithmetic_avg(rls_2) if (avg_type=="arithmetic")\
                                                      else self.geometric_avg(rls_2)
            payoffs_2 = (avg_prices_2 - self.K) if (option_type=="c")\
                                                else (self.K - avg_prices_2)
            payoffs_2 = maximum(payoffs_2, 0)
            av_payoffs = 0.5 * (payoffs + payoffs_2)
            option_price = np.exp(-self.r*self.T) * np.sum(av_payoffs)/self.M
            self.asian_payoffs = av_payoffs
            return option_price
        else:
            option_price = exp(-self.r*self.T) * np.sum(payoffs)/self.M
            self.asian_payoffs = payoffs
            return option_price

    def get_average_rate_Asian_price(self, option_type, avg_type):
        self.main_type = "Call" if option_type=='c' else "Put"
        self.title = self.title + ' ' + self.main_type + " option"

        # starting date of time period for calculating average
        start_point = (self.T1 - self.T0).days -1

        self.asian_price = self.calculate_average_rate(start_point,
                                                       option_type, avg_type)
        SE = self.standard_error(self.asian_payoffs, self.asian_price)
        return (self.asian_price, SE)

    def get_average_strike_Asian_price(self, option_type, avg_type):
        self.main_type = "Call" if option_type=='c' else "Put"
        self.title = self.title + ' ' + self.main_type + " option"

        # starting date of time period for calculating average
```

```
        start_point = (self.T1 - self.T0).days -1
        self.asian_price = self.calculate_average_strike(start_point,
                                                         option_type, avg_type)
        SE = self.standard_error(self.asian_payoffs, self.asian_price)
        return (self.asian_price, SE)

    def plot_asian_paths(self):
        plt.figure(figsize=(16, 6))
        random_walk = self.simulator.av_realisations if self.antithetic_variates \
                                                     else self.simulator.realisations

        # plot realisations
        timeline = pd.date_range(start=self.T0,end=self.T2)
        plt.plot(timeline, random_walk)

        # highlight average price of region
        plt.axvspan(self.T1, self.T2, color="cyan", alpha=0.5)
        plt.axvline(x=self.T1, color="orange",
                    linestyle='-', linewidth=5, label='$T_1$')
        plt.axvline(x=self.T2, color="red", linestyle='-', linewidth=5, label='$T_2$')

        # plot strike price
        plt.axhline(y=self.K, linestyle='--', color='blue',
                    linewidth=2.0, label='strike price')

        plt.xlabel('Time')
        plt.ylabel('Price')
        _title = self.title + " : " + self.simulator.model_name + " random walks"
        plt.title(_title)
        plt.legend()
        plt.show()
```