UNIVERSITÄT KOBLENZ · LANDAU

PROJECT SEMINAR REPORT

PRICING BASKET OPTIONS IN THE BLACK SCHOLES SETTING VIA MONTE CARLO METHOD

GUIDED BY: DR. SEMA COSKUN

DATE: 08 SEPTEMBER 2021

SUMMER SEMESTER 2021

GROUP MEMBERS:

1. KISHANKUMAR MATHUKIYA: 220200695
2. SANDIP MANI JAYARAM: 221100444
3. SANDIP KHUNT: 221100687

Declaration

‘We hereby declare in lieu of oath that we wrote this Document by ourselves and did not use any sources other than those mentioned. All the figures are also made by us. The purpose of this report is to complete the laboratory experiment only.’

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGEMENT

First and foremost, praise and thanks to God, the Almighty, for her showers of blessings throughout my research work to complete the research successfully.

We would like to express our deep and sincere gratitude to our Project Seminar guide, Dr. Sema Coskun for giving us the opportunity to do research and providing invaluable guidance throughout this project. Her dynamism, vision, sincerity, and motivation have deeply inspired us. She taught us the methodology to carry out the research and to present the research works as clearly as possible. It was a great privilege and honor to work and study under her guidance. We are extremely grateful for what she has offered us. We would also like to thank her for her friendship, empathy, and great sense of humor. We are extending our heartfelt thanks for her acceptance and patience during the discussion we had during research work and report preparation.

Abstract

The basic principle behind the option pricing by using the Monte Carlo method is to compute the expected value of a quantity which is a function of the solution of a stochastic differential equation. Hence, we consider the payoff function of the underlying stock price which follows the geometric Brownian motion in the Black-Scholes model. To compute the expected payoff we simulate a large number of stock price paths and for each path we compute the expected payoff. As suggested by the Monte Carlo method, we take the arithmetic average of all possible payoffs of the relevant option. Finally, we discount this expected payoff value in order to obtain the price of an option today. In this report, we initially introduce the basic principles of the Monte Carlo method and later we present its application for option pricing.

1. Introduction

This project will focus on Monte Carlo simulations to price basket options in the Black Scholes setting. Monte Carlo simulation is a widely used tool within finance for computing the price of financial instruments. Monte Carlo methods are ubiquitous in applications in the finance and insurance industry. They are often the only accessible tool for financial engineers and actuaries when it comes to complicated price or risk computations [1].

Option is a type of contract between two parties that provides one party the right (but not the obligation) to buy or sell the underlying asset at a pre-determined price before or at the expiration day [2]. Option are classified in the following way:

Call option: Give holder the right but not obligation to buy the underlying asset at a pre-determined price for a specific time period.

Put option: Give holder the right but not obligation to sell the underlying asset at a pre-determined price for a specific time period.

Options are also classified into a number of styles and the most common are:

American Option: An option that may be exercised on any trading day on or before expiration.

European Option: An option that may only be exercised on expiry.

Now, Option pricing models estimate a value of an options contract by assigning a price also known as premium, based on calculated probability that the contract will finish in the money at expiration. So it gives option’s fair value which traders incorporate into their strategies. Commonly used models to value options are,

- Binominal model
- Black Scholes model
- Monte Carlo simulation

In this seminar, we have used the Black Scholes model to find a fair value of Option using Monte Carlo Simulation initially for the European options and subsequently for the basket options.

In 1973, two professors Fischer Black and Myron Scholes developed a model that could be used to calculate the price of an option. This came to be known as the Black-Scholes Option Pricing Model otherwise known as the Continuous Option Pricing Model. Now, traders had a way of understanding the option pricing mechanism and take advantage of any under-pricing or overpricing in the market. There are many different types of options that can be traded, but in a very broad sense, there are two main types: The European style option and the American style option [3].

2. The Monte Carlo Method

The main idea of the Monte Carlo method is to approximate an expected value $\mathbb{E}(X)$ by an arithmetic average of the results of a big number of independent experiments which all have the same distribution as X [1]. So it relies on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches. Monte Carlo methods are mainly used in three problem classes: optimization, numerical integration, and generating draws from a probability distribution [3].

2.1. Monte Carlo Method: Basic Principle

As discussed in 2, the basic principle of the Monte Carlo method is to take the arithmetic mean of large number of random variables in order to obtain expected value $\mathbb{E}(X)$. The first step for Monte Carlo estimation is to generate n samples that all are independent random variables $X_i(\omega)$ for $i = 1,2, \dots, n$. Where. $n \in N$. And second step is to approximate $\mathbb{E}(X)$ by taking the arithmetic mean,

$$\mathbb{E}(X) \coloneqq \frac{1}{n} \sum_{i=1}^{n} X_i(\omega)$$

The method in this pure form is called the crude Monte Carlo method. As the Monte Carlo estimator is a random variable, each run of it typically produces new values. As the estimator is unbiased, the variance of the estimator is a measure for its accuracy [1]. The basis of this method is strong law of large numbers which is a very important concept in probability theory.

2.2. The strong law of large numbers

The strong law of large numbers is one of the powerful theorems of probability theory and it states that the arithmetic mean of a sequence of independent, identically distributed random variables (X_n) converges almost surely to the expected value $\mu \coloneqq \mathbb{E}(X)$ [1].

2.3 Increasing the accuracy of the Monte Carlo method

By increasing the numbers of random variables, accuracy can be increased and we will come closer to the expected value. But to Increase the (mean) accuracy of the crude Monte Carlo estimate by one digit (i.e. reducing its standard deviation by a factor 0.1) requires increasing the number of Monte Carlo runs by a factor of 100 [1]. So it requires significant effort to achieve higher accuracy.

2.4 Central limit theorem

The standard deviation is a measure for the accuracy of the Monte Carlo method and it's of order $(1/\sqrt{N})$. The use of standard deviation can be justified by central limit theorem. Central limit theorem states that the average sample mean will be the same as the actual population mean. Same as if you find the average of all standard deviation in sample, we can find the actual standard deviation for population as well.

2.5 Variance reduction methods

The Drawback of the Monte Carlo method is slow convergence. The standard deviation of the error decreases in proportion to square root of number of simulations. So if when decrease the variance, we can increase the speed of Monte Carlo in lower simulations run and increase accuracy. Such kind of decreasing the variance in the Monte Carlo method is called variation reduction methods. There are some variation reduction methods as below.

- Antithetic variates
- Control variates
- Stratified sampling
- Importance sampling
- Variance reduction by conditional sampling

2.6 Application of Monte Carlo method

Monte Carlo methods are a broad class of computational algorithms that are independent random sampling to obtain numerical results. It is necessary that the equation understudy can be related to an expectation which is then itself approximated by the arithmetic mean of that random numbers. A Monte Carlo simulation helps one visualize most or all of the potential outcomes to have a better idea regarding the risk of a decision. The Monte Carlo method is an extremely powerful computational technique in any mechanical engineer's toolkit. There are a number of common use cases in engineering to which Monte Carlo simulation naturally applies, for example, statistical tolerance analysis, Reliability design, Design optimization, etc. This technique can be used in different domains like Economics Specially risk management and Extensively use in financial institutions to compute price of financial derivatives such as options which is the aim of this report.

One of the basic examples of the Monte Carlo algorithm is the estimation of π.

2.6.1. Example I: Estimating the value of π

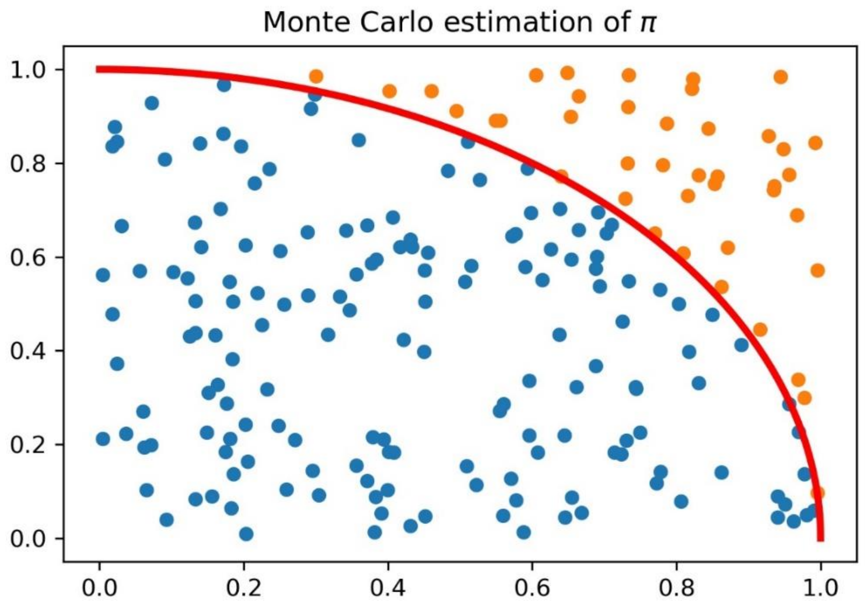


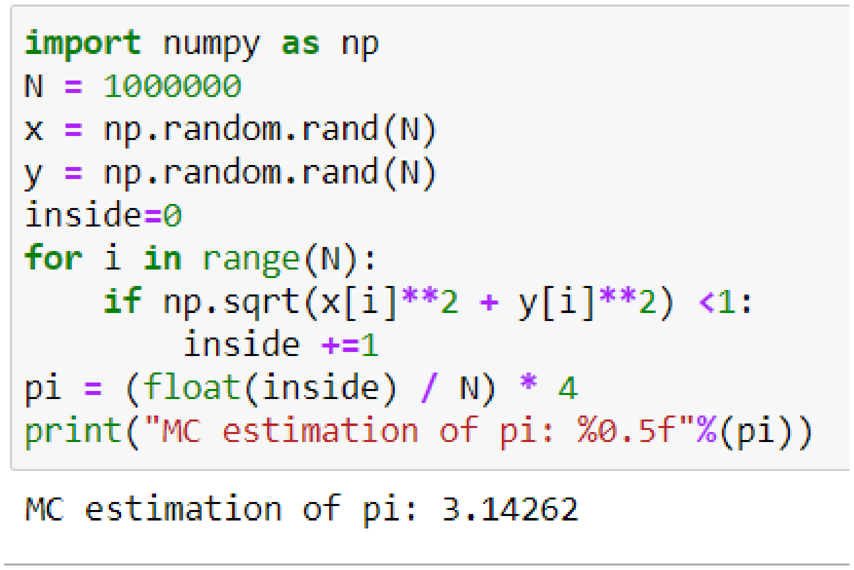
Figure 2-1 Random point generation

For estimation of π with help of Monte Carlo method, consider the circle, enclosed by a square such that diameter of a circle is equal to the side of the square. Now consider large number of uniformly distributed points on entire square, if they fall within the circle, they are colored blue otherwise colored orange. We keep track of the total number of points and the number of points that are inside the circle. Now take the ratio of the probability that point lies inside the circle by considering equation of circle $x^2 + y^2 = r^2$,

$$\frac{N_{inner}}{N_{total}} = \frac{\pi r^2\ (area\ of\ circle)}{4r^2\ (area\ of\ square)}$$

$$\pi = \frac{4 *\ N_{inner}}{N_{total}}$$

Python code for estimating the value of π using Monte Carlo simulation is given below,



```
import numpy as np
N = 1000000
x = np.random.rand(N)
y = np.random.rand(N)
inside=0
for i in range(N):
    if np.sqrt(x[i]**2 + y[i]**2) <1:
        inside +=1
pi = (float(inside) / N) * 4
print("MC estimation of pi: %0.5f"%(pi))
```

```
MC estimation of pi: 3.14262
```

Figure 2-2 Python code to estimate the value of π using Monte Carlo method

Here we have obtained value of π which is same as 3.14. As discussed before as the number of random samples increases, the accuracy also increases which can be seen in below table 2.1. In which for different number of simulation we got different values of π.

N	100	1000	1000000
π	2.84	3.1268	3.14262

Table 2.1: Results of different number of simulation of π

2.6.2. Example II: Estimation of the Robbins constant

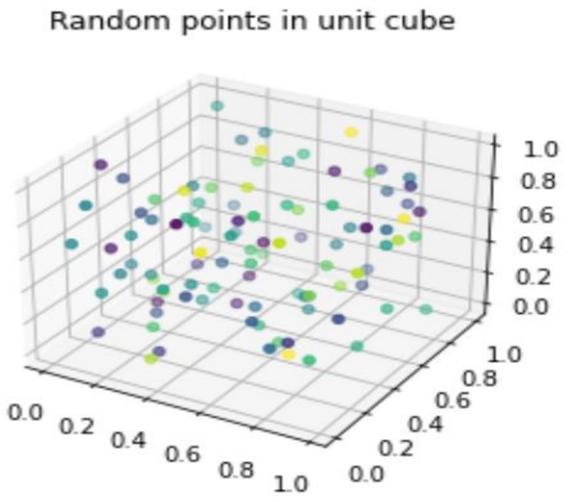
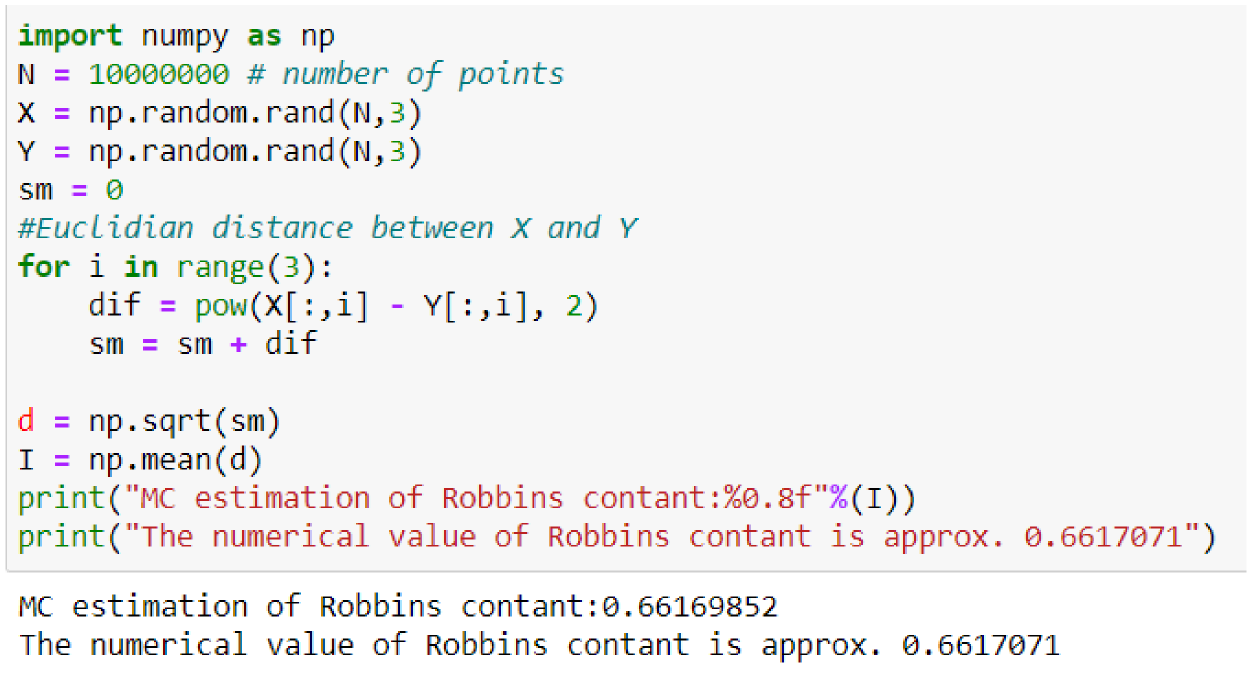


Figure 2-3 Random point in unit cube

Robbins' constant states that if we select two random points in a unit cube than average distance between that two points are always constant [4]. The numerical value of the Robbins constant is 0.661707182 [5].

For estimation of Robbins' constant with help of Monte Carlo consider the unit cube and imagine any two random points inside that cube now for measure distance between that point use Euclidian distance formulas. Now do the same for large number of two different random point and measure the distance between two points. As the Monte Carlo approximation taking the mean of all result we can obtain the expected value of the Robbins constant. Here is the python code for estimation of the Robbins constant.



```python
import numpy as np
N = 10000000 # number of points
X = np.random.rand(N,3)
Y = np.random.rand(N,3)
sm = 0
#Euclidian distance between X and Y
for i in range(3):
    dif = pow(X[:,i] - Y[:,i], 2)
    sm = sm + dif

d = np.sqrt(sm)
I = np.mean(d)
print("MC estimation of Robbins contant:%0.8f"%(I))
print("The numerical value of Robbins contant is approx. 0.6617071")
```

```
MC estimation of Robbins contant:0.66169852
The numerical value of Robbins contant is approx. 0.6617071
```

Figure 2-4 Python code to estimate the value of Robbins constant

3. Stochastic Process

A model for describing the behavior of a family of random experiments performed one after the other in time or it can be thought of as the result of a random experiment where the exact value is partly revealed over time, one piece after another [6]. A stochastic process changes over time. Standard examples of stochastic processes are temperature curves, the evolution of a stock price index, the sequence of the wealth of a gambler taking part in a series of games of chance, etc. A stochastic process is mainly used for randomly evolving phenomena with random components are involved. Since the stochastic process moves across time we need to know when an investor knows what. A stochastic process is defined on a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ where,

Ω – The space or the set of all possible outcomes also called all states of world

$\mathcal{A}$ – The sample space Ω is divided into subsets. $\mathcal{A}$ (Subset of all admissible events) is called σ algebra

$\mathbb{P}$ - Admits probability to each admissible events and hence called the probability measure, $\mathbb{P}: \mathcal{A} \rightarrow [0,1]$

3.1. A Stochastic Process Ω Vs Random Variable

A random variable X maps Ω to real numbers, $X: \Omega \rightarrow \mathbb{R}$

A stochastic process X_t, maps $\Omega \times \tau$ to real numbers, where τ is set of time periods

$\{X_t\}_t: \Omega \times \tau \ \rightarrow \mathbb{R}$, $\tau \in [0, \infty)$

3.1.1. Trajectory

It is one sample path of a stochastic process $\{X_t\}_t$. A trajectory can be denoted as follows. A trajectory summarizes the respective evolution of an outcome of a process across time. If we run a Monte Carlo simulation for different w, leads to different paths or trajectories.

$t \rightarrow X_t(w)$, w is one realization of the world across different time periods. Below such trajectory's example is shown,

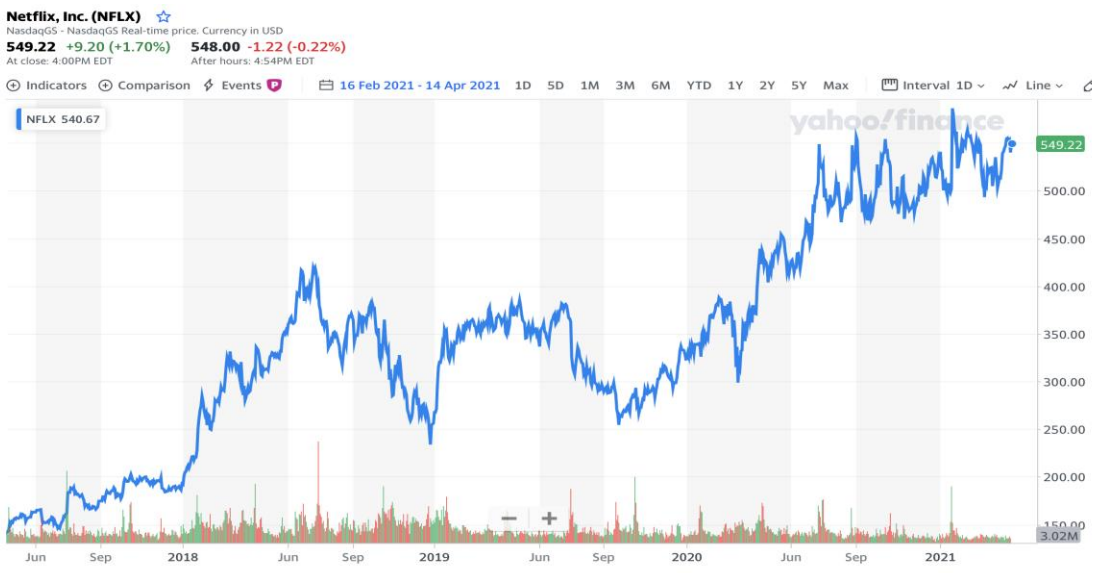


*Figure 3-1 Stock price of Netflix 2017-2021 * Image acquired from Yahoo Finance*

4. Brownian Motion

A Stochastic process $\{ W_t : 0 \leq t \leq \infty \}$, is a standard Brownian motion if the following properties are satisfied [7],

1. W_0=0. The value of the Brownian motion at t =0 takes the value 0 i.e. all realisations of the Brownian motion takes the value 0 at time t =0

2. Brownian Motion is normally distributed, $W_t \sim \mathcal{N}(0, t)$ with mean 0 and variance t also, the Brownian motion increment, $W_t - W_s \sim \mathcal{N}(0, t - s)$, is also normally distributed with mean 0 and variance $t - s$, for $0 \leq s < t$. We can note that the variance is proportional to the time

3. The process W_t has stationary and independent increments

a. Stationary increments means that $W_{t_2} - W_{t_1}$ has the same distribution as $W_{t_2+a} - W_{t_1+a} \sim \mathcal{N}(-t_1)$

b. The term independent increment means that for every choice of $t_0 < t_2 \ldots . < t_n$ the increment random variable $W_{t_1} - W_{t_2}$,$W_{t_3} - W_{t_4}$,…, $W_n - W_{n-1}$ are jointly independent i.e., no dependency on how the process moves in different time intervals

Some more properties of Brownian motion are,

1. Covariance of 2 Brownian motion path is, minimum of $\min(t, s)$
$Cov(W_t - W_{s)} = \min(t, s)$, this is in relation that BM has independent increments

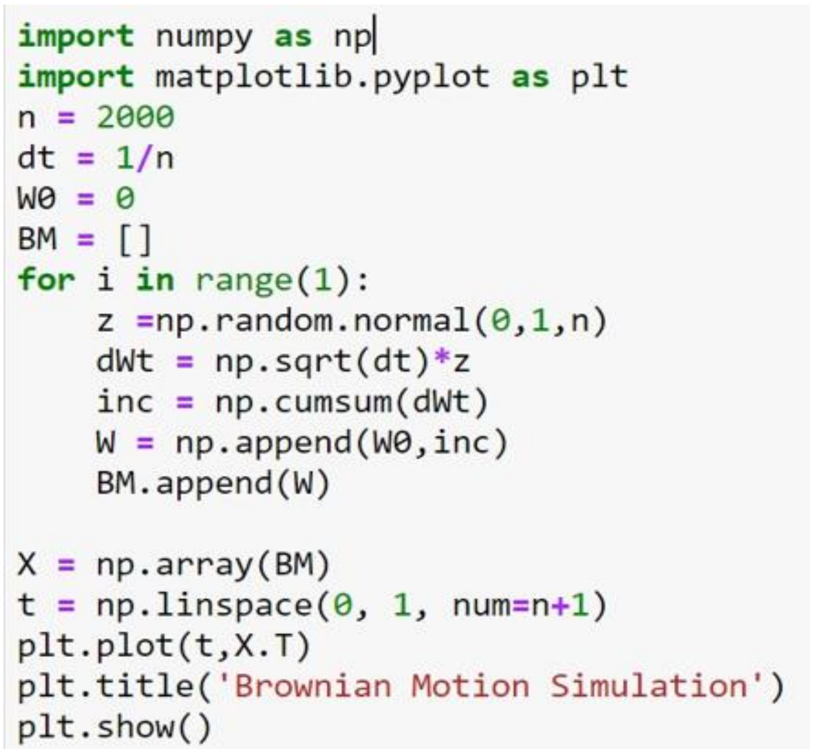
2. BM is a Markov process: Past is irrelevant for where I am going to be in the future

3. BM is a Martingale: It states that the expected value of where I am going to be in the future is where I am now i.e., $E[W(t + s)/F_t] = W(t)$

4. BM has continuous path.

4.1. Simulation of Brownian motion

A Brownian motion is simulated for a period of time, with the time divided into 'n' parts and a random variable is assigned for each time interval and the cumulative sum is taken. This approach is called Random walk simulation. The python code for a Brownian motion simulation is shown below. The number of iteration n is taken as 2000. Time is found by (1/2000). For each time interval, a random value is assigned which may vary according to different realizations taken and the cumulative sum is taken and plotted. The graphs will look different in different iterations. Thus, we say that BM is $W_t \sim \mathcal{N}(0, t)$.



```python
import numpy as np
import matplotlib.pyplot as plt
n = 2000
dt = 1/n
W0 = 0
BM = []
for i in range(1):
    z =np.random.normal(0,1,n)
    dWt = np.sqrt(dt)*z
    inc = np.cumsum(dWt)
    W = np.append(W0,inc)
    BM.append(W)

X = np.array(BM)
t = np.linspace(0, 1, num=n+1)
plt.plot(t,X.T)
plt.title('Brownian Motion Simulation')
plt.show()
```

Figure 4-1 Python code to simulate Brownian motion simulated for n=2000

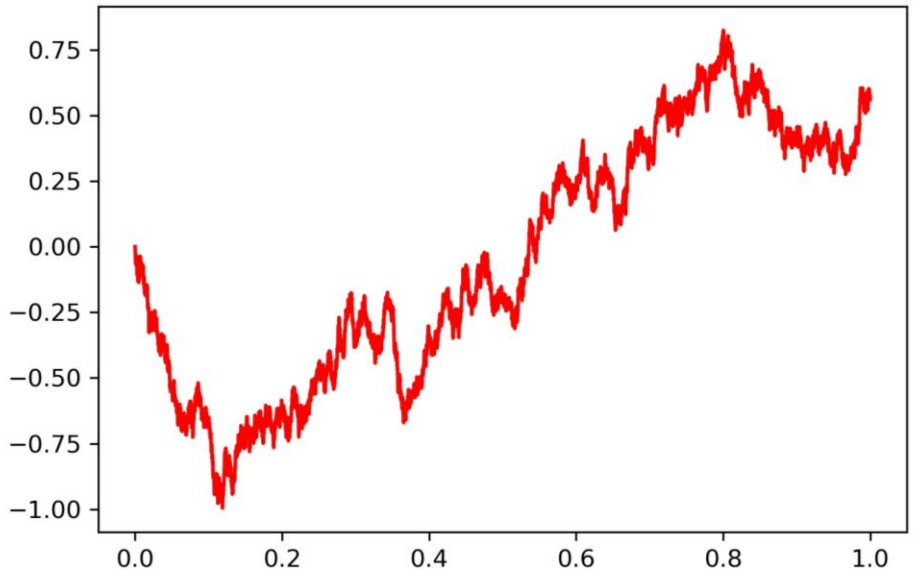


Figure 4-2 Plot of one realization of Brownian motion simulated for n=2000

5. Stochastic Differential Equation

A stochastic differential equation (SDE) [6] is a differential equation in which one or more of the terms is a stochastic process, resulting in a solution that is also a stochastic process. SDEs are used to model, various phenomena such as unstable stock prices or physical systems subject to thermal fluctuations. A Stochastic differential equation is given as follows

SDE = ODE + GWN [8]

ODE – Ordinary Differential Equation

GWN – Gaussian White Noise

Let an Ordinary Differential Equation $x(t)$ has the expected slope

$$f(x) = \frac{dx}{dt}$$

But the realized slope of $x(t)$ deviates from the expected slope due to GWN,

$$\frac{dx}{dt} = f(x) + \xi_{t+dt}$$

The noise is normally distributed with mean 0 and variance 1, $\xi_{t+dt} \sim \mathcal{N}(0,1)$, the ξ_{t+dt} is the realized return shock at the end of the time period dt.

The noise can arrive at different intensities as per the market conditions. If we assume dx_t as the return of an asset for the trading period dt, there can be highs and lows. Hence, we introduce a variable that captures intensities i.e., $g(x)$ which is called stochastic volatility function,

$$\frac{dx}{dt} = f(x) + g(x) \times \xi_{t+dt}$$

5.1. Solution of the Stochastic Differential Equation

An SDE can have 2 kinds of solutions. The simulation approach depends on the type of solution of SDE,

1. Closed form solution
2. No exact solution

When SDE has a closed-form solution/Explicit Solution then we can do exact simulation. When SDE does not have a closed-form solution, we need to use discrete simulation methods given in [9] like,

1. Euler–Maruyama method
2. Milstein method

Since in our applications we are able to exactly simulate the solution of the SDE, we do not use any discrete scheme. We refer to the book [9], for the interested readers about the discretization schemes.

5.2. SDE in Black Scholes model for option pricing

A Stochastic equation takes the form as shown below

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

X_t = Continuous time stochastic process
W_t = Wiener Process (Standard Brownian motion)
b = drift coefficient
σ = volatility

Black Scholes model uses geometric Brownian motion as shown below,

$$dS_t = rS_t dt + \sigma S_t dW_t$$

S_t = Continuous time stochastic process
W_t = Wiener Process (Standard Brownian motion)
r = risk free rate of interest
σ = volatility

The equation will give exact solution as shown; hence we are not interested in other methods like Euler–Maruyama,

$$S_t \coloneqq S_0 e^{\left(r - \frac{\sigma^2}{2}\right)t + \sigma W_t}$$

6. Black Scholes Model

The Black-Scholes Option Pricing Model is a mathematical model for pricing options contracts created by Fisher Black and Myron Scholes. The Black-Scholes Model is also referred to as the Black-Scholes-Merton model for Robert Merton's contribution to the work. For their work in options pricing theory, Scholes and Merton received the 1997 Nobel Prize in Economics [10]. This model is one of the best ways to determine the fair price of the options.

The Black Scholes model makes certain assumptions [11]:

1 The price of underlying assets follows a Geometric Brownian Motion.

2. The risk free rate of interest is constant

3. No dividends are paid out during the life of the option.

4. There are no transaction costs in buying the option.

5. Volatility remain constant throughout the contract.

The Black-Scholes Model has five inputs: the current stock price, the option strike price, the risk-free interest rate, the time remaining until expiration, and the underlying security's volatility. Here, the BSM formula shows how to find the price of an option contract (call and put option) by using simple formula but here we are using European option. The formula for European call C and put option P is [12]:

$$C \coloneqq S_0 N(d_1) - Ke^{-rT} N(d_2)$$

$$P \coloneqq Ke^{-rT} N(-d_2) - S_0 N(-d_1)$$

Where,

$$d_1 \coloneqq \frac{\ln\left(\frac{S_0}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$$

$$d_2 \coloneqq d_1 - \sigma\sqrt{T}$$

Notation:

S_0- Initial price of the stock (Today's price of the stock)

K- Option Strike price

σ- Volatility of the stock

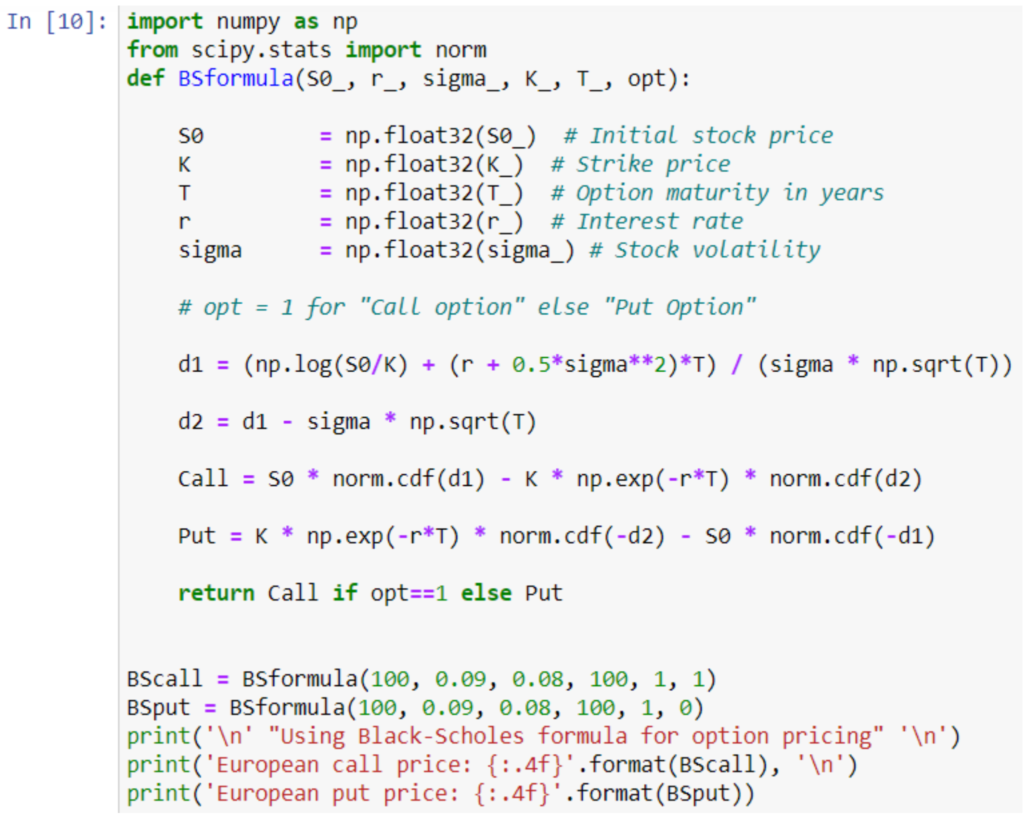
T- Contract time period

r- Risk free interest rate

W_t- Brownian Motion

N- Cumulative distribution function for standard normal distribution

By using these formulas we obtained option price for call and put option. Here, we have taken initial stock price and strike price are 100, risk-free rate of interest is 0.09, Volatility is 0.08, and expiration time is taken is of 1 year. Python code to calculate the option price is given below,



```
In [10]: import numpy as np
         from scipy.stats import norm
         def BSformula(S0_, r_, sigma_, K_, T_, opt):

             S0         = np.float32(S0_)  # Initial stock price
             K          = np.float32(K_)  # Strike price
             T          = np.float32(T_)  # Option maturity in years
             r          = np.float32(r_)  # Interest rate
             sigma      = np.float32(sigma_) # Stock volatility

             # opt = 1 for "Call option" else "Put Option"

             d1 = (np.log(S0/K) + (r + 0.5*sigma**2)*T) / (sigma * np.sqrt(T))

             d2 = d1 - sigma * np.sqrt(T)

             Call = S0 * norm.cdf(d1) - K * np.exp(-r*T) * norm.cdf(d2)

             Put = K * np.exp(-r*T) * norm.cdf(-d2) - S0 * norm.cdf(-d1)

             return Call if opt==1 else Put


         BScall = BSformula(100, 0.09, 0.08, 100, 1, 1)
         BSput = BSformula(100, 0.09, 0.08, 100, 1, 0)
         print('\n' "Using Black-Scholes formula for option pricing" '\n')
         print('European call price: {:.4f}'.format(BScall), '\n')
         print('European put price: {:.4f}'.format(BSput))
```

Figure 6-1 Python code to calculate option price for call and put option using exact solution

According to this formula we obtained call and put option prices are 9.1060 and 0.4991 respectively.

7. Option pricing in Black Scholes model using Monte Carlo simulation

The following equation is used in simulation of stock price in Black Scholes model,

$$S_t := S_0 e^{\left(r-\frac{\sigma^2}{2}\right)t+\sigma W_t}$$

It is commonly used in finance for modeling the option price. The Brownian motion becomes zero at time $t = 0$ which is not possible in case of stock prices because stock prices are never zero. That's the reason why geometric Brownian motion is used instead of Brownian motion. The above formula can be described by a stochastic differential equation. [12]

By using this formula we obtain stock price at time t by simulating the formula thousands of times. We can imagine these prices as huge matrix. After that, we obtain a payoff vector. Payoff is profit made by the holder at the maturity date [12].

Payoff of European call option:

At the expiration time T, if the stock price at time T is greater than strike price K then,

$$Payoff := S_T - K$$

And if stock price S_T is smaller than strike price then $payoff = 0$

So, the value of call option is given by,

$$Payoff\ vector\ for\ call\ option := \max(S_T - K, 0)$$

Payoff of European put option:

At the expiration time T, if the stock price at time T is less than strike price K then,

$$Payoff := K - S_T$$

And if stock price S_T is greater than strike price then $payoff = 0$

So, the value of put option is given by,

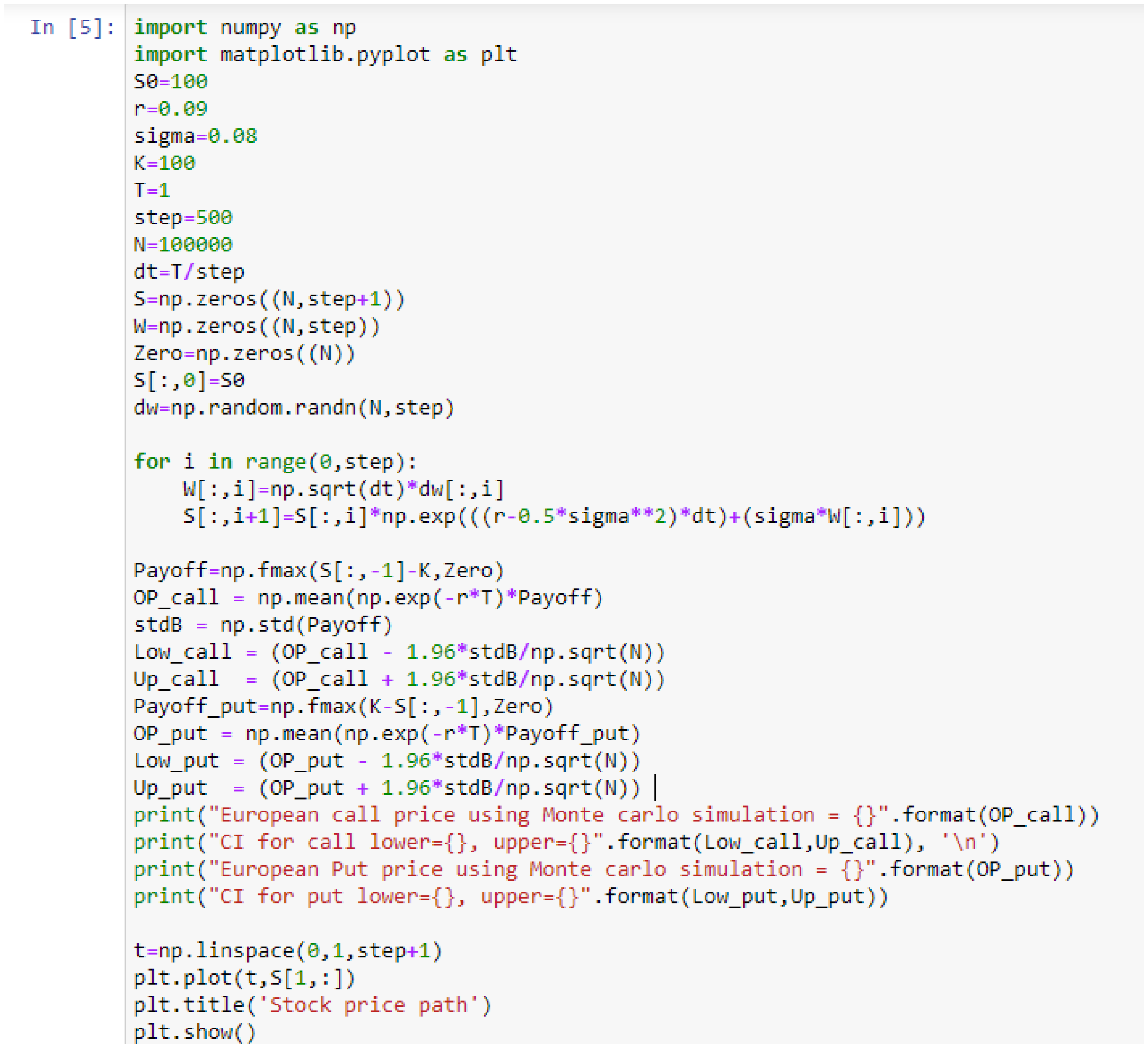
$$Payoff\ vector\ for\ put\ option := \max(K - S_T, 0)$$

After calculating payoff vector we will calculate arithmetic mean of that vector so according to Monte Carlo we will have accurate expected value and then we will discount it to the today's price in order to obtain Option Price.

$$Option\ Price := e^{-rT} * \mathbb{E}(payoff)$$

7.1. A stock price simulation in Black Scholes model using Monte Carlo simulation

By doing simulation as mentioned above for hundred thousands of time we obtained option price for call and put option. Here, we have taken initial stock price and strike price are 100, risk-free rate of interest is 0.09, Volatility is 0.08, and expiration time is taken is of 1 year. So all the values are same as taken above in section 6.1.1. So we can compare option prices with one obtained using Monte Carlo simulation. Python code to calculate the option price is given below,



```
In [5]: import numpy as np
        import matplotlib.pyplot as plt
        S0=100
        r=0.09
        sigma=0.08
        K=100
        T=1
        step=500
        N=100000
        dt=T/step
        S=np.zeros((N,step+1))
        W=np.zeros((N,step))
        Zero=np.zeros((N))
        S[:,0]=S0
        dw=np.random.randn(N,step)

        for i in range(0,step):
            W[:,i]=np.sqrt(dt)*dw[:,i]
            S[:,i+1]=S[:,i]*np.exp(((r-0.5*sigma**2)*dt)+(sigma*W[:,i]))

        Payoff=np.fmax(S[:,-1]-K,Zero)
        OP_call = np.mean(np.exp(-r*T)*Payoff)
        stdB = np.std(Payoff)
        Low_call = (OP_call - 1.96*stdB/np.sqrt(N))
        Up_call  = (OP_call + 1.96*stdB/np.sqrt(N))
        Payoff_put=np.fmax(K-S[:,-1],Zero)
        OP_put = np.mean(np.exp(-r*T)*Payoff_put)
        Low_put = (OP_put - 1.96*stdB/np.sqrt(N))
        Up_put  = (OP_put + 1.96*stdB/np.sqrt(N))
        print("European call price using Monte carlo simulation = {}".format(OP_call))
        print("CI for call lower={}, upper={}".format(Low_call,Up_call), '\n')
        print("European Put price using Monte carlo simulation = {}".format(OP_put))
        print("CI for put lower={}, upper={}".format(Low_put,Up_put))

        t=np.linspace(0,1,step+1)
        plt.plot(t,S[1,:])
        plt.title('Stock price path')
        plt.show()
```

Figure 7-1 Python code to obtain option price using Monte Carlo Simulation

According to this calculation using code, we obtained option price for call and put option are 9.1207 and 0.4952 respectively which are almost similar to the one with using exact solution formula as discussed in section 6.

Here is the results of option price for call and put option and also 95% confidence interval is calculated. Graph of stock price for 1 year of time period is also shown.

```
European call price using Monte carlo simulation = 9.120739415359287
CI for call lower=9.07174210781918, upper=9.169736722899394

European Put price using Monte carlo simulation = 0.49523129860230386
CI for put lower=0.44623399106219597, upper=0.5442286061424118
```

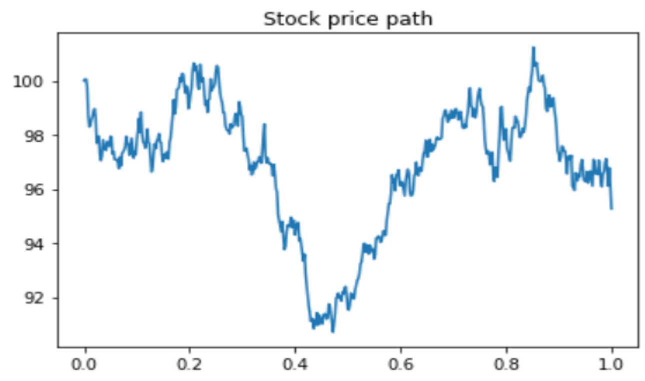


Figure 7-2 Output of above python code

7.2. Basket option pricing in Black Scholes model using Monte Carlo simulation

In basket option pricing average of two or more stock's option prices are grouped together in the basket. In this weighted average prices of the stocks at expiration time is calculated in order to obtain payoff vector in a basket. After calculating payoff vector, it is discounted to today's value to obtain option price. So formulas to obtain option prices are given as,

$$S_i(t) = S_i(0)e^{\left(r-\frac{\sigma^2}{2}\right)t+\sigma W_i(t)}$$

$$Payoff\ vector\ for\ call\ option \coloneqq \max[(\frac{1}{n}\sum_{i=1}^{n} w_i S_i(T) - K),0]$$

$$Payoff\ vector\ for\ put\ option \coloneqq \max[(K - \frac{1}{n}\sum_{i=1}^{n} w_i S_i(T)),0]$$

$$Option\ Price \coloneqq e^{-rT} * \mathbb{E}(payoff)$$

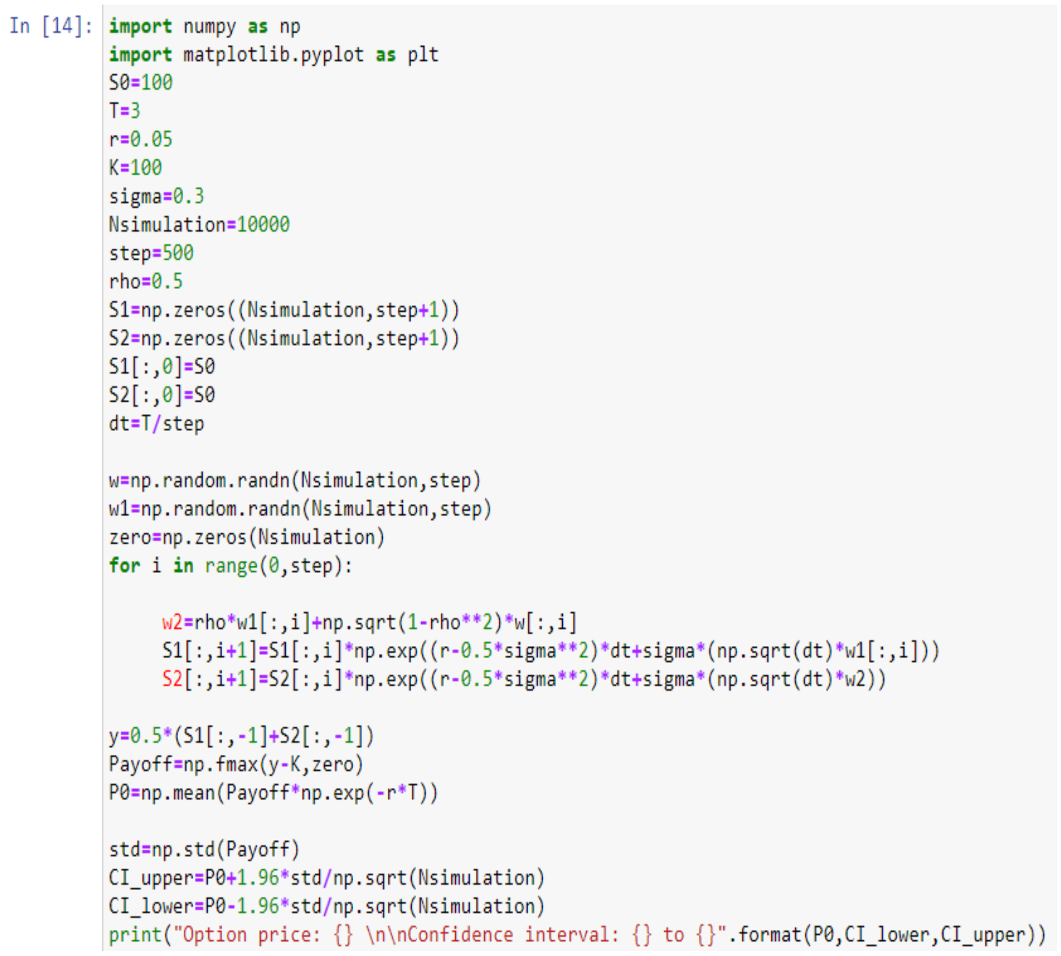
Where,

i- Number of stocks (1,2,…,n)

W_i-correlated Brownian motion

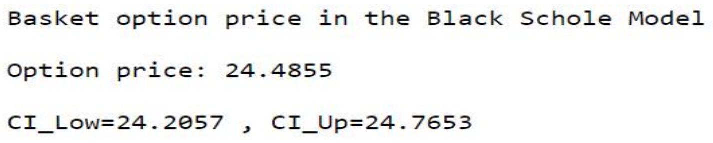
7.2.1. Basket option pricing for 2-stocks in Black Scholes model using Monte Carlo simulation

Above formulas are used in simulation of stock prices. In this, we have also correlated both stock price paths and having the same correlation of 0.5. Here, we have taken initial stock price and strike price are 100, risk-free rate of interest is 0.05, Volatility for both stocks are taken same of 0.3 and expiration time is taken is of 3 years. In this case, the exact price of the basket option is equal to 24.345 [14]. After simulating this for ten thousand times we obtained option price for call option 24.4855. Here is the python code to simulate option price and its results in basket option is given,



```
In [14]: import numpy as np
         import matplotlib.pyplot as plt
         S0=100
         T=3
         r=0.05
         K=100
         sigma=0.3
         Nsimulation=10000
         step=500
         rho=0.5
         S1=np.zeros((Nsimulation,step+1))
         S2=np.zeros((Nsimulation,step+1))
         S1[:,0]=S0
         S2[:,0]=S0
         dt=T/step

         w=np.random.randn(Nsimulation,step)
         w1=np.random.randn(Nsimulation,step)
         zero=np.zeros(Nsimulation)
         for i in range(0,step):

             w2=rho*w1[:,i]+np.sqrt(1-rho**2)*w[:,i]
             S1[:,i+1]=S1[:,i]*np.exp((r-0.5*sigma**2)*dt+sigma*(np.sqrt(dt)*w1[:,i]))
             S2[:,i+1]=S2[:,i]*np.exp((r-0.5*sigma**2)*dt+sigma*(np.sqrt(dt)*w2))

         y=0.5*(S1[:,-1]+S2[:,-1])
         Payoff=np.fmax(y-K,zero)
         P0=np.mean(Payoff*np.exp(-r*T))

         std=np.std(Payoff)
         CI_upper=P0+1.96*std/np.sqrt(Nsimulation)
         CI_lower=P0-1.96*std/np.sqrt(Nsimulation)
         print("Option price: {} \n\nConfidence interval: {} to {}".format(P0,CI_lower,CI_upper))
```

Figure 7-3 Python code to obtain option price for 2-stocks using Monte Carlo Simulation



```
Basket option price in the Black Schole Model

Option price: 24.4855

CI_Low=24.2057 , CI_Up=24.7653
```

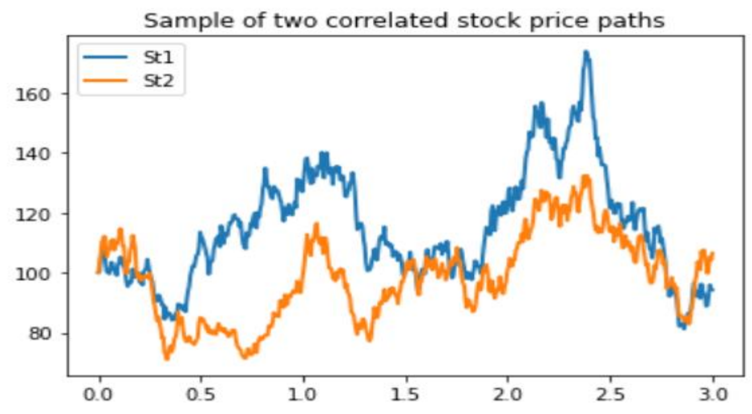
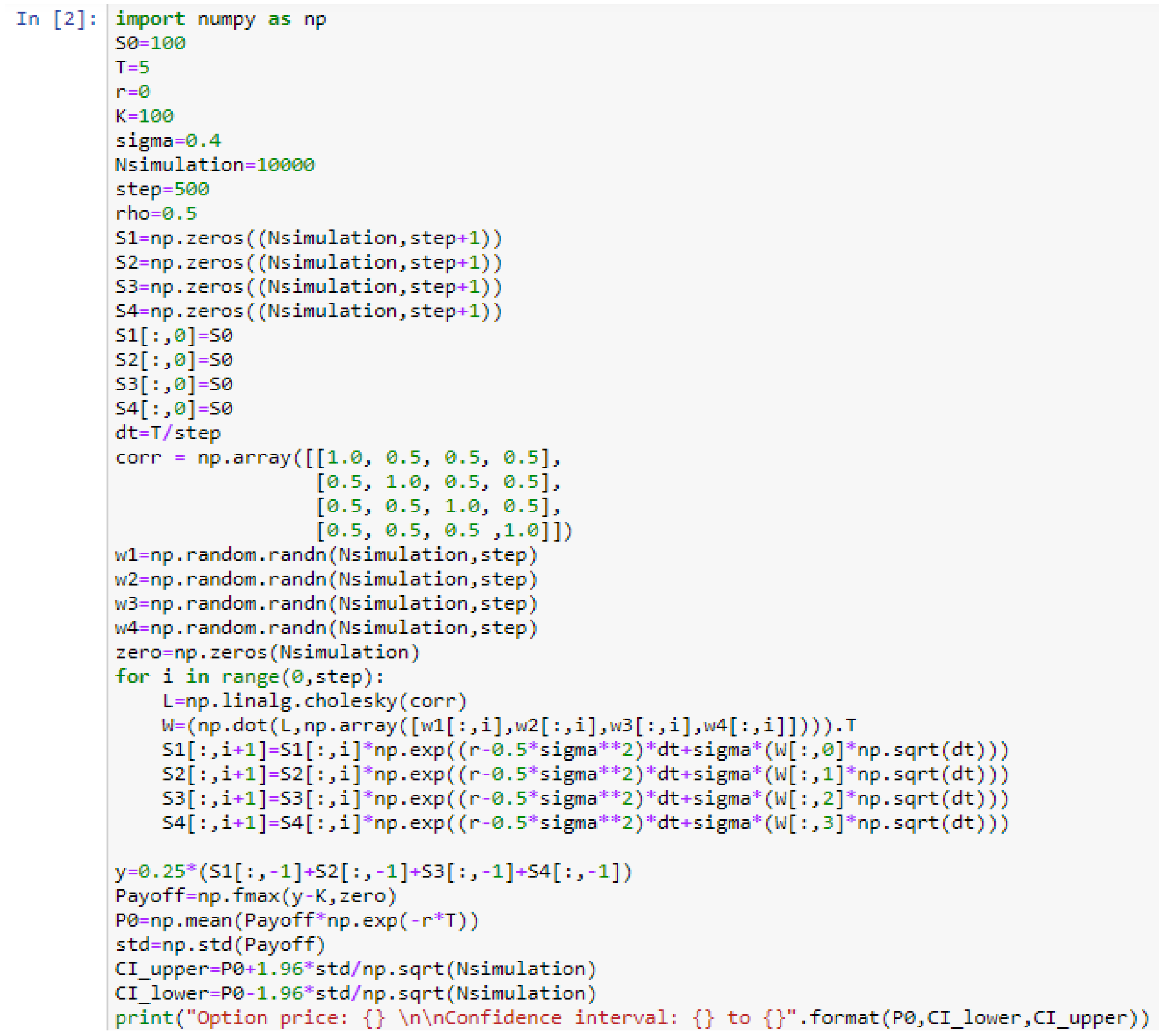


Figure 7-4 Output of python code for 2-stocks

7.2.2. Basket option pricing for 4-stocks in Black Scholes model using Monte Carlo simulation

In this, we have also correlated four stock price paths and having same correlation of 0.5. Here, we have taken initial stock price and strike price are 100, risk free-rate of interest is 0, Volatility for all four stocks are taken same of 0.4, and expiration time is taken is of 5 year. In this case, the exact price of the basket option is equal to 28 [14]. After simulating this for ten thousand times we obtained option price for call option 28.0931. Here is the python code to simulate option price and its results in basket option is given,



```
In [2]: import numpy as np
        S0=100
        T=5
        r=0
        K=100
        sigma=0.4
        Nsimulation=10000
        step=500
        rho=0.5
        S1=np.zeros((Nsimulation,step+1))
        S2=np.zeros((Nsimulation,step+1))
        S3=np.zeros((Nsimulation,step+1))
        S4=np.zeros((Nsimulation,step+1))
        S1[:,0]=S0
        S2[:,0]=S0
        S3[:,0]=S0
        S4[:,0]=S0
        dt=T/step
        corr = np.array([[1.0, 0.5, 0.5, 0.5],
                         [0.5, 1.0, 0.5, 0.5],
                         [0.5, 0.5, 1.0, 0.5],
                         [0.5, 0.5, 0.5 ,1.0]])
        w1=np.random.randn(Nsimulation,step)
        w2=np.random.randn(Nsimulation,step)
        w3=np.random.randn(Nsimulation,step)
        w4=np.random.randn(Nsimulation,step)
        zero=np.zeros(Nsimulation)
        for i in range(0,step):
            L=np.linalg.cholesky(corr)
            W=(np.dot(L,np.array([w1[:,i],w2[:,i],w3[:,i],w4[:,i]]))).T
            S1[:,i+1]=S1[:,i]*np.exp((r-0.5*sigma**2)*dt+sigma*(W[:,0]*np.sqrt(dt)))
            S2[:,i+1]=S2[:,i]*np.exp((r-0.5*sigma**2)*dt+sigma*(W[:,1]*np.sqrt(dt)))
            S3[:,i+1]=S3[:,i]*np.exp((r-0.5*sigma**2)*dt+sigma*(W[:,2]*np.sqrt(dt)))
            S4[:,i+1]=S4[:,i]*np.exp((r-0.5*sigma**2)*dt+sigma*(W[:,3]*np.sqrt(dt)))

        y=0.25*(S1[:,-1]+S2[:,-1]+S3[:,-1]+S4[:,-1])
        Payoff=np.fmax(y-K,zero)
        P0=np.mean(Payoff*np.exp(-r*T))
        std=np.std(Payoff)
        CI_upper=P0+1.96*std/np.sqrt(Nsimulation)
        CI_lower=P0-1.96*std/np.sqrt(Nsimulation)
        print("Option price: {} \n\nConfidence interval: {} to {}".format(P0,CI_lower,CI_upper))
```

Figure 7-5 Python code to obtain option price for 4-stocks using Monte Carlo Simulation

```
Basket option price in the Black Schole Model

Option price: 28.0931

CI_Low=27.6787 , CI_Up=28.5075
```

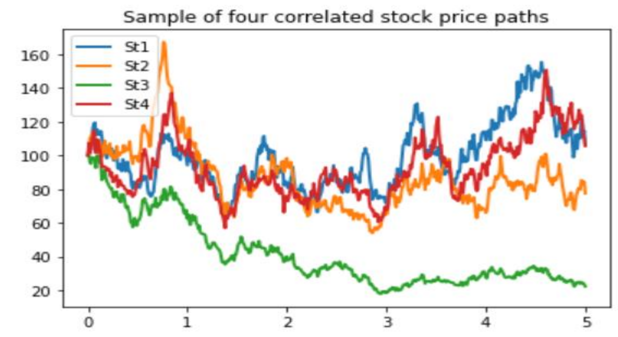


Figure 7-6 Output of python code for 4-stocks

8. Conclusion

The aim of the project is to obtain the price of the basket option in the Black Scholes model using the Monte Carlo simulation. After doing the simulation for different cases using python as a tool, we obtained option pricing for call and put option using the Monte Carlo method. Monte Carlo simulation has played a very important role to find the most probable values for all different cases, which are explained deeply in this report. For all the cases that we have considered in this report, we have shown that the exact option prices fall into the 95% confidence interval of our simulations. This ensures that our simulations deliver an accurate result for both the European options and the basket options. Main difficulty for pricing basket options is to efficiently deal with the correlation structure among the stock prices. However, using the Cholesky decomposition within the Monte Carlo simulation helps us to overcome this difficulty.

Further research can focus on pricing the basket options in a more complicated market model than the Black-Scholes model such as the Heston model or providing a comparison between Monte Carlo method and other methods such as machine learning techniques in the option pricing context.

9. References

[1] R. Korn, E. Korn, and G. Kroisandt. Monte Carlo Methods and Models in Finance and Insurance. Chapman&Hall/Crc Financial MathematicsSeries, London, 2010

[2] https://www.youtube.com/watch?v=MiybniIIvx0&ab_channel=SkyViewTrading

[3] Kroese, D. P., Brereton, T., Taimre, T., and Botev, Z. I. (2014). "Why the Monte Carlo method is so important today?", WIREs Comput Stat. 6 (6): 386–392

[4] *Robbins, David P.; Bolis, Theodore S. (1978), "Average distance between two points in a box (solution to elementary problem E2629)",* American Mathematical Monthly*, 85 (4): 277–278,* doi*:*10.2307/2321177

[5] Plouffe, Simon, "The Robbins constant", Miscellaneous Mathematical Constants

[6] https://www.youtube.com/watch?v=gg7N4QAOcXE

[7] https://www.youtube.com/watch?v=7mmeksMiXp4

[8] https://en.wikipedia.org/wiki/White_noise

[9] Kloeden, P.E. and Platen, E. (1992). Numerical Solution of Stochastic Differential Equations. Springer, Berlin

[11] https://optionalpha.com/options/options-pricing

[12] Ahmad Dar, A., Anuradha, N., and Abdur Rahman, B.S. (2017), "Option Pricing Using Monte Carlo Simulation", British Journal of Economics, Finance and Management Sciences, March 2017, Vol. 13 (2), pp. 53-81.

[13] https://www.investopedia.com/terms/b/blackscholes.asp

[14]https://quant.stackexchange.com/questions/33694/basket-option-pricingof-two-stocks