Monte Carlo technique: Project D

MATH4061 Advanced Financial Mathematics
Group project report
2022/23

School of Mathematical Sciences
University of Nottingham

Group 3:
Munera Alanazi
Alvin de la Cruz
Kieran Lee
Lual Muortat
Oluwatobiloba Oyediran

Abstract

In this report we will introduce the basic idea behind the Monte Carlo method and how it is used as a tool in the pricing of financial derivatives, subsequently, we will cover numerical methods for SDEs and the convergence of such schemes with emphasis on weak approximations. We then briefly cover random number generators and what influenced our choice when implementing the Monte Carlo method. Supported by computational tasks we have carried out, we analyse the practical performance of the Euler scheme and Monte Carlo method, observing convergence in both. Lastly, we overview methods for speeding up the Monte Carlo method by reducing variance, through analysing the results from our implementation of the Control varieties method for pricing a vanilla call option we find that it is an effective method for reducing variance.

Contents

1 Introduction

In this project, we've been tasked with using numerical methods to approximate the Stochastic differential equation corresponding to geometric Brownian motion and approximating the prices of two types of European call options. This will require us to use Monte Carlo methods that allow us to simulate a large number of trajectories and then find an expected price on the option by averaging them out. In the context of option pricing, we need to use a numerical method for solving an SDE that satisfies the weak convergence criteria, this is because we are concerned with the prices computed from our approximation being accurate and necessarily the paths of the processes [2]. We will use the Euler-Maruyama scheme due to it being relatively easy to implement and sufficiently accurate. Fundamental to a good Monte Carlo simulation is a seemingly random sequence of numbers, many random number generators can be used to do this. In our case, Permuted Congruential Generators (PNGs) seem to be a suitable choice due to it's speed and good statistical properties, to transform our uniform sequence of random variables to the normal distribution we'll use some type of rejection method.

Comparing our resulting approximations to the exact price, which in this instance we can find since the SDE can be solved, will tell us how well the methods perform in terms of the convergence of the Euler scheme and Monte Carlo. In a practical setting it is essential to be able to implement an efficient and fast Monte Carlo method, various variance reduction methods can be used to achieve this. Through further study and implementation of some of these techniques such as Control Variate and Antithetic Variate, we will explore their effectiveness.

We begin the report by discussing Monte Carlo methods in more detail. Section 2 will give a background of Monte Carlo methods as well as discuss the analysis of errors arising due to the use of this technique. In section 3, we introduce stochastic numerics, dissect their properties and explain their use in this project. Section 4 begins with a brief overview of the random number generating procedure we use before conveying the practical simulations we have run and analysing the results. Finally, in section 5, we discuss techniques to speed up the convergence of the Monte Carlo method and review the practical implementation of one such technique.

2 Monte Carlo methods

2.1 Introduction

Monte Carlo methods are numerical methods that uses simulation and statistical sampling to solve mathematical problems. It is named after the famous Monte Carlo Casino in Monaco. Essentially, Monte Carlo methods generates a large number of random samples or simulations and then use them to approximate the behaviour of a process or system. In finance, it is commonly used to estimate the price of a derivative by simulating the patterns of the underlying asset using random price movements.

2.2 Main Principle

Monte Carlo methods are grounded on the relationship between probability and volume, which is formalized mathematically using the measure theory. This theory associates an event with a set of possible outcomes, defining the probability of the event as its measure or volume relative to that of a plethora of potential outcomes. In Monte Carlo simulations, this relationship is used in reverse to find the volume of a set by interpreting the volume as a probability. This involves randomly sampling from a set of possible outcomes and taking the fraction of samples that fall within a given set as the estimate of its volume. As the number of draws increases, the estimate converges to the true value due to the law of large numbers.

Consider we have a function f over a unit integral such that

$$\alpha = \int_0^1 f(x)dx \tag{2.1}$$

The point $U_1, U_2, ..., U_n$ are independent and uniform from [0, 1]. Evaluating the function f at n of these random points and averaging the results produces the Monte Carlo estimate

$$\hat{\alpha_n} = \frac{1}{n}\sum_{i=1}^{n} f(U_i) \tag{2.2}$$

The estimator is strongly consistent, that is, as $n \to \infty$

$$\alpha_n \to \hat{\alpha_n} \qquad\qquad \text{with probability 1} \tag{2.3}$$

The variance can be estimated using the sample standard deviation

$$v = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(f(U_i) - \hat{\alpha_n})^2} \tag{2.4}$$

The Monte Carlo error $\hat{\alpha_n} - \alpha_n$ is approximately 0 and it is normally distributed with a variance $\frac{v^2}{n}$. Essentially, it is an N(0, 1)[1] random variable scaled by $v/\sqrt{n}$. The ratio $v/\sqrt{n}$ is often referred to as the standard error. Reducing this error by half requires increasing the number of points by a factor of four; adding one decimal place of precision will require 100 times as many points.

2.3 Practical Aspects

What is the relevance of Monte Carlo methods in financial engineering? A fundamental implication of asset pricing theory is that under certain circumstances, the price of derivatives can be represented as expected values. However, quite often, if we tried to write these expectations as integrals, their dimension gets very large (sometimes infinite). Monte Carlo methods provide a useful solution to this. Using Monte Carlo methods to price derivatives involves simulating paths of stochastic processes used to describe the evolution of underlying relevant factors such as asset prices, interest rates, model parameters, etc. The dimension will then be as large as the number of time steps in the simulation, which makes the square-root convergence rate for Monte Carlo methods competitive in comparison to other methods. The resulting Monte Carlo algorithm can be summarized as follows:

```
for i = 1 to n
    draw $\epsilon_i$ from a N(0,1) distribution
    set $S_i = S_0 e^{(r-\frac{1}{2}\sigma^2)T+\sigma\sqrt{T}\epsilon_i}$
    set $f(U_i) = e^{-rT}\Delta(S_i)$            where $\Delta(S_i) = S_i - S_{i-1}$
end
set $\hat{\alpha_n} = \frac{1}{n}\sum_{i=1}^{n} f(U_i)$
set $v = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(f(U_i) - \hat{\alpha_n})^2}$
```

The output provides an approximate option price $\hat{\alpha_n}$ and error v.

[1]Normal distribution with mean 0, variance 1 (standard normal distribution)

3 Stochastic Numerics

3.1 Introduction

Stochastic Differential Equations (SDEs) are used to model the behaviour of assets that have randomness (e.g stock prices). To solve these SDEs, we use numerical methods to approximate the solutions. We need to measure the closeness between the exact solutions and their approximations. However, this can be challenging because the approximations depend on randomness and may not be smooth. Also, the trajectories of SDEs are usually not differentiable, which makes it difficult to prove the convergence of numerical schemes, hence we have to consider a variety of approaches to approximate the solutions to SDEs. Weak approximation methods in the category of Euler-type schemes are useful in approximating the solutions to SDEs. In this chapter, we will examine the basics of stochastic numerics involved in developing numerical schemes to approximate solutions to SDEs, evaluate sources of error, and discuss the convergence rate.

3.2 Stochastic Differential Equation

A Stochastic Differential Equation (SDE) describes the time evolution of a system subject to randomness. It is an equation that contains drift and stochastic components. Consider an SDE of the form

$$dX_t = a(X_t)dt + b(X_t)dW_t \tag{3.1}$$

for $t \in [0, T]$.

X_t is the solution of the SDE. $a(X_t)$ is the drift coefficient, $b(X_t)$ is the diffusion coefficient, $W_t = W(t)$ is a Wiener process, and dt represents the time increment.

3.3 Numerical methods for Stochastic Differential Equations

Numerical methods for SDEs involve approximating the solution of an SDE using a discrete-time approximation. This discretization can be seen as a random walk over a finite time interval where each step depends on the current position and some random realization. The main challenge in numerical methods for SDEs is how to approximate the Wiener process.

The simplest numerical method used for SDEs is the Euler-Maruyama scheme, an extension of the standard Euler method used for ordinary differential equations (ODEs). The Euler-Maruyama scheme is an explicit method, that is, the approximation at the next time step is obtained by adding drift and noise terms to the approximation at the current time step. Some other numerical methods include the Milstein scheme, the Runge-Kutta scheme, and the stochastic Taylor expansion scheme.

The choice of numerical method depends on the factors such as the order of accuracy required, the stability of the method, and the computational efficiency. An important aspect of numerical methods for SDEs is the choice of step size, which can affect both the accuracy and stability of the scheme. The accuracy of the numerical scheme is typically measured in terms of the strong error, which is the difference between the true solution and the numerical approximation at a fixed time point.

When using numerical methods to approximate SDEs, it is important to consider the sources of error in the approximation. Two sources of error are discretization error and approximation error. Discretization error arises due to the use of discrete time steps, while approximation error arises due to the use of a numerical scheme to approximate the solution.

3.4 Consistency, Convergence and Stability

To ensure an approximation will be reasonably accurate, reliable, and can be computed efficiently; the unknown discretization and approximation errors have to be sufficiently small. This can be achieved by checking if the method is consistent with the differential equation if the estimates of the global discretization error converge to zero within the maximum time step, and if the method is stable, that is if propagated errors remain bounded.

Given a differential equation

$$\frac{dx}{dt} = a(t, x), \tag{3.2}$$

on any finite time interval $[t_0, T]$ and a one-step method

$$y_{n+1} = y_n + \psi(t_n, y_n, \Delta n)\Delta n \tag{3.3}$$

n = 0,1,..,N, $y_0 = y(t_0), y_N = y(t_N) = y(T)$

where a(t,x) and its partial derivatives are continuous and the increment function $\psi = \psi(t, y, \Delta)$ is continuous and satisfies a lipschitz condition in x.

The one-step method is **consistent** if it satisfies

$$\psi(t, x, 0) = a(t, x) \tag{3.4}$$

everywhere.

The one-step method is **convergent** if the global discretization error converges to 0 with the maximum time step $\Delta = \max_n \Delta n$, i.e

$$\lim_{\Delta \to 0} |e_{n+1}| = \lim_{\Delta \to 0} |x(t_{n+1}; t_0, x_0) - y_{n+1}| = 0 \tag{3.5}$$

where $y_0 = t_0$.

The one-step method is numerically **stable** if for each interval $[t_0, T]$ and differential equation eq. (3.1) with $a(t, x)$ satisfying the Lipschitz condition, there exist positive constants Δ_o and M such that

$$|y_n - \hat{y_n}| \leq M\,|y_0 - \hat{y_0}|\,, \qquad \forall n \tag{3.6}$$

3.5 Discrete time approximations of SDEs

In general, it is impossible to find explicitly the solution $X = X(t; t_0, x_0)$ of an initial value problem (IVP) (as eq. (3.2)) for the deterministic differential equations that occur in many models. Even when the solution exists, it may be only in implicit form or be very complicated to visualize and evaluate numerically. Necessity has thus led to the development of methods for calculating numerical approximations to the solutions of such initial value problems. The most widely applicable and commonly used of these is the time discrete approximation or difference methods, in which the continuous-time differential equation is replaced by a discrete-time difference equation generating values $Y_1, Y_2, ..., Y_n, ...$ to approximate $X(t_1; t_0, x_0), X(t_2; t_0, x_0), ..., X(t_n; t_0, x_0), ...$ at given discretization times $t_0 < t_1 < t_2 < ... < t_N$. These approximations are accurate, if the time increments $\delta n = t_{n+1} - t_n$ for $n = 0, 1, 2, ..., N$ are sufficiently small.

The simplest discrete approximation for the IVP (1.1) is the Euler method (see [1])

$$y_{n+1} = y_n + a(t_n, y_n)\Delta n \tag{3.7}$$

for a given time discretization $t_0 < t_1 < t_2 < ... < t_N$ with increments $\Delta n = t_{n+1} - t_n$ where $n = 0, 1, 2, ..., N$. The approximations $y_1, y_2, ..., y_N$ can be calculated by recursively applying eq. (3.7).

The difference

$$l_{n+1} = x(t_{n+1}; t_n, x_n) - y_{n+1} \tag{3.8}$$

$$e_{n+1} = x(t_{n+1}; t_0, x_0) - y_{n+1} \tag{3.9}$$

are called the **local discretization error** and **global discretization error** respectively.

There are two types of discrete-time approximations of SDEs: strong discrete-time approximations and weak discrete-time approximations. These are discussed briefly below.

3.5.1 Convergence of approximations of SDEs

Criteria to judge the accuracy of an approximation should reflect the main goal of the practical simulation. In some situations, we would want a good path-wise approximation. Another scenario, which is relevant in our case, is focused on computing expectations of functionals of the SDE[3]. These two criteria are called strong and weak convergence, and different approximations have been derived to satisfy these criteria.

We say a discretization Y^δ converges strongly with order γ at time T if there exists a C such that,

$$E[||Y^\delta(T) - X_T||) \leq Ch^\gamma \text{ [3]}$$

These are numerical methods used to approximate the solution of an SDE at each discrete time point. They are more expensive to implement, both in development and computing time as they approximate the actual paths of the SDE. Examples of strong approximation methods for SDEs include the Milstein method, the Taylor method and A-stability and implicit strong methods.

3.6 Weak convergence and Euler-Maruyama Scheme

The second scenario requires a weaker form of measuring the accuracy of the approximation called weak convergence. It essentially requires that the probability distributions of the approximation and exact solution are close. [2].

A discrete approximation Y^δ converges weakly, with order β, to X(T) (=X_T) as δ approaches 0 if, for each g in the set $C^{2(\beta+1)}$, there exists a positive constant C such that,

$$|E(g(X_T)) - E(g(Y^\delta(T))| \leq Ch^\beta \text{ [3]}$$

Weak approximations are approximations that satisfy the weak convergence criterion, the Euler-Maruyama scheme is an example of one.

The Euler-Maruyama scheme is of the form,

$$Y_{n+1} = Y_n + a\Delta_n + b\Delta W_n,$$

where

$$\Delta_n = \tau_{n+1} - \tau_n$$

and

$$\Delta W_n = W_{\tau_{n+1}} - W_{\tau_n}$$

[3]

These weak methods are often preferred for Monte Carlo simulations as they only require the evaluation of the drift and diffusion coefficients of the SDE at discrete time points. Therefore, they are simpler and faster to implement than strong approximation methods that require the solution of the SDE at each time step. This makes it much easier to repeat simulations many times, an important practical consideration. For weak convergence, we only need to approximate the measure induced by the Stochastic process X. We can, consequently, replace the Gaussian increments ΔW with other random variables with similar moment proprieties. This makes the randomness easier to simulate(pl).

The Euler-Maruyama scheme is the simplest weak approximation method and is suitable for approximating the SDE corresponding to GBM as we'll be doing in this project. We can expect weak convergence of order $\beta = 1$ as the smoothness and growth conditions that we need on the diffusion and drift coefficients will be satisfied [2].

4 Implementation

4.1 Random Number Generation

This project requires the ability to generate random samples from the normal distribution. To achieve this, we will make use of the numpy.random.default_rng().normal() function in the Numpy library of Python [6].

Random number generation in Numpy follows two stages. Firstly, a BitGenerator is used to generate sequences of random numbers. Secondly, Generators convert the created sequence into a sequence that follows a specific distribution (the normal distribution in our case). For the first stage, Numpy uses a permuted congruential generator. Ziggurat algorithms are used for the second step.

Permuted Congruential Generators (PCGs) have many properties that give them a significant advantage over other types of pseudorandom number generators. Our choice of algorithm, in particular, is very cryptographically secure. This means that it is hard (almost impossible) to determine the previous states of the Generator given its current state. While we do not directly require this property (primarily a security consideration), it leads to samples which give a more realistic mimicking of true randomness.

PCGs are also extremely fast and have extremely good statistical properties. The possession of good statistical properties means the output of these generators 'look random'. Surprisingly, many common pseudorandom number generators can fail to pass simple eye tests. More details of the improvements of PCGs over other generators can be found in O'Neill (2014) [7].

The original Ziggurat method was developed in the early 1980s but a modified version, which we will use, was introduced in 2000. This algorithm is due to Marsaglia and Tsang (2000) [4]. Ziggurat methods can be used for a number of distributions, as before, we are interested in the application to normal distributions.

The Ziggurat method is a type of rejection method. The broad aim of rejection methods for random number generation is to generate points from some (easy to sample from) set Z until they lie in a desired set C ($C \subset Z$). Z is chosen so that it satisfies the 3 following properties.

1. It is easy to generate random points from Z
2. It is easy to determine whether those points lie in C (or not)
3. area(Z) $\approx$ area(C)

Satisfying these properties means the Ziggurat method is an appropriate choice. It is slightly complex to implement, however, this complexity ensures the algorithm will, in most cases, run quickly.

To generate a single realisation from the standard normal distribution, we must first create a Generator object, before calling its standard_normal (or normal) function.

```
from numpy.random import default_rng
```

```
rng = default_rng()
Z = rng.standard_normal()
```

4.2 Using the Euler-Maruyama method

The general approach to approximating the price of one option comprises 3 stages. Firstly, we simulate trajectories of the stock price using the Euler-Maruyama method. Secondly, we can take the value of the simulated trajectory at the maturity time and calculate the implied option price. Finally, after repeating the first 2 stages a number of times, we can average the results to give an approximation to the option price (this is the Monte Carlo step).

To begin, we require a method of calculating the value of an option given the stock price at T (the maturity time). This is simple in the case of a call option.

```
np.exp(-r*T) * max(S_T - K, 0)
```

We have to evaluate the payoff $(S_T - K)_+$ and discount its value to the initial time (assumed to be 0). In the case of the binary cash-or-nothing option, we can tweak the above code to reflect the change in the payoff. The following steps are the same for all European options.

The following code implements N steps of the Euler-Maruyama method (introduced in eq. (3.1)). After the loop, S will contain a realisation of S(T).

```
dt = T / N
S = S0
for j in range(1, N+1):
    Z = rng.standard_normal()
    S = S + r*S*dt + sigma*S*np.sqrt(dt)*Z
```

Following on from this simulation we use the above code to calculate the fair price of the option (given the realised trajectory) and repeat this process many times (c.1000). Then the prices are averaged to give an approximation to the fair price at the initial time (t=0).

Let's examine the results of applying our method. We can fix S0 = 110, K = 100, σ = 0.02, r = 0.05, T = 1. For this choice of parameters, the true price of a European call option is 14.877 (5sf), according to the Black-Scholes formula.

Approximate price	Number of time-steps	Absolute error	Execution time
14.8585	100	0.0186	4.6800 s
14.8555	200	0.0216	10.4771 s
14.8608	300	0.0162	24.4518 s
14.8819	400	0.0048	39.3745 s
14.8808	500	0.0037	60.1362 s

Table 1: *Performance of Euler-Maruyama method for different time-steps*

Table 1 shows the performance of the method for certain choices of time-step width. In each run, we use a number of realizations equal to 100 times the number of time steps. For these choices of N (number of time-steps) and M (number of realizations), the Euler-Maruyama method is delivering reasonable approximations to the option price.

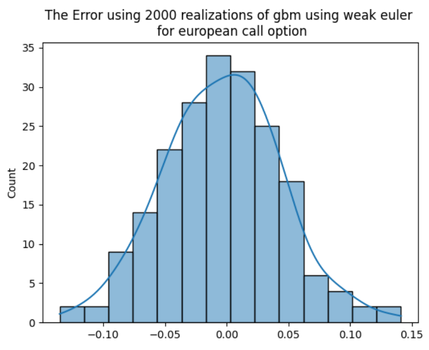
In addition, As the number of time steps, and realizations increase in Monte Carlo simulations, the computational cost of the simulation also increases. This relationship can be observed in table 1, where we see that increasing N and M results in longer execution times for the Monte Carlo simulations. The increase in computational cost can be attributed to the fact that each additional time step and realization requires additional calculations and memory storage. Therefore, when using Monte Carlo simulations, it is important to carefully balance the level of accuracy required with the available computational resources to avoid excessively long execution times. Specifically, in the next part of the task, we examined how the accuracy of the simulations changes as we increase the number of time steps and realizations, providing insights into the optimal balance between accuracy and computational cost.

Having seen that our method is working reasonably, we can now seek a more rigorous relationship between N/M and the order of convergence. We will aim to match this with the theory seen earlier.

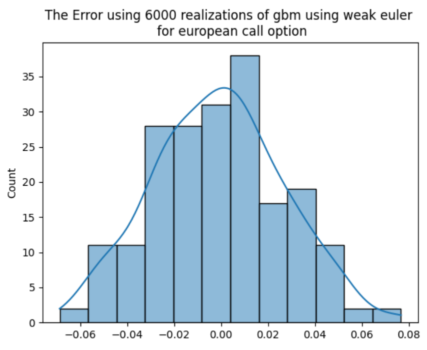
4.3 Analysing Convergence

In this section, we will scrutinize the performance of the Euler-Maruyama method. Looking at both the Monte-Carlo error and the discretization error. We will compare this with section 3.

Throughout the course of this task, we encountered various challenges related to the computational cost of the simulations. In response, we employed several strategies to address this issue. One approach was to adjust the sample size and time step, which we found to be a more feasible parameter to modify. To analyse the performance of our algorithm we can repeatedly run it, creating a sample of errors, with a sample size of 100. As well as desiring an average error close to zero, we also expect the errors (from each run of the algorithm) to be normally distributed. We can examine the cases where the number of Monte-Carlo simulations are 2000 and 6000.



(a) *N=200, M=2000*



(b) *N=600, M=6000*

Figure 1: *Distribution of errors*

Figure 1 presents 2 histograms of the error's produced on repeated runs of our algorithm (100 repetitions are presented here). Overlaid is the kernel density estimate. In the case M = 2000, we observe a very good approximation to the normal distribution. When M = 6000, the density estimate is less smooth but still mimics the shape of the normal distribution (high density close to zero, reducing to zero as error departs from 0). The variance of the errors is much lower for M = 6000, this means that computational errors will have more of an influence, potentially causing the non-smoothness in the density estimate. Overall, we don't have any evidence to contradict the assumption the errors are normally distributed.

A second approach to minimize the computational cost and, at the same time, complete the analysis is to take N to be small (8, 16, 32, 64, 128 & 256 are used) and a number of the realizations 100 times the number of time steps. The Euler-Maruyama errors are calculated on repeated runs of the algorithm, we will run our algorithm only 5 times (producing a sample size of 5) to reduce the computational cost. We can visualize in fig. 2 the error against increasing N. Moreover, we can see, in fig. 3, the mean error decreases as the realizations increase while the confidence intervals (-MC, MC) [2] shrink as the number of realizations increases.

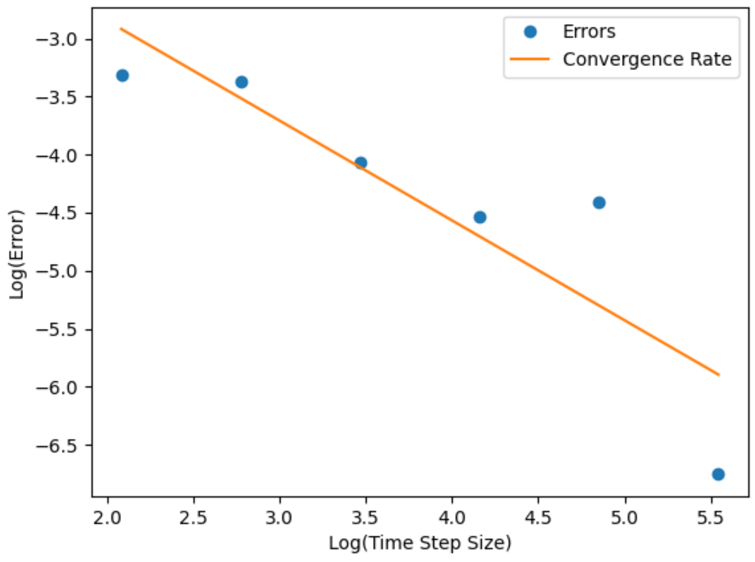


Figure 2: *Change in the error of Euler-Maruyama method as N changes*

In fig. 2, we fit a line that has a slope of 0.86. It is, therefore, reasonable to suggest, as section 3, the Euler-Maruyama error is $\mathcal{O}(h)$. We would ideally like to obtain a rate of convergence close to 1.

Figure 4 shows the error bars, and hence the Monte Carlo error, decreasing as the number of realizations increases. We can seek a more precise relationship by plotting the log of Monte-Carlo error against the log of the number realizations in fig. 4.

[2]MC represents the Monte Carlo Error

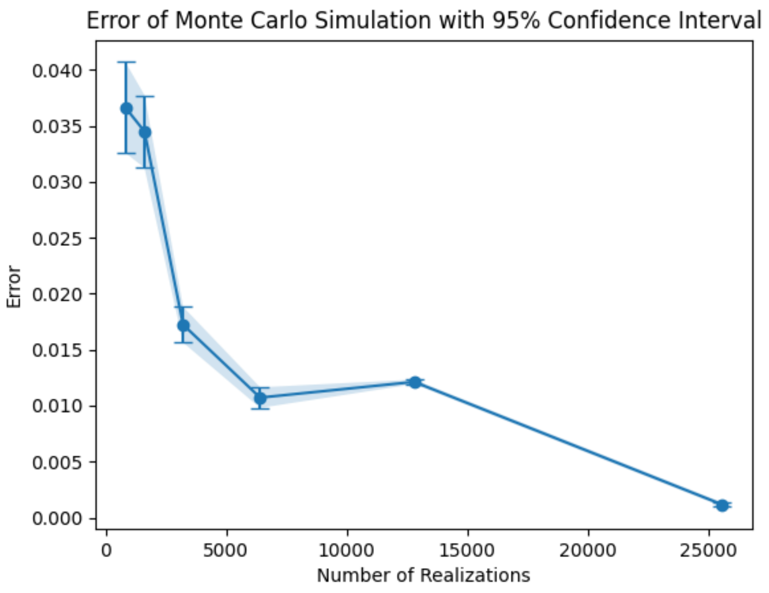


Figure 3: *Change in the error of Euler-Maruyama method with confidence interval representing Monte Carlo Error*

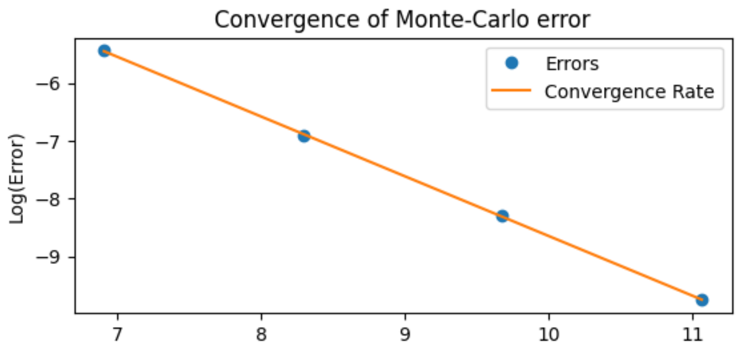


Figure 4: *Change in the error of Monte Carlo as M changes*

Realizations number	time Step	Mean error	Variance	Monte Carlo error	Execution time
800	8	0.0366	0.0034	0.0041	0.2690 s
1600	16	0.0345	0.0044	0.0032	0.6405 s
3200	32	0.0172	0.0021	0.0016	1.9377 s
6400	64	0.0107	0.0014	0.0009	7.6712 s
12800	128	0.0121	0.0001	0.0002	27.7704 s
25600	256	0.0012	0.0003	0.0001	119.2194 s

Table 2: *Performance of Euler-Maruyama method for different time-steps while increasing the number of realizations of Mote Carlo*

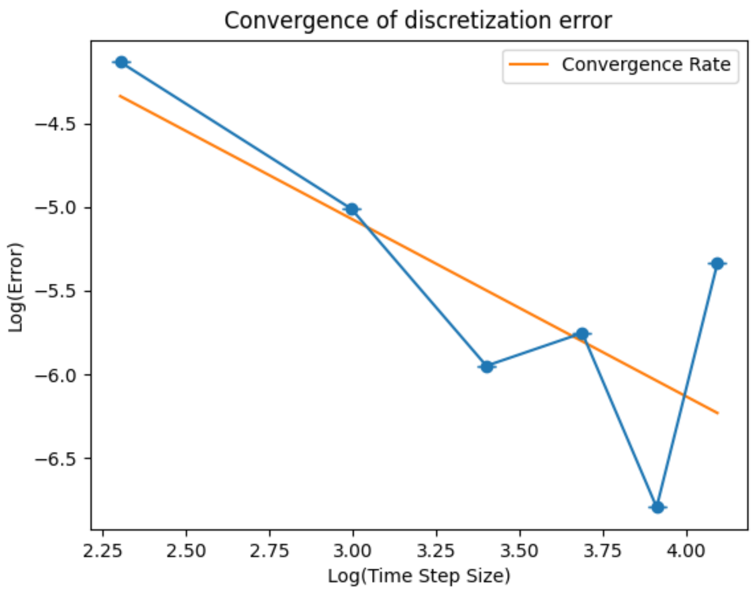


Figure 5: *Discretization error*

Table 2 provides an overview of the execution times, variance, Monte Carlo error, and mean error observed for these simulations. As the number of realizations and time steps increases, the mean error and Monte Carlo error decrease, suggesting that increasing the number of realizations and time steps can improve the accuracy of the Monte Carlo estimate. Furthermore, we can observe that the variance initially decreases with increasing realizations and time steps, indicating that the Monte Carlo estimate becomes less variable. However, the variance levels off after a certain point, suggesting that there may be diminishing returns to increasing the number of realizations and time steps beyond a certain point. In terms of computational cost, the execution time increases significantly as the number of realizations and time steps increases, reflecting the increased computational complexity of the simulation. Therefore, there is a trade-off between computational cost and accuracy, and one needs to choose a balance between the two based on their specific needs. In addition, we can observe that the Monte Carlo error decreases as the number of realizations and time steps increases, and the ratio between the Monte Carlo error and the square root of the number of realizations seems to approach a constant value. This suggests that the Monte Carlo error follows an $\mathcal{O}\left(\sqrt{h}\right)$ convergence rate, as expected.

While our results indicate a reasonable rate of convergence of 0.86 for the Euler-Maruyama method given our limited computational resources, we explored a different approach in an attempt to improve the rate of convergence and bring it closer to 1. We can create a similar plot to study the relationship between N and the discretization error. To perform this analysis we need to fix a high value of M so that the Monte-Carlo error is sufficiently small ($< 10^{-4}$) and not having a significant effect on the absolute error. We can then change the value of N (use 10, 20, 30, 40, 50, 60, and 5 sample sizes (to prevent the computations from taking excessive time).

Figure 5 shows a linear relationship and the line of best fit has a decreasing slope of magnitude $1.0560 \approx 1$. This also suggests that the discretization error is O(h). We do note a number of observations (particularly for larger N) that depart significantly from the line. This is of concern and needs investigating further.

In addition, despite using small sample sizes and time steps, our results revealed that simulations in these circumstances incur a significant computational cost. The following table provides an overview of the execution times observed for these simulations, variance, Monto Carlo error, and Mean error.

Realizations	time Steps	Mean error	Variance	Monte Carlo error	Execution time
500000	10	0.0160	5.7072e-06	6.6219e-06	143.2878 s
500000	20	0.0067	1.2813e-05	9.9221e-06	216.5884 s
500000	30	0.0026	1.0830e-05	9.1218e-06	274.5995 s
500000	40	0.0032	1.6940e-05	1.1409e-05	386.3310 s
500000	50	0.0011	2.0621e-06	3.9804e-06	483.3666 s
500000	60	0.0048	1.3904e-05	1.0336e-05	667.0695 s

Table 3: *Performance of Euler-Maruyama method for different time-steps while fixing the number of realizations*

The results in table 3 indicate that as the number of time steps increases, the mean error decreases. However, there is a trade-off between accuracy and computational cost, as evidenced by the increase in execution time for simulations with more time steps. Additionally, the Monte Carlo error also increases as the number of time steps increases, indicating that the accuracy of the simulation is affected not only by the mean error but also by the variance of the estimate. These findings highlight the importance of carefully selecting the appropriate number of time steps to balance accuracy and computational cost in Monte Carlo simulations.

4.4 Cash-or-nothing option

We can repeat this analysis for a European Binary cash-or-nothing call option. This type of option has the payoff f(x) (defined below).

$$f(x) = \begin{cases} 1, & \text{if } x > K \\ 0, & \text{otherwise} \end{cases}$$

To approximate the price for this type of option, we can use the Euler-Maruyama method as before (section 4.2) to simulate values for S(T). Then we calculate the option price on each path, $e^{-rT} f(S(T))$ (where f is as above). Finally, we apply the Monte-Carlo method (as in section 4.2) to average the results and deliver an approximation to the fair option price. Let's examine the results of applying our method. We can fix S0 = 100, K = 100, σ = 0.2, r = 0.05, T = 1. For this choice of parameters, the true price of a European binary call option is 0.5323 (5sf). Approximating this price by our code we got the following results.

Table 4 shows the performance of the Euler-Maruyama method when pricing a binary option for different numbers of time steps, with the number of realizations fixed at 100 times the number of time steps. The results reveal that as the number of time steps increases, the absolute error decreases, indicating improved accuracy in the approximation of the

Approximate price	Number of time-steps	Absolute error	Execution time
0.5244	8	0.0080	0.0224 s
0.5291	16	0.0032	0.0965 s
0.5336	32	0.0013	0.7151 s
0.5285	64	0.0038	1.1243 s
0.5251	128	0.0072	4.1444 s
0.5278	256	0.0045	16.5985 s

Table 4: *Performance of Euler-Maruyama method for different time-steps for binary option*

option price. However, the trade-off for improved accuracy is an increase in execution time, which also increases as the number of time steps increases. The execution times are still relatively small, however, if we are required to repeatedly run the algorithm (e.g. in a practical scenario where a range of options needs to be considered), we will suffer much more. Notably, the computational cost increases exponentially with the number of time steps, which can make simulations impractical for very large numbers of time steps. By increasing the number of realizations inversely proportional to the number of time steps, we can preserve the accuracy of the simulations without sacrificing too much computational efficiency.

Now, we continue the analysis of this method by examining the Euler error. We take the same numbers of N and sample size used when analysing the plain vanilla European option. We can visualize in fig. 6 the error decreases over increasing N with the rate of convergence of 0.6. Moreover, we can see in fig. 6 how the mean error behaves in a strange way as the realizations increase while the confidence intervals (-MC, MC) shrink as the realizations increase.

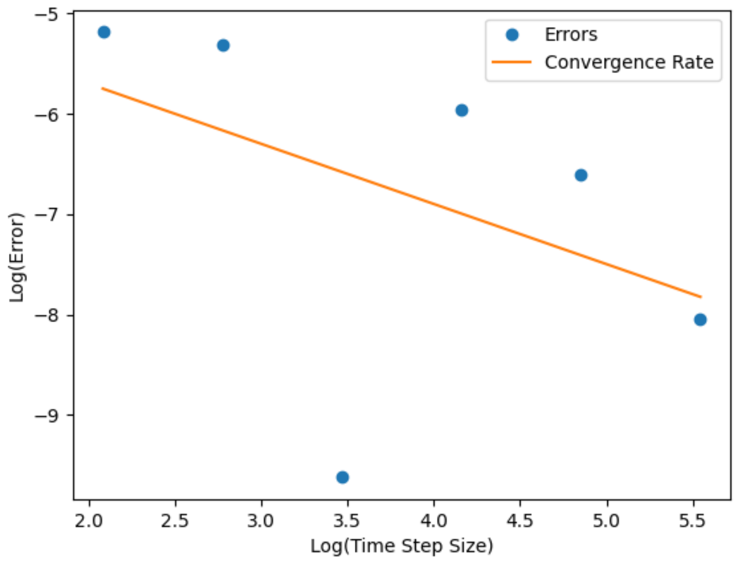


Figure 6: *Plot of the error of Euler-Maruyama against a number of time step*

We summarize the results in table 5

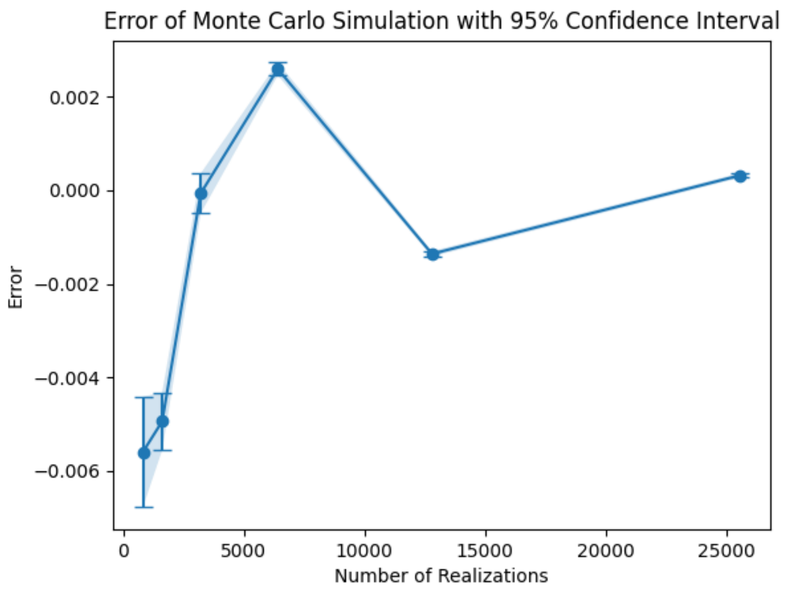


Figure 7: *Change in the error of Euler-Maruyama method with representing Monte Carlo Error as a confident interval*

Realizations	Time Steps	Mean error	Variance	Monte Carlo error	Execution time
800	8	-0.0056	0.0003	0.0012	0.2021 s
1600	16	-0.0049	0.0002	0.0006	1.0283 s
3200	32	-0.0001	0.0002	0.0004	2.9485 s
6400	64	0.0026	0.0000	0.0001	10.3385 s
12800	128	-0.0014	0.0000	0.0001	47.4120 s
25600	256	0.0003	0.0000	0.0001	180.2116 s

Table 5: *Performance of Monte Carlo simulation with increasing realizations and time steps*

In table 5, the Monte Carlo error decreases as the number of realizations increases and as the size of time steps decreases. Also, the variance initially decreases with increasing realizations and time steps. However, the variance levels off after a certain point, suggesting that there may be diminishing returns to increasing the number of realizations and time steps beyond a certain point. This may be due to the nature of the discontinuity of the payoff function.

Furthermore, we noticed a strange sudden change. When considering the values for N: 10, 20, 30 and 40; M = 500000 (sufficiently high to eliminate the Monte Carlo error). We can plot the log of the absolute error (which will be dominated by the discretization error) against the log of N in fig. 8

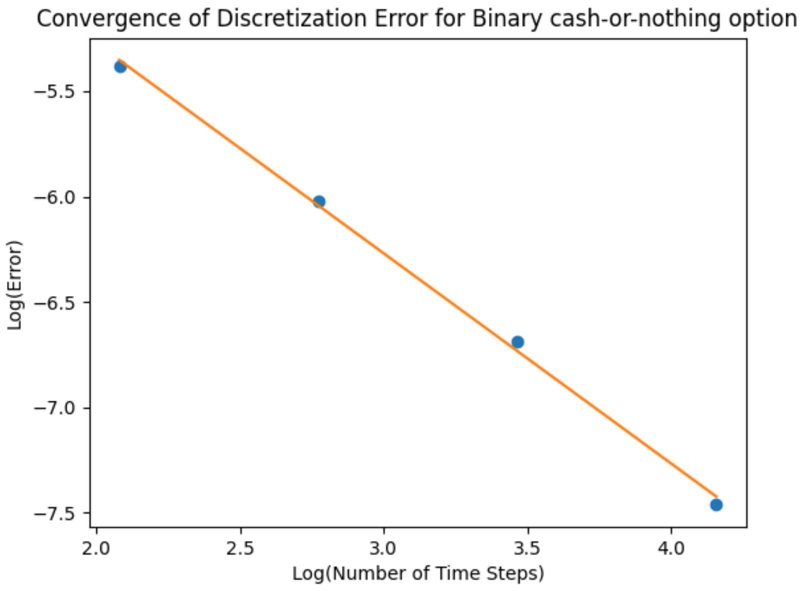


Figure 8: *Convergence of discretization error*

Figure 8 shows the observed error of the Monte Carlo method when approximating the price of a Binary cash-or-nothing option. The observed convergence rate for this simulation is 0.99578, which is close to 1.

Obtaining these results was not a trivial task 'as we thought', as the computational cost was not the only issue that needed to be addressed. In contrast to approximating the price of an ordinary European option, a peculiar behavior in the convergence of the error emerged when varying the number of time steps and the maturity time need more investigation.

As an alternative strategy, we explored the exponential solution of the stochastic differential equation of the geometric Brownian motion. This approach circumvented the need to compute the discretization error and proved to be more manageable, as it ensured the preservation of the positivity of the price, an assumption in our model. This assumption won't be violated if S0 is sufficiently high and sigma sufficiently small. However, to ensure the robustness of the method, it is prudent to consider the exponential solution.

We replaced this code

```python
dt = T / N
S = S0
for j in range(1, N+1):
    Z = rng.standard_normal()
    S = S + r*S*dt + sigma*S*np.sqrt(dt)*Z
```

by just this

```python
Z = rng.standard_normal()
S = S0*math.exp((r-(1/2)*sigma**2)*T+(sigma*np.sqrt(T)*Z))
```

We examined the code approximating the price of a European binary call option with the same choices of parameters. As a result, we get the following table

Approximate price	Number of Realisation	Absolute error	Execution time
0.5305	10,000	0.0018	0.0547 s
0.5307	100,000	0.0017	0.4851 s
0.5316	1,000,000	0.0008	5.0324 s
0.5321	10,000,000	0.0002	53.8476 s

Table 6: *Performance of exponential solution with Monte Carlo for binary option*

As we can see from table 6 and the previous table, the exponential solution with Monte Carlo method has a significantly lower absolute error than the Euler-Maruyama method, especially for larger values of M or N. Additionally, the execution times for the exponential solution are also generally lower than those for the Euler-Maruyama method. Therefore, the exponential solution with Monte Carlo method appears to be a better choice for approximating the price of binary options. For studying the convergence of Monte Carlo we have the following table showing the

Num. of realizations	Mean error	Variance	Monte Carlo error	Execution time
10^4	0.000939522	0.000014731	0.000075226	0.7586 s
10^5	-0.000231441	0.000002288	0.000009376	5.6068 s
10^6	0.000096638	0.000000283	0.000001042	61.6388 s
10^7	-0.000031341	0.000000015	0.000000077	566.6564 s

Table 7: *Performance of exponential solution with Monte Carlo for binary option*

Table 7 shows the performance of the Monte Carlo simulation method for pricing binary options. As the number of realizations increases, the mean error decreases, the variance decreases, and the Monte Carlo error decreases. However, the execution time also increases significantly. Overall, the method is effective in pricing binary options, but 'again' the trade-off between accuracy and computational cost needs to be carefully considered.

Plotting the log of the error against the log of realizations in fig. 9, and calculating the slope indicates a rate of convergence approximately equal to 0.48. This suggests that the Monte Carlo error follows an $\mathcal{O}\left(\sqrt{h}\right)$ convergence rate, as expected.

In the preceding two methods, it is noteworthy that the former approach entailed setting the realizations to a large value, thereby allowing us to disregard the Monte Carlo error and instead focus on examining the convergence of the discretization error. In the

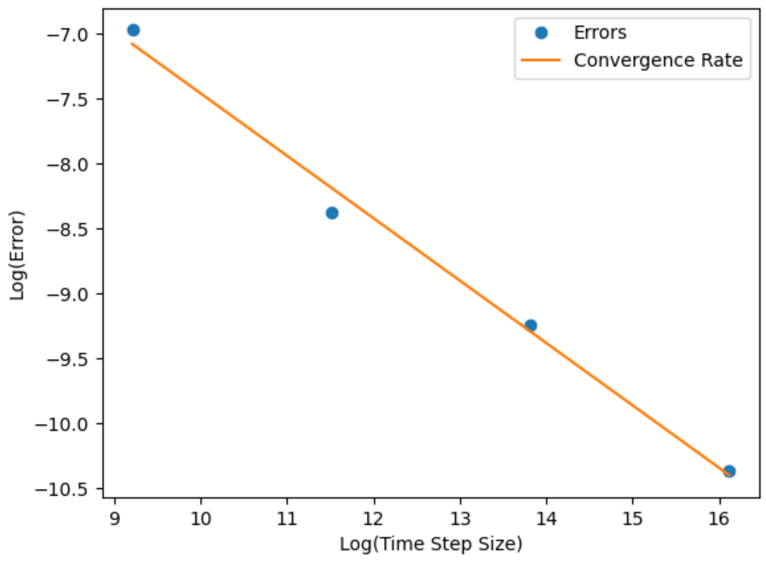


Figure 9: *Convergence of Monte Carlo error using the exponential solution of SDE*

latter approach, we obtained the exponential solution of the SDE, enabling us to solely consider Monte-Carlo convergence (there will be no discretization error). When using either technique to analyse both discretization error and Monte-Carlo error, we observed peculiarities. Considering the natural discontinuity of binary European option payoff, unexpected increasing and decreasing behavior as realizations increase can be explained. When using different approaches to study the 2 errors we can obtain more well-behaved convergence rates. Despite the possibility of utilizing the exponential solution of SDE to determine the price of regular European options and deriving additional insights, the limitations of time and pages preclude us from pursuing this avenue.

5 Speeding up the Monte Carlo method

5.1 Variance reduction

The error of the Monte Carlo method is normally distributed around 0 with standard error $\frac{\sigma}{\sqrt{n}}$. Moreover, as n increases, the error approaches 0 with a variance of 0. Therefore, it is said that the Monte Carlo produces a consistent estimator of the integral of f. However, the convergence of the Monte Carlo method is rather slow in practice. For instance, reducing the standard error by a factor of 10 requires increasing n by 100.

Variance reduction methods are used to increase the Monte Carlo's convergence by reducing the variance σ. The way variance reduction is achieved depends widely on the method chosen. The Control Variates method achieves variance reduction by introducing a new estimator which depends on the variable of interest and a control variable with known expectation. Antithetic Variates introduce extra samples drawn from a random variable with the same distribution as the random variable of interest but with negative covariance. Stratified Sampling tries to improve the sampling used by Monte Carlo by drawing samples from equally probable strata. Finally, Importance Sampling changes measures, e.g. the risk-neutral measure, so that certain outcomes have higher probabilities.

The results obtained by these methods depend highly on the type of problem being solved. However, Control Variates and Antithetic Variates are general methods in the sense that they can be easily implemented in any type of problem whether it lies in financial engineering or not.

5.2 Control Variates

Let $\{Y_i\}_{i=1}^{n}$ be the set of independent identically distributed realizations generated by our simulation where Y_i could be a payoff function, then the sample mean estimator $\bar{Y}$ would be an unbiased consistent estimator of Y. Now, let $\{X_i\}_{i=1}^{n}$ be another set of dependent identically distributed outcomes but this time $E[X_i]$ is known. Then, for a fixed b, the following quantity can be calculated for each of the i-th replications:

$$Y_i(b) = Y_i - b(X_i - E[X]) \tag{5.1}$$

Then by computing the sample mean of 5.1, the control variate estimator is obtained:

$$\bar{Y}(b) = \bar{Y} - b(\bar{X} - E[X]) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - b(X_i - E[X]) \tag{5.2}$$

The control variate estimator $\bar{Y}(b)$ is distributed normally by the central limit theorem. The expectation and the variance of 5.2 is computed to determine its distribution:

$$\begin{aligned} E[\bar{Y}(b)] &= E[\bar{Y} - b(\bar{X} - E[X])] \\ &= E[\bar{Y}] - b(E[\bar{X}] - E[X]) = E[Y] \end{aligned}$$

$$
\begin{aligned}
\mathrm{Var}[\bar{Y}(b)] &= \mathrm{Var}[\bar{Y} - b(\bar{X} - E[X])] \\
&= \mathrm{Var}[\bar{Y}] - 2b\mathrm{Cov}(\bar{X}, \bar{Y}) + b^2\mathrm{Var}[\bar{X}] \\
&= \frac{1}{n}(\sigma_Y^2 - 2b\sigma_X\sigma_Y\rho_{X,Y} + b^2\sigma_X^2)
\end{aligned}
$$

As n goes to infinity, $E[\bar{Y}(b)] \to E[Y]$ and $\mathrm{Var}[\bar{Y}(b)] \to 0$, therefore 5.2 is a consisted estimator of Y. Moreover, if $b^2\sigma_X < 2b\sigma_Y\rho_{X,Y}$, then $\bar{Y}(b)$ will have less variance than $\bar{Y}$.

Now, the value of b that minimizes the variance in 5.2 is found:

$$
b^* = \frac{\sigma_Y}{\sigma_X}\rho_{X,Y} \tag{5.3}
$$

Finally, the ratio of the variance of the optimally controlled estimator $\bar{Y}(b*)$ to the sample mean $\bar{Y}$ allows us to determine how correlated Y and X must be so that the variance decrease:

$$
\frac{\mathrm{Var}[\bar{Y} - b^*(\bar{(X)} - E[X]}{\mathrm{Var}[\bar{Y}]} = 1 - \rho_{X,Y}^2 \tag{5.4}
$$

The expression above tells us that as more correlated (independently of its sign) are X and Y, the variance reduction will be maximized.

5.3 Antithetic Variates

The method of Antithetic Variates proposes to reduce variance by drawing i.i.d samples $\{(Y_i, \tilde{Y_i})\}_{i=1}^n$ such that $(Y_i, \tilde{Y_i})$ forms an antithetic pair. An antithetic pair is pair of a random variables with the same distribution but with negative covariance. For example, the uniform random variables U and $1 - U$ in $[0, 1]$ form a pair of antithetic variables. The estimator of Y defined by this method is known as the antithetic estimator and has the form:

$$
\hat{Y}_{\mathrm{AV}} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i + \tilde{Y_i}}{2}\right) \tag{5.5}
$$

By the central limit theorem,

$$
\frac{\hat{Y}_{\mathrm{AV}} - E[Y]}{\sigma_{\mathrm{AV}/\sqrt{n}}} \to N(0, 1) \tag{5.6}
$$

Where the variance of the estimator is:

$$\sigma^2_{\text{AV}} = \text{Var}\left[\frac{Y_i + \tilde{Y}_i}{2}\right] = \text{Var}[Y_i] + \text{Cov}(Y_i, \tilde{Y}_i) \tag{5.7}$$

As $\text{Cov}(Y_i, \tilde{Y}_i) < 0$ always because Y_i and Y_i form an antithetic pair, the antithetic estimator will have less variance than the sample mean estimator of Y.

5.4 Implementation

In this section, we explore the implementation of the Control Variates method for approximating the price of Vanilla European and Binary cash-or-nothing European options.

The main reason for choosing Control Variates is that the method has a general implementation because it only requires using a variable X such that $E(X)$ is known. Additionally, to guarantee variance reduction, X and Y should be highly correlated. When working with European options, which only depend on the price at time T, X can be chosen to be $S(T)$. Moreover, the discounted price at time T, $e^{-rT}S(T)$, is a martingale with respect to $S(0)$, therefore, $E[S(T)] = e^{rT}S(0)$.

Additionally, the coefficient b^* needs to be computed using equation eq. (5.3). However, we cannot compute b^* because the population parameters for Y are unknown in practice. Therefore, we estimate b^* by using sample parameters:

$$\hat{b} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2} \tag{5.8}$$

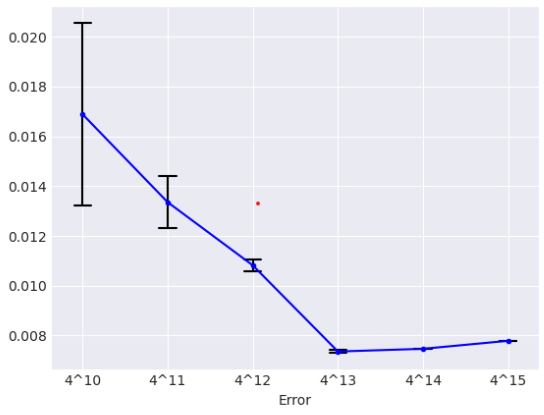
In the following figure, we list the code for computing the control variate estimator using the equation 5.2. The function accepts the option (a vanilla European or binary cash-or-nothing option) and n samples of $S(T)$. In line 7, $\hat{b}$ is computed using the sample parameters $\bar{X}$ and $\bar{Y}$ obtained in lines 4 and 5.

```
1 def control_variates(option: Option, ST: NDArray[np.float64]):
2     X = ST
3     Y = option.discounted_payoff(X)
4     EX = option.forward_stock_price()
5     Y_bar = np.mean(Y, axis=0)
6     X_bar = np.mean(X, axis=0)
7     b_hat = np.sum((X-X_bar)*(Y-Y_bar), axis=0)
8     b_hat /= np.sum(np.power((X-X_bar),2), axis=0)
9     return np.mean(Y - b_hat*(X - EX), axis=0)
```

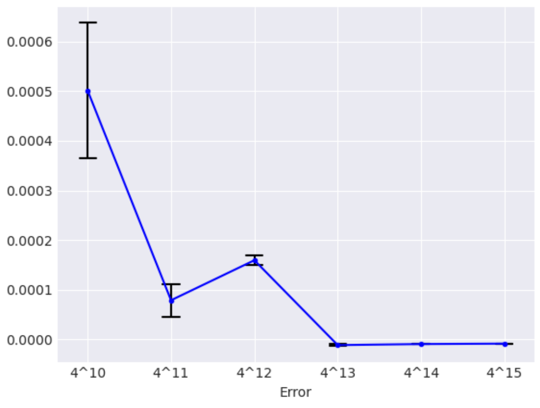
To compare the method's effectiveness, we use Monte Carlo and Control Variate methods to approximate the option price of a vanilla and a binary cash-or-nothing European call option using $4^5, 4^6, \ldots, 4^{10}$ realizations. For each realization, 100 samples are produced to compute the mean and the standard error. Table 8 lists the standard error produced by both methods for the vanilla European call option.

Realizations	Monte carlo S.E	Control variates S.E
4^5	5.9568e-2	1.9640e-3
4^6	3.4197e-2	9.8915e-4
4^7	1.5017e-2	4.9506e-4
4^8	8.5642e-3	2.2296e-4
4^9	3.8868e-3	9.8953e-5
4^{10}	1.7343e-3	5.8836e-5

Table 8: *Variance produced by Monte Carlo and Control Variates methods for vanilla European call option with strike price K=100 and T=1. Price is simulated using a Geometric Brownian Motion with S(0)=100, r=0.05, and σ=0.02. σ.*



(a) *Monte carlo error with 95% C.I*



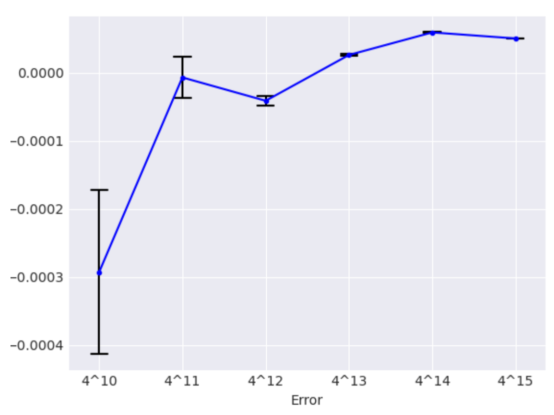
(b) *Control Variates error with 95% C.I*

Figure 10: *Vanilla European call option. The plots illustrate how effectively the Monte Carlo with control variates reduces the standard error when applied to a vanilla call option.*

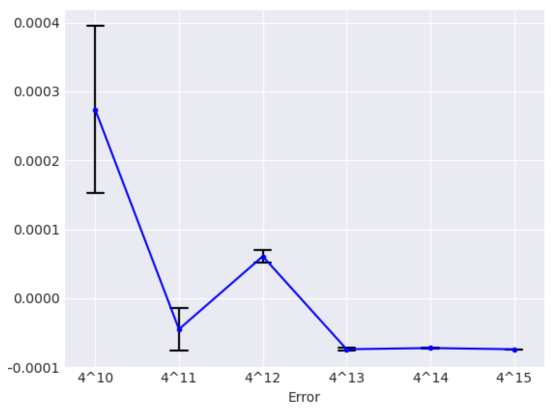
As can be observed, the standard error produced by control variates is 10 times less than Monte Carlo's standard error. For the binary cash-or-nothing European call option, a less drastic effect on standard error can be observed in Table 9.

Realizations	Monte Carlo S.E	Control Variates S.E
4^5	2.2277e-3	1.9700e-3
4^6	1.0770e-3	1.0278e-3
4^7	6.7339e-4	6.2727e-4
4^8	2.8089e-4	2.7648e-4
4^9	1.4924e-4	1.3789e-4
4^{10}	6.7781e-5	6.3396e-5

Table 9: *Variance produced by Monte Carlo and Control Variate methods for a binary cash-or-nothing European call option with strike price K=100 and T=1. Price is simulated using a Geometric Brownian Motion with S(0)=100, r=0.05, and σ=0.02.*



(a) *Monte carlo error with 95% C.I*



(b) *Control Variates error with 95% C.I*

Figure 11: *Binary cash-or-nothing European call option.*

6 Conclusions

The application of the Euler-Maruyama method and the Monte Carlo simulation technique for the numerical solution of stochastic differential equations requires significant computational resources, particularly when increasing the number of realizations and time steps. As a result, the computational time for obtaining accurate results can be so long, which may limit the practicality of the method for certain applications. In the context of the European Binary cash-or-nothing call option, the performance of the numerical approximation technique of Euler-Maruyama exhibits peculiar behaviour. Specifically, the error function displays sharp fluctuations as the time to maturity T is approached. This phenomenon can be attributed to the discontinuous nature of the underlying payoff function. This behaviour needs more investigation. If the analytical solution of a given stochastic differential equation is known, the Monte Carlo method can be employed with higher efficiency. Specifically, as the number of realizations approaches infinity, the Monte Carlo error will tend to zero which would be without worrying about the specific numerical simulation method employed.

Control Variates achieved a considerable variance reduction for the vanilla European option. This is due to the high correlation between the final price and the payoff function. However, the correlation seems to depend on the strike price of the option. Therefore, different values of K would yield different proportions in variance reduction. In a real scenario, it would be more suitable to have a method less dependent on the strike price. For the Binary cash-or-nothing call European option, a less drastic effect could be observed in variance. We could explain this by saying that the Monte Carlo error and standard error are decently low for this type of option therefore we would not see a drastic variance reduction for this type of option with any of the methods available. A second interpretation is that the variance and the standard error are already low with for Monte Carlo method because the option's discounted payoff will always result in values between 0 and 1.

A User Guide

The code for this project is written in Python and as such available in .py files. Python is a high-level language (highly interpretable) that does not need compiling before execution [8]. There are a vast array of programs capable of running Python code (such as the Python IDLE [8]), we recommend Microsoft's Visual Studio Code [10]. Visual Studio Code (available here [10]) can be downloaded for Windows or Mac systems. It is capable of working with many program languages, including Python.

To run programs in Visual Studio Code, simply open the file containing the source code and select run from the dropdown menu. Please note, **task4c.py** can take around an hour to run. We have run all programs in Python 3.10.7, in theory, they should run in any version of Python 3.

Our submission includes 5 programs, their uses are summarised in below.

- **task4a.py** defines 2 functions calculate_european_call_option_price & Applying_Monte_Carlo. These are used to implement the Euler-Maruyama method in **task4b.py** & **task4c.py**.

- **task4b.py** also defines Black_Scholes_formula to calculate the true price of a European call option. This program is capable of applying the Euler-Maruyama method.

- **task4c.py** uses the previous 2 programs to analyse the Euler-Maruyama method and study convergence.

- **task4d.py** follows similar procedures to the previous 3 programs to study convergence for a binary call option. It defines functions Black_Scholes_formula_for_binary_call_option, calculate_european_binary_call_option & Applying_Monte_Carlo

- **task4d_exp.py** follows similar procedures to the previous task4d programs but just changes Monte-Carlo to use the exponential solution of the SDE, to study convergence for a binary call option. It defines functions Black_Scholes_formula_for_binary_call_option, calculate_european_binary_call_option & Applying_Monte_Carlo

- **task5.py** compares methods for speeding up Monte Carlo convergence. It defines functions GBM, monte_carlo_solver & control_variates.

Library used in our programs are: numpy, math, scipy, time, matplotlib, seaborn, abc. Some of these will need to be downloaded if not already present on a users device.

The parameters controlling a call option are specified in the source code. The user is welcome to alter these by altering the files. All parameters are named conventionally (e.g. T for maturity time) and comments also indicate their meaning. In the implementation, we do assume t (initial time) = 0.

The user is also able to produce custom outputs by calling our functions with desired parameter values.

```
Black_Scholes_formula(S0=120, K=150, sigma=0.04, r=0.02, T=5)
```

The code snippet above will return (after **task4b.py** has been run) the value of a European call option with the parameters as provided.

References

[1] Platen, Eckhard. “An Introduction to Numerical Methods for Stochastic Differential Equations.” Acta numerica 8 (1999): 197–246. Web.

[2] Glasserman, Paul. *Monte Carlo Methods in Financial Engineering* . New York: Springer, 2003. Print.

[3] Kloeden, Peter E., and Eckhard. Platen. *Numerical Solution of Stochastic Differential Equations* . Berlin: Springer, 1992. Print.

[4] Marsaglia, George, and Wai Wan Tsang. “The Ziggurat Method for Generating Random Variables.” Journal of statistical software 5.8 (2000): 1–7. Web.

[5] Milstein, G. N., and Michael V. Tretyakov. *Stochastic Numerics for Mathematical Physics* . Berlin; London: Springer, 2004. Print.

[6] https://numpy.org/doc/stable/reference/random/index.html Retrieved 15th March 2023

[7] O’neill, Melissa E. ”PCG: A family of simple fast space-efficient statistically good algorithms for random number generation.” ACM Transactions on Mathematical Software (2014).

[8] https://www.python.org/ Retrieved 26th March 2023.

[9] Thomas, David et al. “Gaussian Random Number Generators.” ACM computing surveys 39.4 (2007): 11–es. Web.

[10] https://visualstudio.microsoft.com/ Retrieved 26th March 2023.