

Academic Year: 2020-2021

*Final Year Project*

Evaluation of Call Option Prices via Monte Carlo Methods applying Variance Reduction Techniques

Darío Martínez Barrera (Degree in Economics)

Tura Ventolà Franch (Degree in Economics)

Laia Sagué Carbonell (Degree in Business Administration)

Tutor

Elisa Alòs Alcalde

Abstract

In this paper, Monte Carlo reduction variance techniques are studied in order to approximate the value of European Call options. Simulation modeling provides an important and key tool for analysis and insights into complex mathematical systems. Control variates reduce simulation errors by applying variance reduction techniques. After the analysis needed to acquire the knowledgment, a code in Python was created with the intention of comparing the results and concluding which technique was the most accurate.

*Keywords:*

*Monte Carlo, Black-Scholes, option pricing, variance reduction, European Call option.*

Acknowledgements

We are grateful to our supervisor Elisa Alòs Alcalde for the support, guidance and the opportunity to perform our final year project in collaboration with her.

Index

[**Introduction**](#_heading=h.o4p1px6bz7f1) **1**

[**Financial Derivatives, European Call Options, and Black Scholes**](#_heading=h.iufr78qiz55b) **2**

[**Monte Carlo Methods**](#_heading=h.6aig02vm7ssb) **5**

[**Variance reduction techniques**](#_heading=h.5r549t6wq06e) **7**

[3.1 Antithetic Variates Method](#_heading=h.wbtekpanryut) 8

[3.2 Control Variate Method](#_heading=h.lhft8ge8ruqi) 9

[3.3 Conditional Monte Carlo](#_heading=h.5pv32cmba5tm) 10

[3.4 Antithetic Conditional Monte Carlo](#_heading=h.qiangsdxs7dy) 11

[**Python code**](#_heading=h.s7wi99mmo7f3) **12**

[4.1 Black Scholes](#_heading=h.o8tqw52n1owu) 13

[4.2 Crude Monte Carlo](#_heading=h.55p05mcet8ot) 13

[4.3 Antithetic Monte Carlo](#_heading=h.7zbqzc21o16t) 15

[4.4 Control Variate Monte Carlo](#_heading=h.xlq30w5q5cze) 16

4.5 [Conditional Monte Carlo](#_heading=h.965v0jkopcb0) 18

4.6 [Antithetic Conditional Monte Carlo](#_heading=h.6ij56q41zcnl) 19

[**Results**](#_heading=h.t64qw6hzm1am) **20**

[5.1 Initial Variables](#_heading=h.63hk15rwp32f) 21

[5.2 Effects of Changes on Initial Variables](#_heading=h.bqcf9f1t2iq0) 23



# Introduction

Pricing an option is an essential part of the risk management and hedging analysis. Investment firms and banks have advanced techniques running in fast, professional computers to get to the real value of an option. Our motivation is to understand those techniques used and to be able to replicate them in a shorter scale using Python. As our computers are not as powerful as the ones used by banks and hedge funds, we will have to understand and use mathematical variance reduction methods that will help us reduce the variance of our results while keeping the computational complexity low.

Our main limitation was the shortage of computer coding skills. This limitation will be solved by applying methods that rely on a strong mathematical base rather than on complex, coding-intense algorithms. In particular using variation reduction techniques that allowed us to have better results without carrying our computational-intensive codes.

The project will be divided into six sections. The first section will contain an introduction on financial derivatives with a more detailed focus on European Call options, which will be explained. The section will end with an explanation and derivation of the Black Scholes Model for option pricing and the distribution of prices for the underlying under this model.

The second section will contain the theory behind Monte Carlo methods and its mathematical properties. This will be followed by an explanation on the application of those Monte Carlo methods for option pricing. The third section will follow the idea of pricing options via Monte Carlo adding the explanation and application of different mathematical methods that can be carried out in the Monte Carlo computation process that will help us to increase accuracy while keeping computation intensity low.

In the fourth section, Python codes created to simulate all theories previously explained will be presented and explained step by step for easy comprehension. The fifth section will be dedicated to show the results yielded by the previously explained codes, [and to study the effects of changes in the variables introduced (to be confirmed)].

The project will end with the exposition of our conclusions given the results obtained and the knowledge gathered throughout the whole execution of the project.

# Financial Derivatives, European Call Options, and Black Scholes

A derivative contract is a financial security whose price depends on another underlying asset. There are many types of derivatives, but the most common ones are futures, forwards, options, and swaps. These securities are used by investors for many goals such as speculation or hedging.

There are many different types of options. Depending on its characteristics, options can be classified as European, the exercise of the option is only permitted at agreed time, American, allowing to exercise the option rights at any time, and Exotic options, have many other characteristics that make each one of them different from the others -e.g. Bermudan, Asian, Barrier, Binary, etc.- Each derivative yields a different level of risk so they can fit the needs of the investor.

The kind of option that we will study in this thesis, the call option, alongside with its opposite, the put option, are known as Vanilla options for being one of the simplest sorts of options. A call option is a contract between two parties that gives the right to buy an asset for an agreed price. As explained above, the European version of the call option forces the contract to be executed at the date established on the contract, this is known as Maturity Time, *T*. The price negotiated is known as the strike, *K*, and it is all based on the price of the underlying asset, , at time period *t*.

Although we will just study the case for the European calls, all the analyses presented in this thesis are valid and can be completely applicable on any other European option by changing the formula defining the payoff of the call option (1) by the one defining the chosen European option.

The price of the call option, called premium, is the price of buying the option, independent of the final value of the asset. If the price of the underlying asset gets below the strike price K, the buyer of the option loses the premium paid, this is the maximum loss a buyer can concur. On the other hand, if the asset’s price is above the strike price, the buyer gets as profit the difference between the asset price and the premium.

The formula that defines the payoff of a call option is:

(1)

The premium is neglected from the formula as it will be required to pay independently of the behaviour of the underlying asset, and it is considered a sunk cost once the option has been bought.

And the present value of the call option is,

(2)

Where: = present value of the option; = risk-free rate; = maturity time; = price of the underlying asset; = strike price



(Insertar Fuente)

As an example, in illustration 1, we have a Call option with a Premium and a strike price K. If the price of the underlying asset at maturity is below K, the payoff will be 0 and a loss equal to the premium paid. However, if the price of the underlying asset at maturity is above K, the payoff is the difference between the price of the underlying asset minus the strike price and the profit would be equal to the profit minus the premium.

In a put option applies the same logic but inversely since the buyer of the option is no longer buying an asset but selling it. Thus, the contract will be executed when the strike price is higher than the market price. The payoff of a put option is:

The Black Scholes model is a mathematical model used in quantitative finance to know the fair price of an option. Fischer Black and Myron Scholes derived in 1973 the formula for the model that was awarded with a Nobel Prize in 1997 (*Falta citar*). By replicating an option and building a replicating portfolio, the authors of the model found an equation that let us price options.

Black Scholes model equation is derived from the change of the stock price which is given by:

(3)

After the derivation of the equation by applying a Taylor expansion, they obtained a partial differential equation:

(3.1)

Then, the formula of the Black Scholes to price a European option is,

(4)

Where:

(5)

(6)

Following the Black Scholes model, is defined by a log-normal distribution. Precisely, is defined by the following formula using a geometric Brownian motion :

(7)

Where: = Price of the underlying asset at time t; = Initial price of the underlying asset; = risk-free rate; = sigma; = point in time; = maturity time.

is a Brownian motion, also known as Wiener process due to the mathematician that developed the method. It is a stochastic process with mean 0 and variance equal to 1, used in quantitative finance when modelling random asset prices. Wiener N. (*falta citar*) found out that the random process created some years ago by Bachelier L. was useful and adaptable for some finance methods.

A brownian motion has some important properties that are interesting to remark.The path created in the process is finite and continuous, there are no missing values between times, nor an infinite path. Moreover, as we have pointed out before, it follows a normal distribution.

As a direct consequence of this definition, the Brownian motion used can be defined as:

(8)

In order to simplify the model and make the derivation of the equation easier, some assumptions were made. In particular, the underlying option follows a log-normal random walk and the risk-free interest rate is constant and known for all maturities. There are no transaction costs or dividends on the underlying and no arbitrage opportunities.

# Monte Carlo Methods

Monte Carlo methods are statistical models based on repeated random variables that try to approximate the expected value of these simulations. It was in 1964 when Hammersley and Handscomb (*falta citar*) presented a very good approach to the method and later on used by Boyle (*falta citar*) to price an option.

We will use in our derivations two main mathematical theories, the laws of Large Numbers and the Central Limit Theorem.

The first one, the laws of Large Numbers, is a theorem that states that when the sample size of identically distributed randomly generated variables increases, the closer its mean gets to the expected value. In other words, if an experiment is done a large number of times, by probabilities, we should get the real value estimated as the mean value of the results. Moreover, the better the estimation method is, the smaller the variance of the result.

The Central Limit Theorem states that, in general conditions, if a sample is the sum of *n* random variables with known mean and finite, non-zero variance, then the distribution function of should take the shape of a normal, Gaussian, distribution when the sum of random, independent variables is large enough.

We want to estimate the parameter that follows a distribution:

(9)

Where is a function such that .

If we have independent random observations taken from a probability function , the appropriate estimator of our parameter is:

(10)

And the sample variance yielded is as follows:

(11)

Given the mentioned properties of (9) and following the Law of Large Numbers, the formula for the estimator can be rewritten as:

(12)

And given the Central Limit Theorem, the formula for the sample variance can be rewritten as:

(13)

This shows us that our parameter , the final option price from now on, and its variance can be approximated.

Let us now focus on the utilization of Monte Carlo simulations to price call options. This method is highly used as it gives more flexibility compared to other option pricing techniques; although it is not always the most accurate one, it allows you to implement this method in all kinds of options. Also, as stated in the previous section, we have to consider that Monte Carlo methods for option pricing assume a risk-neutral environment where options are priced. That will help us understand the use of the risk-free rate when discounting the future value (FV) to the present value (PV), or converting PV to FV.

When pricing a vanilla European option, steps to obtain a value are very clear: First, we need to simulate the final price of the asset, , many times and compute the payoff obtained every simulation. Once all the payoffs are computed, the mean of all the simulated payoffs is calculated and valued at the present value.

When pricing path dependent options, we will have to simulate the whole time path of any of its variables. The steps to follow are: First, we need to simulate a path of a Brownian Motion with random normal shocks *n* times. The price of the asset is simulated every step of the path following the asset price formula (7). A final price is obtained for every path simulated, the payoff is computed for each price, and finally, we compute the mean of all the payoffs and its present value.

In our Python codes we will use the first course of action. We will deep dive into the codes in section 5, but now let us focus on the idea behind our simulations. To price a Call Option via Monte Carlo simulations we have followed the next steps:

1. Create a Random Normal with mean 0 and variance 1. This will replicate normal shocks following the intuition behind (5).
2. Create the Brownian Motion, as stated in (8).
3. Calculate final price of the underlying asset, as stated in (7).
4. Compute the payoff using the final price calculated and value it at the present value by applying the formula (2).
5. Repeat *j* times and finally compute the mean.

In the following section, to study the goodness of this process, we took a sample *n* of Monte Carlo simulations and computed the variance and mean out of the *n* simulations.

# Variance reduction techniques

As stated in the Monte Carlo Section, one of the benefits of using this simulation method for pricing derivatives is its flexibility and accuracy, but the downside is that it involves a high computational cost which, moreover, has decreasing marginal accuracy-cost ratio if we try to increase accuracy by increasing the number of sample size *n*; indeed, to reduce variance by a factor of 0.01, the number of *n* must be increased by a factor of 100.

Variation Reduction Techniques aim to lower the variance, thus obtain a more accurate result, not by increasing the number of computations *n* but by reducing variance using different mathematical properties, thus increasing the accuracy while reducing computational complexity and timespan of calculations.

In this section we will focus on three of the six techniques explained in Computational Methods in Finance (Hirsa, A. 2013, Chapter 6.9). The three methods we will study are the Antithetic Variates Method, the Control Variate method, and the Conditional Monte Carlo. Moreover, by the end of this section, we will try to mix two of the methods by applying Antithetic Variates to the Conditional Monte Carlo. We will focus on these three methods as we believe that they are the ones who fit better the scope of our work.

## 3.1 Antithetic Variates Method

The Antithetic variates method consists in taking advantage of the mathematical properties of the correlation between two samples to reduce the desired estimator variance.

The mathematical idea behind this is, suppose we have  as we had before, and suppose we have two samples generated, and , which are both a sample of . We obtain an unbiased estimator using:

(14)

with a variance defined by

(15)

If we have that and are i.i.d than

(16)

Moreover, if we get to a situation where , when applying it in (15) there is a substantial reduction of the variance. If we get two samples with a we will obtain the minimum variance that this technique can yield.

Knowing the general theory, applying it into our simulation does not involve any big trouble. Our sample Y is the payoff obtained via the crude Monte Carlo we have generated in the previous section. In order to obtain two samples,and , we have computed , this will allow us to have a new sample with without increasing the number of computations carried out by the program.

Doing the same process than for the crude Monte Carlo, we will obtain a present value of and . Applying those values into (14) we know the value of the Antithetic estimator.

By repeating this process *n* times and computing the mean we can obtain a consistent estimator of , , with lower variance than when computed with crude Monte Carlo while keeping a relatively small number of computations.

## 3.2 Control Variate Method

Control Variate is a worldwide used technique introduced by Phelim Boyle (*falta citar*) that tries to reduce the variation of the Monte Carlo methods. By setting a random variable as a control variable, it is possible to obtain more accurate results than when applying the crude method.

The assumption made in this method is that the control variate estimator created, is unbiased and consistent. Moreover, this method requires finding a variable that has a certain degree of positive correlation with the estimator variance and a known mean.

The idea behind this method is as follows. As before, suppose we have . We can obtain an unbiased estimator using:

(17)

Where is the new estimator, which is the value obtained with (1), is a constant that we will define in the following paragraphs, is the control variable, and is the expected value of the control variable.

The variance is defined as:

(18)

The constant *c* is equal to the value that minimizes the variance of the new estimator. After some derivations, the resulting formula for the constant is:

(19)

Applying (19) into (18), we can rewrite the variance of the new estimator as:

(20)

Moreover, if we get to a situation where , when applying it in (20) there is a substantial reduction of the variance and, again, the number of computations has remained low.

To apply this method to our simulation, we have used the price of the subjacent asset as our control variable . This allowed us to have a Z such that and it did not require to increase the number of computations done. Finally, we have computed as shown in (19) and used it alongside our control variable to compute (17).

By repeating this process *n* times and computing the mean we can obtain a consistent estimator of , , with lower variance than when computed with crude Monte Carlo while keeping, once more, a relatively small number of computations.

## 3.3 Conditional Monte Carlo

So far, we have learned that the Monte Carlo model is used to approximate the expected value of a variable when simulated using some mathematical intuitions. In all the previous analysis, we have used Monte Carlo methods to simulate and estimate the final price of an option considering constant volatility, and we have compared the results yielded with Black Scholes, using it as the reference for being the nearest to the actual real value.

Trotter and Tukey (falta citar, 1956) came up with an extension of the Monte Carlo method called the Conditional Monte Carlo. This method follows the idea that variance is not constant in time anymore but rather time varying, thus will need to be updated each period.

The Conditional Monte Carlo method takes this idea from Trotter and Tukey and uses Monte Carlo methodology to simulate the expected values of the variance and apply it directly to the Black Scholes model -(4), (5), and (6)- to get an accurate expected option price.

The idea now is not just estimating the final value of the variance but rather estimating the whole path. In order to do this we have to divide the time between time 0 and maturity into *m* portions and update each one with the information of *m-1.* Following Monte Carlo methodology, the information is updated by applying a Brownian Motion with random normal shocks into the variance formula used.

The process we have followed to estimate the variance is the SABR model. This model introduced by Hagan et al. (*falta citar,* 2002) assumes stochastic volatility and allows to accurately model implied variance by capturing the whole volatility smile. It is widely used by financial institutions to model and estimate their assets and options volatility. We will not enter into more detail in this essay, more information on this topic can be found in (*citar 1 o 2 articulos/TFGs/TFMs/etc.).*

The variance function derived from SABR model is:

(21)

Where is a parameter that represents the instantaneous volatility. And is a Brownian motion such that:

(22)

Where is a random normal with mean 0 and variance 1 in time t.

Then, SABR variance is simulated for each division *j* of the total time. After that, the mean of the variance throughout the whole path is computed and it is applied into the Black Scholes formula altogether with the rest of the needed variables to derive a Black Scholes price. This process must be repeated *i* times, the mean of the *i* Black Scholes prices computed will be one Monte Carlo simulation of the price. By repeating this whole process *n* times and compute the mean of all *n* Monte Carlo simulations we obtain the Conditional Monte Carlo estimator.

## 3.4 Antithetic Conditional Monte Carlo

Last variance reduction method for Monte Carlo simulations presented in this essay is the application of the Antithetic variate technique into que Conditional Monte Carlo.

As we have seen in section 3.1, we can take advantage of the negative correlation between two samples to reduce the estimator variance. This time, the idea is doing the same -create a new Brownian motion which is the negative of the one we had already created and use it into SABR variance to compute a new sample of variances with negative covariance with the one already created-, and use formula (14) to obtain a final estimated mean variance. Apply this estimated mean variance into a Black Scholes. Repeat *j* times to get one Monte Carlo estimation of the option price. And repeat the whole process *n* times and compute the mean to obtain the Antithetic Monte Carlo estimator.

# Python code

Once knowing the theory behind Monte Carlo methods and the different variance reduction techniques applicable, we want to take all that knowledge into practice. The following section will contain the explanations and the logic behind the codes created to simulate all theory explained in previous sections.

This section’s codes, and screenshots in appendix, will be presented using variables defined as: Initial price of the underlying asset , strike price , real interest rate , volatility , and maturity time (annualized).

Before deep diving into the codes, basic notions of Python will be provided to better understand our codes. The command *def* defines a function if followed by the function name and the inputs given, e.g. “def BS()” so when we write BS(100,80,.01,.4,.1) it will return us the Black Scholes solution for the parameters given. More information has to be provided to have a fully working function; we need to program the process that Python must do to get to the Black Scholes price. Inside this process programming we can use several types of structures, the one we will use the most is *for,* a structure that creates a loop (repeats the same process over and over) the number of times indicated. *Return* is used at the end of a function to define which should be the outcome.

The first, and necessary, step needed for programming in Python is to find the most appropriated library for your analysis. We searched among the most used Python libraries to find the one that would be the most useful library we could use to carry out our simulations. As all our simulations were based on mathematical algorithms, the library Numpy is the one that fulfils our requirements better. It is ideal to develop computational analysis as it allows you to carry out mean and variance calculations out of vectors and matrices, as well as compute exponentials, square roots, and many other mathematical operations. But most importantly it allows you to create matrix and vectors. This will be really useful to compute Monte Carlo as it will allow us to store the final price, or the different t in sigma path, of each one of the n simulations carried out and therefore analyse the results obtained.

Illustration 2: Code for import Numpy Library



(Source: Own Python code)

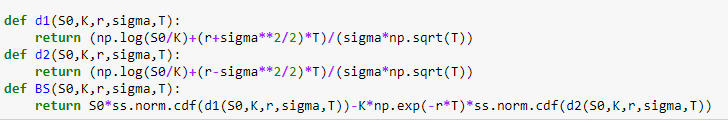
## 4.1 Black Scholes

Once with the library chosen and with the intention of following the order presented in the previous section, Black Scholes was calculated. Three functions were needed to compute the Black Scholes price following the intuition on (4), (5), and (6). These three functions are pretty straight forward to program as they do not involve any complex coding structure. The coding process was:

1. d1 is the function that, given initial variables, it computes (5).
2. d2 follows the same idea; compute (6) given initial variables.
3. Finally, the function BS uses d1 and d2 plus the initial variables to yield the Black Scholes price.

This function will appear in all codes and results screens as we will use it to compare the results obtained with Monte Carlo methods and to evaluate the accuracy of those.

Illustration 3: Code for Black Scholes computation



(Source: Own Python code)

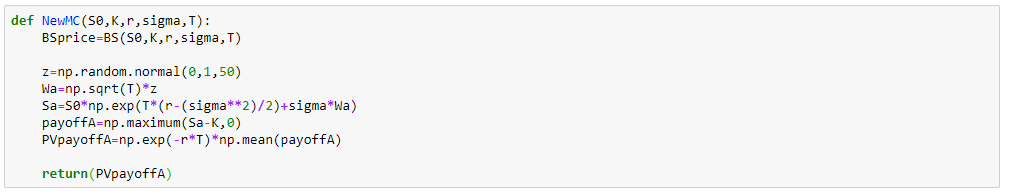
## 4.2 Crude Monte Carlo

The idea behind the code for the Crude Monte Carlo has already been mentioned in page 7, last paragraph of section 3. Let us see now how we coded it:

The code was divided into two parts, the first one is the calculation of a single Monte Carlo simulation:

1. Create , is a random normal with mean 0, variance 1, and size 50.
2. is the Brownian motion created multiplying into the square root of maturity time T as stated in formula (8).
3. Apply the computed into formula (7) to obtain a sample of 50 different values of the final price of the underlying asset. Getting a vector of .
4. Use the 50 values in sample to calculate payoff via retaining the highest, maximum number between or 0 of each one, as shown in formula (1). Getting a vector of .
5. Compute the net present value from the mean of the 50 Getting a single value which is one Monte Carlo simulation.
6. Finally, it is asked to return when the function is invoked followed by the desired variables.

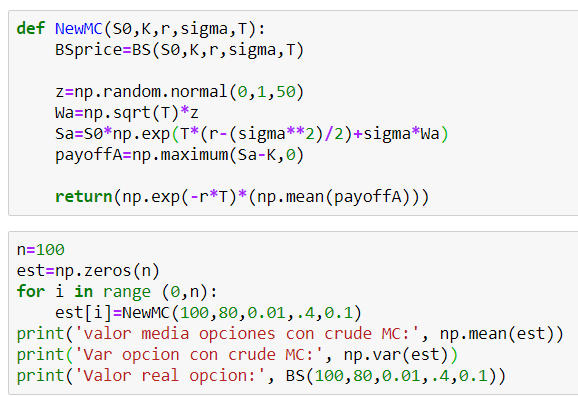
Illustration 4: 1st part of the code for Crude Monte Carlo computation



(Source: Own Python code)

The second part of the code consists of repeating the first part *n* times, thus obtaining a set of Monte Carlo simulation and compute the mean and variance to get to the final value:

1. Define *n*, number of Monte Carlo simulations.
2. Create a vector of size *n, .*
3. Use a loop *for* to compute the first part of the function -one Monte Carlo simulation- *n* times and store the value of each computation in the corresponding position into vector . This will create a vector composed by *n* Monte Carlo simulations.
4. It is asked to compute the mean of the vector , the mean of all Monte Carlo simulations, which will define the option price. Preceded by the explanatory message “Mean value of the option with Crude Monte Carlo”.
5. It is asked to compute the variance of the vector , which will define the volatility, thus the accuracy, of our computations. Preceded by the explanatory message “Variance of the option with Crude Monte Carlo”.
6. And as stated above, it is asked to show the Black Scholes price to compare. Preceded by the explanatory message “Real option price”.

Illustration 5: 2nd part of the code for Crude Monte Carlo computation 

(Source: Own Python code)

## 4.3 Antithetic Monte Carlo

If we recall from theory previously explained, the basis for Antithetic method consisted in having two samples and that if than variance would be reduced and thus we would obtain a more precise estimator of the price of the option.

In order to obtain two samples out of the Crude Monte Carlo code without increasing the computation complexity of the code, we computed taken from -, thus using the same simulations as before. This way we will not need to create another random normal and all its values for each simulation while we ensure having two samples with negative covariance. The procedure followed was the following:

The code was divided into two parts, the first one is the calculation of a single Antithetic Monte Carlo simulation:

1. A function was created with the name NewAT that requires the variables .
2. A variable was created. is a random normal with mean 0, variance 1, and size 50.
3. is the Brownian motion created multiplying into the square root of maturity time T.
4. is the Brownian motion created by taking the negative of , .
5. Apply the computed (into formula (7) to obtain a sample of 50 different values of the final price of the underlying asset. Getting a vector of ().
6. Use the 50 values in sample to calculate payoff via retaining the highest, maximum number between or 0 of each one, as shown in formula (1). Getting a vector of .
7. Compute the net present value from the mean of the 50 Getting a single value .
8. Compute the new estimator using and as shown in formula (14). This will be the Monte Carlo Antithetic Estimator of the option price.
9. Finally, it is asked to return the new Antithetic Estimator when the function is invoked followed by the desired variables.

Illustration 6: 1st part of the code for Antithetic Monte Carlo computation 

(Source: Own Python code)

The second part of the code consists of repeating the first part *n* times, thus obtaining a set of Antithetic Monte Carlo simulation and compute the mean and variance to get to the final value. This part of the process is exactly the same as for the Crude Monte Carlo so no explanation will be provided. We have added the solution for the Crude Monte Carlo and for the Black Scholes into the outcome of the code for comparison purposes. The code will be provided as illustration 7 for further revision.

Illustration 7: 2nd part of the code for Antithetic Monte Carlo computation



(Source: Own Python code)

## 4.4 Control Variate Monte Carlo

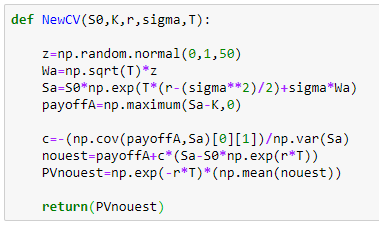
Going back to the theory explained for Control Variate method, the idea consisted in using a control variate such that in order to obtain a value and apply it into formula (17) to get the Control Variate Monte Carlo estimator.

To get a consistent control variate Z from the Crude Monte Carlo code without increasing the computational complexity, we will be using the final value of the underlying asset as control variate for two reasons 1) it is already computed and 2) it will provide for sure a positive covariance with our *Y*, the payoffs.

The code has two parts, the first one is the calculation of a single Control Variate Monte Carlo simulation, which, de facto, the 5 firsts steps are the same as for the Crude Monte Carlo, so let us begin from the 6th one on:

1. Use previously computed sample of asset prices and sample of payoffs into formula (19) to obtain the value .
2. Compute the vector of values of the new estimator using the sample of , the sample of , and into formula (17). Getting the vector of estimators *nouest.*
3. Compute the Net Present value on the mean of the vector *nouest.* Getting a single value *PVnouest* which will be the Control Variate Monte Carlo Estimator of the option price.

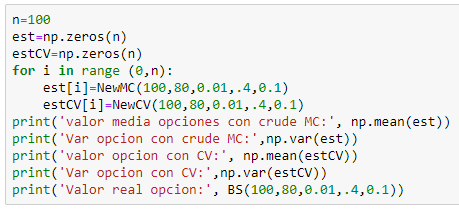
Illustration 8: 1st part of the code for Control Variate Monte Carlo computation



(Source: Own Python code)

The second part of the code consists of repeating the first part *n* times, thus obtaining a set of Control Variate Monte Carlo simulation and compute the mean and variance to get to the final value. This part of the process is exactly the same as for the Crude Monte Carlo so no explanation will be provided. We have added the solution for the Crude Monte Carlo and for the Black Scholes into the outcome of the code for comparison purposes. The code will be provided as illustration 9 for further revision.

Illustration 9: 2nd part of the code for Control Variate Monte Carlo computation



(Source: Own Python code)

## Conditional Monte Carlo

If we recall the theory on Conditional Monte Carlo, the idea was to use Monte Carlo method to estimate the average value of the volatility, sigma, rather than the final value of the price, and apply the mean value into Black Scholes pricing formula. Let us explain how to program this.

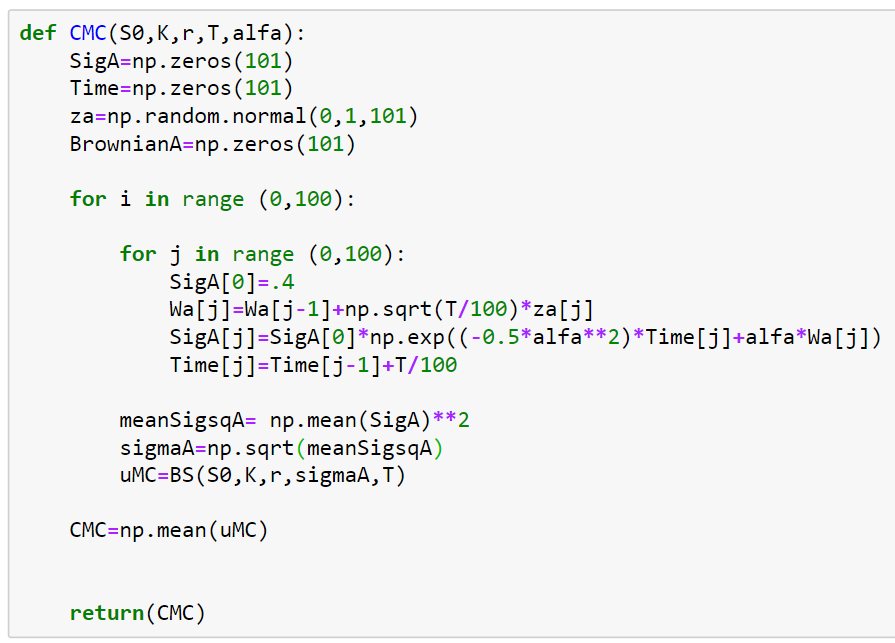
The codification is divided into two parts, the first one is used to calculate a single Monte Carlo simulation:

1. Create , is a random normal with mean 0, variance 1, and size 101.
2. Create a loop *for* that will compute *j =* 100 times (the number of time divisions we used) the following steps:

* Set initial volatility, , equal to 0.4.
* Create a Brownian motion following the formula (22) and updating it every *j.*
* Apply into SABR formula (21) and update it every *j.*
* Update time period t every *j.*

1. Compute the mean squared of all *j* and compute the square mean to have all values in absolute numbers. Getting .
2. Use the loop “*for”* to repeat 2. and 3. *i* times and computes the mean of the *i* simulations. This will give us one Monte Carlo simulation.

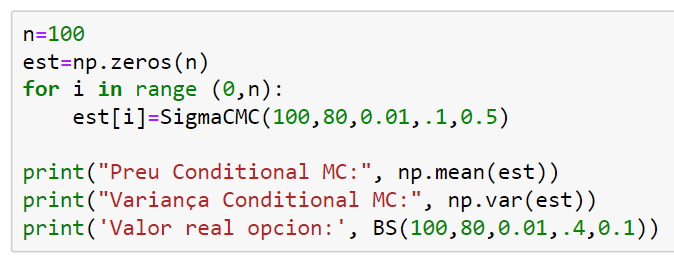
Illustration 10: 1st part of the code for Conditional Monte Carlo computation



(Source: Own Python code)

Second part of the code is used to repeat the single Monte Carlo simulation *n* times and compute the mean and variance of the estimations. This part of the process is exactly the same as for the Crude Monte Carlo so no explanation will be provided. We have added the solution for the Black Scholes model into the outcome of the code for comparison purposes. The code will be provided as illustration 11 for further revision.

Illustration 11: 2nd part of the code for Conditional Monte Carlo computation



(Source: Own Python code)

## Antithetic Conditional Monte Carlo

As seen before, this model applies the Antithetic variate into Conditional Monte Carlo. The steps followed are the ones below:

1. Create , a random normal with mean 0, variance 1, and size 101. And use it to create by applying .
2. Create a loop *for* that will compute *j =* 100 times (the number of time divisions we used) the following steps:

* Set initial volatilities, , equal to 0.4.
* Create a two Brownian motions following the formula (22) and updating it every *j.*
* Apply into SABR formula (21), one for sample *a* and one for sample *b,* and update it every *j.*
* Update time period t every *j.*

1. Compute the mean squared of all *j* of sample *a* and sample *b,* and compute the square mean to have all values in absolute numbers. Getting .
2. Apply into formula (14) to obtain the antithetic estimator.
3. Use the loop “*for”* to repeat 2. and 3. *i* times and computes the mean of the *i* simulations. This will give us one Monte Carlo simulation.

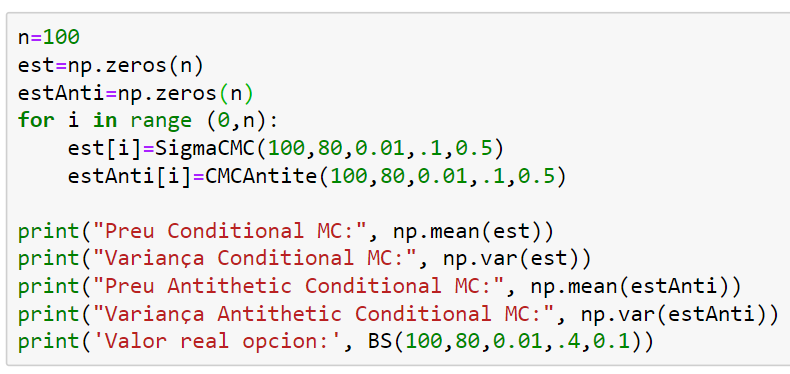
Illustration 12: 1st part of the code for Antithetic Conditional Monte Carlo computation



(Source: Own Python code)

Part two of the code is used to repeat the single Monte Carlo simulation *n* times and compute the mean and variance of the estimations. Once more, this part of the process is exactly the same as for the Crude Monte Carlo so no explanation will be provided. We have added the solution for the Black Scholes model into the outcome of the code for comparison purposes. The code will be provided as illustration 13 for further revision.

Illustration 13: 2nd part of the code for Antithetic Conditional Monte Carlo computation



(Source: Own Python code)

# Results

As stated in section 2, when explaining the main characteristics of the Black Scholes model, we have already presented why the result of this model will be used as the price of an option.

Through this section we will compare the outcome obtained from our Monte Carlo code, and it varies, by comparing the price reached with the price obtained with Black Scholes model. Not only price will be taken as a measurement of success, but also the variance obtained with each method, which indicates its accuracy.

## 5.1 Initial Variables

The first part of the computations, and screenshots in appendix, will be presented using initial variables defined as: Initial price of the underlying asset , strike price , real interest rate , volatility , and maturity time (annualized). These variables will remain fixed during this section. Will be in the next section, where we will study how changes in these variables affect the computations and thus the results obtained. The only variable that will change now will be *n*, which will be used to see how an increase in *n* affects the accuracy of our computations.

Using the initial variables, the Black Scholes price of the option is 20.25285. (Illustration X in appendix)

Starting with the Crude Monte Carlo simulation, as we increase the number of simulations, the price yielded gets closer to the Black Scholes model although the variance does not get reduced. (Illustration X in appendix)

Table 1

|  |  |  |  |
| --- | --- | --- | --- |
|  | Price | Variance | Price Delta vs. BS |
| N=100 | 19.7586 | 2.8856 | 0.4942 |
| N=1,000 | 20.1525 | 2.9389 | 0.1003 |
| N=10,000 | 20.2156 | 2.9919 | 0.0373 |

(Source: Own Python code)

With the Antithetic Monte Carlo method. We can observe a huge reduction of the variance when compared to the Crude Monte Carlo, and when the number of simulation *n* is increased*.* The same phenomena happens with the price yielded; we obtain smaller differences from the Black Scholes price, and the difference gets reduced as the number of simulations is increased. The price yielded when *n=10,000* is just 0.00025 units apart from the Black Scholes value. (Illustration X in appendix)

Table 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Price | Price diff. vs. BS | Variance | Variance diff vs. Crude MC | Variance reduction. Vs. n/10 |
| n=100 | 20.1616 | 0.09125 | 0.08218 | -2.80342 |  |
| n=1,000 | 20.2625 | 0.00965 | 0.06892 | -2.86998 | -0.01326 |
| n=10,000 | 20.2531 | 0.00025 | 0.06451 | -2.92739 | -0.00441 |

(Source: Own Python code)

Using the Control Variate method and its benefits from the positive correlation between our estimator and the variable of control, we achieved smaller, thus better, variance than with the Antithetic Monte Carlo, although the price is further away from the Black Scholes value. (Illustration X in appendix)

Table 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Price | Price diff. vs. BS | Variance | Variance diff vs. Crude MC | Variance reduction. Vs. n/10 |
| n=100 | 20.2164 | 0.0364 | 0.02221 | -2.86339 |  |
| n=1,000 | 20.2331 | 0.0197 | 0.02028 | -2.91862 | -0.00193 |
| n=10,000 | 20.2373 | 0.0155 | 0.02014 | -2.97176 | -0.00014 |

(Source: Own Python code)

Looking at the outcome from Conditional Monte Carlo simulation, in terms of price yields a better result than Antithetic Monte Carlo when *n* islow, but the increase in *n* does not improve the accuracy unlike what happened with Antithetic. The biggest improvement when compared with the previous methods is the variance outcome, which is way smaller than the one from Crude Monte Carlo and a decimal smaller than Control Variate’s one. (Illustration X in appendix)

Table 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Price | Price diff. vs. BS | Variance | Variance diff vs. Crude MC | Variance reduction. Vs. n/10 |
| n=100 | 20.2619 | 0.0091 | 0.009888 | -2.875712 |  |
| n=1,000 | 20.2596 | 0.0068 | 0.007797 | -2.931103 | -0.002091 |
| n=10,000 | 20.2562 | 0.0034 | 0.008176 | -2.983724 | 0.000379 |

(Source: Own Python code)

The last method simulated has been Conditional Monte Carlo combined with the Antithetic Variance method. This has resulted to be the best method that we have applied, at least, in terms of variance. The mean of the prices is a bit below the Black Scholes price. There is no significant change between 100 repetitions and 10.000 neither for the price nor for the variance. (Illustration X in appendix)

Table 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Mean | Price diff. vs. BS | Variance | Variance diff vs. Crude MC |
| n=100 | 20.2441 | 0.0087 | 0.0000357 | -2.8855643 |
| n=1,000 | 20.2442 | 0.0086 | 0.0000512 | -2.9388488 |
| n=10,000 | 20.2437 | 0.0091 | 0.0000438 | -2.9918562 |

(Source: Own Python code)

## 5.2 Effects of Changes on Initial Variables

In this section of the thesis, we would like to use the models presented to observe the effects of changing initial variables on price and variance yielded. To do so, we will fix the number of simulations *n =* 1,000.

Let us first recall the results with the initial variables , , , , :

|  |  |  |
| --- | --- | --- |
|  | Price | Variance |
| Black Scholes | 20.25285 | - |
| Crude Monte Carlo | 20.1525 | 2.9389 |
| Antithetic Monte Carlo | 20.2625 | 0.06892 |
| Control Variate Monte Carlo | 20.2331 | 0.02028 |
| Conditional Monte Carlo | 20.2596 | 0.007797 |
| Antithetic Conditional Monte Carlo | 20.2442 | 0.00865 |

Let us see the changes on increasing and decreasing in 10 units, *ceteris paribus*:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 90 | | 110 | | |
|  | Price | Variance | Price | Variance |
| Black Scholes | 11.0811 | - | 30.1018 | - |
| Crude Monte Carlo | 11.1055 | 2.1925 | 30.0322 | 3.9530 |
| Antithetic Monte Carlo | 11.0701 | 0.1539 | 30.1090 | 0.0369 |
| Control Variate Monte Carlo | 11.0297 | 0.1046 | 30.0964 | 0.0033 |
| Conditional Monte Carlo | 11.0588 | 0.0661 | 30.1056 | 0.0004 |
| Antithetic Conditional Monte Carlo | 11.0537 | 0.0003 | 30.1001 | 0.000002 |

Values for a change of 10 units in , *ceteris paribus,* are:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | K=70 | | K=90 | | |
|  | Price | Variance | Price | Variance |
| Black Scholes | 30,0772 | - | 11,4291 | - |
| Crude Monte Carlo | 30,1625 | 3,0812 | 11,3701 | 2,3806 |
| Antithetic Monte Carlo | 30,0786 | 0,0277 | 11,4367 | 0,2293 |
| Control Variate Monte Carlo | 30,0760 | 0,0022 | 11,3513 | 0,1586 |
| Conditional Monte Carlo | 30,0789 | 0,0001 | 11,4190 | 0,0890 |
| Antithetic Conditional Monte Carlo | 30,0765 | 3e-07 | 11,3973 | 0,0006 |

We see that an increase (decrease) in 10 units on the initial value of the underlying asset changes approximately in 10 units the value of the option in the same direction. The opposite happens when changing the strike price, when increasing (decreasing) its value by 10 units, decreases (increases) the price of the option by, approximately, 10 units.

This is due to the effect of proximity (farness) of to *K*; the nearest (furthest) the initial value is to the strike price, the easiest (more difficult) that the underlying asset price in maturity time is below (above) it, and thus the less (more) chances to execute the option, price of the option changes accordingly.

On the other hand, the variance is augmented (reduced) when the value of the underlying asset decreases (increases) or the strike price augments (decreases). The reasoning behind this is the same one as before, there are less changes to execute the option when underlying price is nearest to the strike price*,* thus decreasing the variance given that for the payoff function yields the same price with than with .

Let us study now a change in volatility *, ceteris paribus.* Conditional Monte Carlo does not apply as sigma is indeed computed by the method*:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | σ=0.2 | | σ=0.8 | |
|  | Price | Variance | Price | Variance |
| Black Scholes | 20.0802 | - | 22.3988 | - |
| Crude Monte Carlo | 20.0714 | 0.7897 | 22.3366 | 9.9576 |
| Antithetic Monte Carlo | 20.0017 | 0.0017 | 22.3700 | 1.4336 |
| Control Variate Monte Carlo | 20.0800 | 0.0003 | 22.2672 | 0.4521 |

When σ decreases the price of the option also decreases, the same happens the other way around but with a stronger effect. As the volatility increases, the chances to obtain a high payoff also increase, thus the option increases in price, and vice-versa.

Variance, logically, moves in the same direction as the changes in sigma. Again, the effect of increasing the sigma is higher than the one when decreasing it.

Finally let us study how changes in maturity time *T,* *ceteris paribus,* affect our results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *T* = *0.01* | | *T* = 0.2 | |
|  | Price | Variance | Price | Variance |
| Black Scholes | 20.0080 | - | 20.9518 | - |
| Crude Monte Carlo | 19.9832 | 0.3170 | 21.0312 | 5.4158 |
| Antithetic Monte Carlo | 20.0005 | 0.0002 | 20.8278 | 0.3526 |
| Control Variate Monte Carlo | 20.0085 | 0.0001 | 20.8998 | 0.1250 |
| Conditional Monte Carlo | 20.0079 | 1.03e-16 | 20.9647 | 0.1427 |
| Antithetic Conditional Monte Carlo | 20.0079 | 1.54e-20 | 20.9249 | 0.0018 |

When *T* decreases, the estimation of the price gets a lot more accurate and the variance is reduced drastically. The reason is that, by reducing the lifespan of the option, taking its maturity time nearer to the present, is easier to predict how it will end. Contrary, taking the maturity time further from the present time, prediction span is longer and thus yields uneven prices and higher variance.

Changes in real interest rate have been studied but no relevant differences in price and volatility have been observed; when *r* increases, the price of the option increases and vice-versa, but as *r* tends to have low fluctuation the changes in price are not remarkable. Volatility does not change.

**Bibliography and Webliography**

1. Boyle, P. (1976) “Options: A Monte Carlo Approach North-Holland Publishing Company
2. Brereton, T. (2015) “Methods of Monte Carlo Simulation” Ulm University
3. Hagan, P. (2002) “Managing Smile Risk” pp. 84-108 3rd. Wilmott Magazine
4. Hammersley, J. M. and Handscomb, D. C. (1964) “Monte Carlo Methods” 1st ed. Fletcher & Son Lid
5. Hirsa, A. (2012) "Computational methods in finance" ch. 6.9 pp. 240-249 (& others) 1st ed. Boca Raton CrC Press
6. Hull, J. (2018) “Options, futures and other derivatives” pp. 343-376 9th ed. Pearson
7. Lidebrandt, T. (2007) “Variance Reduction Three Approaches to Control Variate” Stockholm University
8. Merton, R. (1973) “Theory of Rational Option Pricing”. pp.141-183 1st ed. The Bell Journal of Economics and Management Science
9. Mohammed, S. (2014) “Pricing Options Using Monte Carlo Methods” M.Sc. Thesis, Chennai Mathematical Institute
10. Trotter, H. & Tukey, J. (1956) “Conditional Monte Carlo for normal samples” pp. 64-79, Princeton University, ed. H. A. Meyer
11. Wiener, N. (1976) “Collected Works vol.1” 1st ed. [P. Masani](https://mitpress.mit.edu/contributors/p-masani)
12. Wilmott, P. (2013) “Quantitative Finance” vol.3, 2nd ed. Wiley
13. Bogatyreva N. Grandez R. Rodriguez S. and Soldevila S. (2019). “SABR: A Stochastic Volatility Model in Practice”