Monte Carlo Methods and Variance Reduction Techniques on Floating Asian Options

Joan Antoni Seguí Serra

Advisor: Elisa Alòs Alcalde

Project Code: EMC16

Academic Year: 2018/2019

Abstract

In this work, Monte Carlo simulations coded in Python are used to estimate short-term floating Asian options. Afterwards, variance reduction techniques are used on the Monte Carlo simulations to reduce their variance and to compare those estimates to the first and second approximation formulas by Alòs and León[1]. Finally, an analysis of the volatility of each technique and the errors of both approximation formulas is performed.

Keywords: exotic options; floating Asian options; Monte Carlo simulations; variance reduction techniques

Contents

1 Introduction 2

2 Financial Derivatives. Standard and Exotic Options 2

2.1 European Options 5

2.2 Fixed Strike and Floating Strike Asian Options 6

2.3 Approximation Formulas for Floating Strike Asian Options 7

3 The need of Monte Carlo Methods in Finance 9

4 Variance Reduction Techniques 11

4.1 Antithetic Variates Method 11

4.2 Control Variate Method 12

5 Python Code 14

6 Results 16

7 Conclusions 23

References 24

8 Appendix 25

1 Introduction

In the first section, financial derivatives will be introduced, by focusing on options. Standard and exotic options will be distinguished, and a more extensive explanation of European options and Asian options will be made. Asian options will be explained in order to follow the reasoning of the paper. European options will be explained for two reasons. Firstly, these options are the most basic ones, which make them easy to understand, to calculate and to simulate. Secondly, European options will be the control variate that will reduce the variance of the simulated floating Asian options.

After explaining the fundamentals of the options, Monte Carlo simulations will be explained as well as the reason why they are needed in finance. In the following section it will be explained how the variance of the simulations can be reduced by using two variance reduction techniques: the control variate method and the antithetic method. Once explained the theory, it will be explained the code in python in the following section.

Finally, an analysis and a conclusion of the results will be carried out. It will be discussed which method of variance reduction is more effective. The two approximation formulas will be benchmarked against the simulations, and how they perform in each case will be analyzed in order to see if one is more accurate than the other one.

2 Financial Derivatives. Standard and Exotic Options

A financial derivative is an instrument whose value depends on one or more underlying assets, including stocks, bonds, interest rates, currencies, or more complex entities such as weather conditions. The main financial derivatives are forwards, futures, swaps and options. Focusing on options, there are call options and put options.

Call options are contracts that give the buyer the right to buy the underlying asset at a specified strike price until a certain date or at a certain date. The seller has the

obligation to fulfill the contract. Thus, options normally will be exercised when the strike price is below the spot price, or market price, of the underlying asset. On the other hand, put options are contracts that give the buyer the right to sell the underlying asset.

The most common option styles are American and European options,and they are known as standard options. The rest of option styles are known as exotic options. The difference between an European and an American option is the time when they can be exercised, until a certain date or at a certain date. If the option can only be exercised on the expiration day (or maturity), it is an European option. If the option can also be exercised on any trading day before the expiration, it is an American option. The requisites to be a standard option are the following: to be formed by only one underlying asset, the starting time is the present and, finally, the price of the underlying at maturity is the only factor that affects the payoff of the option. Thus, the payoff is always the difference between the strike price and the underlying asset spot price.The payoff of an exotic option depends on some function of the price of the underlying asset.

The strong point of exotic options is flexibility. Exotic options can be customized and created to fulfill different needs. For example, if an investor is only interested in the difference between two stocks, he or she can invest in spread options, as the underlying of this option is the difference between the prices of two assets. A binary option only pays a fixed amount only when triggered by some event, such as a stock reaching some price level. A chooser option allows an investor to invest in changes in volatility.

Nelken[2] classified exotic options in the following way,

- Path-Dependent. These options depend on the price path of some underlying asset and not only on the price at a specific date.
- Singular Payoffs. These options do not follow a continuous payoff. Instead, the payoffs take jumps discontinuously.
- Time-Dependent or Preference. These options have some different preference, or

dependence, on a certain time.

- Multivariate. These options consist of two or more underlying assets instead of a single one.
- Leveraged. The payoffs of this options do not follow a linear function of the price of the underlying.

An Asian option, which is an exotic option, is an option whose payoff, instead of being determined by the spot price, depends on the average price of the underlying asset over some present time period. The price does not depend only on the price of the underlying asset at maturity, but on the average price. Thus, an Asian is a path-depending exotic option, meaning that its settlement price or its strike is formed by the aggregation of the underlying asset prices during some present period. Being a path-depending option implies that the simulations will be harder than non path-depending options as, for example, European options. For European options, simulations for the last period price S_T are sufficient, but for the Asian it is necessary to simulate all the path of the stock, from S_0 to S_T.

As Ton Vorst explained in Nelken[2], one of the first companies that used Asian options was the Dutch company Oranje Nassau, in 1985. The company issued bonds with a repurchase option without using a fixed price, but with an average price of a few barrels of Brent Blend oil over the last year of the contract. The worry for the oil price manipulation impulsed the company to use Asian options. Note that manipulating an average price is harder than manipulating a specific daily price. Besides manipulation concerns, using Asian options is useful in different scenarios. Some examples are found when an importing/exporting company is worried about the average exchange rate over time, or when the market for the underlying asset is highly volatile and it is desirable to trade this asset with a lower volatility.

2.1 European Options

As stated before, the European call will only be exercised if the strike price is below the spot price. Mathematically, an European call option is

$$c = max(S_T - K, 0) \tag{1}$$

where c is the price of the call, S_T is the spot price of the underlying at time T, and K is the strike price.

And the European put is

$$p = max(K - S_T, 0) \tag{2}$$

It is assumed the classical Black-Scholes model. So,S_T follows a log-normal distribution continuous in time. In particular, this specific geometric brownian motion,

$$S_{t+\Delta t} = S_t \mathrm{e}^{(r-\frac{1}{2}\sigma^2)\Delta_t + \sigma\varepsilon\sqrt{\Delta t}} \tag{3}$$

Where $\varepsilon \sim N(0,1)$ and the lattice or the space between intervals Δt is equal to T/N.

Under these assumptions, it can be analytically derived[3] a closed-formula to price an European option without dividends,

$$\begin{aligned} c &= \mathrm{e}^{-rT}\left(S_0 N(d_+) - K N(d_-)\right) \\ p &= \mathrm{e}^{-rT}\left(K N(-d_-) - S_0 N(-d_+)\right) \\ d_+ &= \frac{ln(S_0/K)}{\sigma\sqrt{T-t}} + \frac{\sigma}{2}\sqrt{T-t} \\ d_- &= \frac{ln(S_0/K)}{\sigma\sqrt{T-t}} - \frac{\sigma}{2}\sqrt{T-t} \end{aligned} \tag{4}$$

where K is the strike price, σ is the volatility, N is the distribution function of a standard normal, S_0 is the stock price at $t = 0$, and T is the time to maturity.

2.2 Fixed Strike and Floating Strike Asian Options

There are two styles of Asian Options. A fixed strike Asian option is similar to an European option. However, instead of S_T, it uses the average price over a certain present time $A(0,T)$. Mathematically,

Fixed Asian Option Call

$$c = max(A(0,T) - K, 0) \tag{5}$$

Fixed Asian Option Put

$$p = max(K - A(0,T), 0) \tag{6}$$

A floating strike Asian option uses S_T like an European one, but instead of having a fixed strike price K, uses $kA(0,T)$ as the strike price.

Floating Asian Option Call

$$c = max(S_T - kA(0,T), 0) \tag{7}$$

Floating Asian Option Put

$$p = max(kA(0,T) - S_T), 0) \tag{8}$$

The average price of a stock can be computed in different ways.

Arithmetic Average in a discrete case

$$A(0,T) = \frac{1}{n}\sum_{i=0}^{n-1} S_{t_i} \tag{9}$$

Arithmetic Average in a continuous case

$$A(0,T) = \frac{1}{T}\int_0^T S_t dt \tag{10}$$

Geometric Average in a continuous case

$$A(0,T) = \exp\left(\frac{1}{T}\int_0^T ln(S_t)dt\right) \tag{11}$$

It is extremely important to note that it cannot be analytically derived a closed-form expression for the price of an Asian option in all of the cases. One of those cases where it does not exist a closed-form expression is the arithmetic average cases, which are the cases covered in this work. The reason why a closed formula can or cannot exist is the following. As stated before, the underlying asset follows a log-normal distribution. The product of log-normal distributions is also a log-normal distribution, implying that it is possible to analytically derive a solution. However, this is not the case for the sum of log-normal distributions. The closed-form for the geometric case[3],

$$\begin{aligned} c &= \mathrm{e}^{-rT}\left(M_1 N(d_+) - kN(d_-)\right) \\ p &= \mathrm{e}^{-rT}\left(kN(-d_-) - M_1 N(-d_+)\right) \\ d_+ &= \frac{ln(M_1/k)}{S\sqrt{T-t}} + \frac{S}{2}\sqrt{T-t} \\ d_- &= \frac{ln(1_1/k)}{S\sqrt{T-t}} - \frac{S}{2}\sqrt{T-t} \end{aligned} \tag{12}$$

where,

$M_1 = \frac{\mathrm{e}^{rT}-1}{rT}S_0$

$M_2 = \frac{2\mathrm{e}^{\left(2r+\sigma^2\right)T}S_0^2}{(r+\sigma^2)(2r+\sigma^2)T^2} + \frac{2S_0^2}{rT^2}\left(\frac{1}{2r+\sigma^2} - \frac{\mathrm{e}^{rT}}{r+\sigma^2}\right)$

$S = \sqrt{\frac{1}{T}ln\left(\frac{M_1}{M_2}\right)}$

2.3 Approximation Formulas for Floating Strike Asian Options

Alòs and León[1] found two approximation formulas for floating strike Asian options. Both approximations will be compared with Monte-Carlo simulations in order to study the gain in accuracy in the second approximation formula. For simplicity, the interest rate is assumed to be zero, $r = 0$. The arithmetic continuous case is used for computing the average price of the stock. Concretely,

$$A(0,T) = \frac{1}{T}\int_0^T S_t dt \tag{13}$$

The first approximation formula is

$$E(BS(0, X_0, M_0^T, \frac{\sigma}{\sqrt{3}})) \tag{14}$$

Where $M_t^T = E[A_T|F_t^w] = k\frac{1}{T}(S_t(T-t) + \int_0^t S_u du$
$= kS_tF(T,t) + k\frac{1}{T}\int_0^t S_u du$
and $F(T,t) = \frac{T-t}{T}$

In this paper $t = 0$ is used, meaning that options are computed from now to T. So, $M_t^T = kS_t$ as $F(t,t) = 1$ and $k\frac{1}{T}\int_0^t S_u du = 0$

This means that is the same as (4) but replacing σ with $\frac{\sigma}{\sqrt{3}}$ and K with kS_t.

$$\begin{aligned} c &= (S_0N(d_+) - kS_tN(d_-)) \\ p &= (kS_tN(-d_-) - S_0N(-d_+)) \\ d_+ &= \frac{ln(S_0/kS_t)}{\frac{\sigma}{\sqrt{3}}\sqrt{T-t}} + \frac{\frac{\sigma}{\sqrt{3}}}{2}\sqrt{T-t} \\ d_- &= \frac{ln(S_0/kS_t)}{\frac{\sigma}{\sqrt{3}}\sqrt{T-t}} - \frac{\frac{\sigma}{\sqrt{3}}}{2}\sqrt{T-t} \end{aligned} \tag{15}$$

The second approximation formula is

$$E(BS(0, X_0, M_0^T, \hat{I}_0(k))) \tag{16}$$

Where $\hat{I}_0(k) = \frac{\sigma}{\sqrt{3}} - \frac{\sqrt{3}}{40}(k-1)$

Just as in the previous case, is the same equation as (4) but replacing σ with $\frac{\sigma}{\sqrt{3}} - \frac{\sqrt{3}}{40}(k-1)$ and K with kS_t.

$$c = (S_0 N(d_+) - kS_t N(d_-))$$

$$p = (kS_t N(-d_-) - S_0 N(-d_+))$$

$$d_+ = \frac{ln(S_0/kS_t)}{\frac{\sigma}{\sqrt{3}} - \frac{\sqrt{3}}{40}(k-1)\sqrt{T-t}} + \frac{\frac{\sigma}{\sqrt{3}} - \frac{\sqrt{3}}{40}(k-1)}{2}\sqrt{T-t} \tag{17}$$

$$d_- = \frac{ln(S_0/kS_t)}{\frac{\sigma}{\sqrt{3}} - \frac{\sqrt{3}}{40}(k-1)\sqrt{T-t}} - \frac{\frac{\sigma}{\sqrt{3}} - \frac{\sqrt{3}}{40}(k-1)}{2}\sqrt{T-t}$$

3 The need of Monte Carlo Methods in Finance

Boyle[4] was the first study to use Monte Carlo methods to price an option. Let's describe the general Monte Carlo approach.

Let θ be the parameter to be estimated following some distribution.

$$\theta = E(f(X)) \tag{18}$$

$f(X)$ is a function with the property that $E(g(X)^2) \to \infty$. Generating n independent random observations $X_1, X_2, X_3, ...X_n$ from some probability function $p(X)$, the estimator of θ is given by

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} g(X_i) \tag{19}$$

Given the property $E(g(X)^2) \to \infty$ and by the law of large numbers, as $n \to \infty$,

$$\frac{1}{n}\sum_{i=1}^{n} g(X_i) \to E(g(X)) \tag{20}$$

The sample variance is

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(g(X_i) - \hat{\theta}\right)^2 \tag{21}$$

Due to central limit theorem, as $n \to \infty$,

$$\frac{\hat{\theta} - \theta}{s/\sqrt{n}} \to N(0,1) \tag{22}$$

$\hat{\theta} - \theta$ is approximately a standard normal variable, with the scale $s/\sqrt{n}$. For a large n, an estimation of a confidence interval of the 95% for θ,

$$CI = \left(\hat{\theta} - 1.96\frac{s}{\sqrt{n}}, \hat{\theta} + 1.96\frac{s}{\sqrt{n}}\right) \tag{23}$$

Hence, θ and its variance can be approximated. The main inconvenient of using Monte Carlo methods is its computational cost. To reduce the standard error $s/\sqrt{n}$ by a factor of 0.1, it is necessary to increase n by a factor of 100. Much more computations are required to increase the accuracy. Indeed, as it will be seen in the next section, variance reduction techniques can be used to increase the accuracy without having to increase the sample size n. Instead, s can be reduced.

How the Monte Carlo simulations have been used and coded in python will be explained in detail in section 4. Briefly, Monte Carlo simulations to price a fixed strike Asian call work as follows:

1. Generate a vector V of n normal standard random numbers.
2. Simulate n paths of a stock plugin each generated random number in the previous vector in the equation(3).
3. Compute the arithmetic mean of the n daily prices of the stock.
4. Compute the payoff of a fixed strike Asian call for this simulation. That is, $max(A(0,T) - K, 0)$. Store this payoff in a vector.

5. Repeat the four previous steps 10,000,000 times. A vector with 10,000,000 payoffs will be created.
6. Compute the mean and the variance of the vector. This will give the simulated price of a fixed Asian call an its standard deviation.

It is used $n = 252$, which are the trading days within in a year. Note that the approximation formulas use an arithmetic mean in a continuous case, and the Monte Carlo simulation calculates an arithmetic mean in a discrete case. A computer can represent a continuous variable only with a finite number of evaluations. The error that arises using a discrete arithmetic mean to estimate a continuous one is known as discretization error. One way to minimize this error is to reduce the distance between the points. In this case, a year is divided in 252 days instead of the number of weeks or months. As for reducing the variance, the computational cost for reducing the discretization error also increases. For example, if the stock path consisted of 5 more days, $5 * 10,000,000$ extra computations are needed.

4 Variance Reduction Techniques

As stated above, Monte Carlo simulations are costly, because reducing the variance by increasing the sample size requires more computations. Instead of increasing n to increase the accuracy of the simulations, s can be reduced, thus diminishing the standard deviation $s/\sqrt{n}$. Two of the variance reduction techniques used by Hirsa [5] are the antithetic variates method and the control variate method.

4.1 Antithetic Variates Method

Let θ be estimated through Y, that is $\theta = E(h(x)) = E(Y)$. Let Y_1 and Y_2 be two samples of Y, so the estimate is $\hat{\theta} = \frac{Y_1+Y_2}{2}$. The variance of the estimate is

$$Var(\hat{\theta}) = \frac{Var(Y_1) + Var(Y_2) + 2Cov(Y_1, Y_2)}{4} \tag{24}$$

If Y_1 and Y_2 are *i.i.d*, $Var(\hat{\theta}) = \frac{Var(Y)}{2}$. And if $Cov(Y_1, Y_2) < 0$ the variance is reduced even more. In the extreme case where $\rho = -1$, the variance is reduced to the maximum that that technique allows.

Recall that in equation(4) the path of the stock was defined as a geometric brownian motion with random normal shocks. Later, it was briefly described how a Monte Carlo simulation works. In the second step of that explanation, a previous generated vector V of normal standard random numbers was used. In the fourth step the payoff of the Asian option was calculated thanks to the generated vector V. Instead, two different vectors, V and $-V$, can be used to compute that payoff. The payoff of the Asian call is done by computing one payoff P_1 using V, another payoff P_2 using $-V$, and taking the mean of the two. Note that P_1 and P_2 are Y_1 and Y_2 described before. Also note that V and $-V$ have a correlation of -1. As a result, the variance of the Monte Carlo is reduced at the maximum that the technique can achieve.

4.2 Control Variate Method

Let $\theta = E(h(X))$. Let Y be our estimator. Let Z be correlated with the random variable and let $E(Z)$ to be known. It can be constructed an unbiased estimator,

$$\hat{\theta}_c = Y + c(Z - E(Z)) \tag{25}$$

Thus, the expectation is

$$E(\hat{\theta}_c) = E(Y) = \theta \tag{26}$$

And the variance,

$$Var(\hat{\theta}_c) = Var(\hat{\theta}) + c^2 Var(Z) + 2cCov(Y, Z) \tag{27}$$

To minimize the variance, it should be derived the variance over c and find the c^* which minimizes the variance.

$$\begin{aligned} \frac{\partial Var(\hat{\theta}_c)}{\partial c} = 2cVar(Z) + 2Cov(Y,Z) = 0 \\ c^* = -\frac{Cov(Y,Z)}{Var(Z)} \end{aligned} \tag{28}$$

Now, plugging c^* in (27),

$$\begin{aligned} Var(\hat{\theta}_{c^*}) &= Var(\hat{\theta}) + \left(-\frac{Cov(Y,Z)}{Var(Z)}\right)^2 Var(Z) + 2\left(-\frac{Cov(Y,Z)}{Var(Z)}\right) Cov(Y,Z) \\ &= Var(\hat{\theta}) + \frac{Cov(Y,Z)^2}{Var(Z)} - 2\frac{Cov(Y,Z)^2}{Var(Z)} \\ &= Var(\hat{\theta}) - \frac{Cov(Y,Z)^2}{Var(Z)} \end{aligned} \tag{29}$$

Thus, if $Cov(Y,Z) \neq 0$, $Var(\hat{\theta}_{c^*}) < Var(\hat{\theta})$

In this work, θ will be a floating strike asian call, Y the simulation of the floating strike asian call, and Z an European call. The price of an Asian call under the second approximation formula(17) is used as a strike price of the European call. So, both Y the floating strike asian option and Z an European call have to be simulated. And $E(Z)$ is known because it exists a closed formula to price an European call. That equation to price an European call without the need of simulating is equation (4). To find c has to be computed the covariance between our simulations of the Asian and the European. Having c, a more unbiased accurate estimator for an Asian call can be calculated using equation (25).

Both methods are going to be used separately and jointly. It is possible to use the control variate method to a variable that has been computed via antithetic variates. The three methods will be compared.

5 Python Code

In this section the logic of the code is shown. Nevertheless, the full code can be found in the appendix.

The first step is to import the libraries needed to do the computations. Some mathematical and statistics libraries have to be imported in order to compute basics operations, such as square roots, exponential, covariances, means, standard deviations... and to be able to operate with arrays. They work like a vector in R. The parameters that will not change are defined: the number of simulations $M = 10000000$, the standard deviation $s = 0.5$, the initial price of the stock $S_0 = 100$, and the number of periods in each path of the stock price $N = 252$.

The function def is used to define a function in python, followed by a name and what will be the inputs. Inside the function there is the code of what has to be computed, and what python has to return as a result. The functions have to be defined for the first approximation formula(15), the second approximation formula(15), an European call(1), the closed formula for an European call(4) and, the two floating Asian options(7) to compute the antithetic variate.

Control variate and antithetic methods are used to reduce the variance. Recall equation(25), the antithetic method has three parameters: a simulated floating Asian call Y, a simulated European call Z and its expectation $E(Z)$. For the first method, the control variate, it is necessary to define an European call as in equation(1) to get Z from the simulations of S_T, and its closed-form equation(4). Both have an strike price of the second approximation formula for the floating Asian call, which has been previously defined. So, for the antithetic variable method two simulated floating Asian calls(7) are needed.

Three periods are used to evaluate the results, $T = 0.1$, $T = 0.5$ and $T = 1$. And for every period five differents k: $k = 0.9$, $k = 0.95$, $k = 1$, $k = 1.05$ and $k = 1.1$. It is necessary to create an empty vector for every case for every simulated variable, which

is done with $emptyvector = []$. The variables were two floating Asians calls, and an European call. Fifteen empty vectors have to be created, one for each combination.

Each simulation needs to simulate a stock price path. To do this, a loop inside another loop is needed. The first loop goes from 1 to $M = 10000000$ and sets $S_0 = 0$ and creates the two random vector V and $-V$ needed for the antithetic variables. Just after creating these variables the second loop is triggered, which simulates the two stock paths following equation(3) needed for the antithetic variables. When the second loop finishes to simulate the first two stock paths, it will append to the empty vectors the new values. The last lines of the first loop save the results of each Monte Carlo simulations, for each k of both the two floating Asian calls and the European call. To save the values in a vector, the *append* function is used, which appends a value to the vector as the last element. So, every time a number is appended, the dimension of the vector grows by one. The mean of the two vectors of the floating Asian calls creates the vector of the antithetic results for each k.

The control variate method is used in two cases, the first one with a raw floating Asian and the second one with the antithetic floating Asian, both controlled with the Euro call. First, it is computed the raw floating Asian case. c is computed by equation (28). Having each c for every k case, the estimated is computed using c in equation (25). This will return the vector of the floating Asian calls controlled for the European call.

After that, is computed c for the second case in which the antithetic floating Asian call is controlled. Next, c is plugged in equation (25) as before, to get the vector of the controlled floating Asian calls.

Having all the vectors: including the vectors of the raw Asians, the vectors of the controlled floating Asians and the vectors of the controlled antithetic Asians, each parameter can be estimated by using the equation (19). This is, computing the mean of each vector and its standard deviations.

Now with all the results, the data frame of the results has to be created. The

data frame consists of the first and second approximation formulas for each k, and the estimations and standard deviations for each k of the four different methods: floating Asian, controlled floating Asian, antithetic Asian and, controlled antithetic Asian. A data frame in python is done in the following way: after naming our data frame we have to name our vectors (the columns of the data frame) and assign the values for each vector.

The last step is to compute the errors of each estimation with the two approximation formulas. The errors of both approximation formulas are computed against the controlled floating Asian, the antithetic floating Asian, and the controlled antithetic floating Asian.

6 Results

The first three tables show the results for the two approximation formulas, the raw Monte Carlo simulations, the controlled Monte Carlo simulations, the antithetic Monte Carlo simulations, and the controlled antithetic Monte Carlo for the floating Asian calls.

Table 1: Comparative of 1st and 2nd order approximations with Monte Carlo simulations for T=0,1.

T=0,1	1st Order	2st Order	MC	Control	Antithetic	Control antithetic
k=,9	10,53297	10,55983	10,54165	10,54636	10,54585	10,54598
k=,95	6,59506	6,61780	6,59760	6,60143	6,60090	6,60106
k=1	3,64056	3,64056	3,62676	3,62947	3,62909	3,62926
k=1,05	1,75173	1,72752	1,72418	1,72582	1,72548	1,72562
k=1,1	0,73247	0,69956	0,70346	0,70429	0,70411	0,70420

Table 2: Comparative of 1st and 2nd order approximations with Monte Carlo simulations for T=0,5.

T=0,5	1st Order	2st Order	MC	Control	Antithetic	Control antithetic
k=,9	13,72466	13,82575	13,77002	13,76933	13,76327	13,76319
k=,95	10,67403	10,73142	10,68352	10,68292	10,67732	10,67724
k=1	8,12926	8,12926	8,09733	8,09682	8,09163	8,09156
k=1,05	6,06851	6,00801	5,99651	5,99609	5,99174	5,99167
k=1,1	4,44562	4,33151	4,34125	4,34091	4,33711	4,33705

Table 3: Comparative of 1st and 2nd order approximations with Monte Carlo simulations for T=1.

T=1	1st Order	2st Order	MC	Control	Antithetic	Control antithetic
k=,9	16,61521	16,76707	16,67147	16,66937	16,67019	16,66979
k=,95	13,86460	13,94662	13,86591	13,86401	13,86512	13,86473
k=1	11,47661	11,47661	11,41905	11,41735	11,41858	11,41822
k=1,05	9,42994	9,34360	9,31463	9,31312	9,31434	9,31400
k=1,1	7,69622	7,52654	7,52893	7,52762	7,52871	7,52840

The Monte Carlo approximation with the lowest standard deviations will be chosen as the benchmark for the first and second order approximation formulas, as the accuracy is provided by having a lower volatility.

The sum of standard deviations will be used to choose the best performer in case there is not a Monte Carlo method with lower standard deviation for all k.

Table 4: Comparative of Monte Carlo simulations standard errors for T=0,1.

T=0,1	Std MC	Std MC control	Std MC antithetic	Std MC control antithetic
k=,9	0,00281	0,001253	0,00074	0,000737
k=,95	0,002396	0,001251	0,000958	0,000954
k=1	0,001857	0,00116	0,001034	0,00103
k=1,05	0,001296	0,000956	0,000832	0,000829
k=1,1	0,000816	0,000686	0,000556	0,000554
Sum	0,009175	0,005306	0,00412	0,004104

For $T = 0.1$, the antithetic method lowers the standard deviation more than only using the control variate method for every k. Using both, the antithetic and the control, the lowest standard deviation is achieved. However, the difference between using both or using only the antithetic is very low. The method performing the best in terms of low volatility is the control antithetic, followed by the antithetic, and, in the last place, the control. For $T = 0.1$ the control antithetic will be the benchmark as it is the most accurate technique in this case.

Table 5: Comparative of Monte Carlo simulations standard errors for T=0,5.

T=0,5	Std MC	Std MC control	Std MC antithetic	Std MC control antithetic
k=,9	0,00597	0,002756	0,002945	0,002876
k=,95	0,005379	0,002768	0,00296	0,002895
k=1	0,004765	0,002718	0,002841	0,00278
k=1,05	0,004151	0,002602	0,00261	0,002555
k=1,1	0,00356	0,002429	0,002321	0,002274
Sum	0,023825	0,013273	0,013677	0,01338

For $T = 0.5$, using only the control technique is the best option in terms of accuracy for $k \leq 1$, but not for $k > 1$. Since its sum of deviations is the lowest, the control

technique will be the benchmark for $T = 0.5$.

Table 6: Comparative of Monte Carlo simulations standard errors for T=1.

T=1	Std MC	Std MC control	Std MC antithetic	Std MC control antithetic
k=,9	0,008855	0,003961	0,005033	0,004802
k=,95	0,008196	0,00401	0,004896	0,004678
k=1	0,007532	0,004005	0,004674	0,00447
k=1,05	0,006875	0,003948	0,004392	0,004204
k=1,1	0,006235	0,00384	0,004074	0,003904
Sum	0,037693	0,019764	0,023069	0,022058

For $T = 1$, the best performer in volatility is the control variate method, as in $T = 0.5$. However, in this case, the control method is better than the other two for all k in terms of volatility.

As expected, using variance reduction techniques lowers the standard deviation. Moreover, using both the antithetic and the control together is always better than only using the antithetic method, but not always better than using only the control variate method. This is because of the way how the control antithetic method is created, that is, by using a control variate to the antithetic variate, and not the other way around.

As stated before, for $T = 0.1$ the benchmark will be control antithetic technique. And, for $T = 0.5$ and $T = 1$ the benchmark will be the control technique.

The next tables show the relative errors of the approximation formulas compared to the Monte Carlos. First the case for $T = 0.1$ will be shown, followed by the case $T = 0.5$, and to finalize, the case for $T = 1$

Table 7: Comparative of 1st order errors (%) with Monte Carlo simulations for T=0,1.

T=0,1	1st Control error	1st Antithetic error	1st Control antithetic error
k=,9	0,127028	0,122131	0,123398
k=,95	0,096458	0,088425	0,090889
k=1	-0,305608	-0,316139	-0,311566
k=1,05	-1,501266	-1,521312	-1,513446
k=1,1	-4,000935	-4,026628	-4,014033
Squared Sum	18,380117	18,650802	18,523541

Table 8: Comparative of 2nd order errors (%) with Monte Carlo simulations for T=0,1.

T=0.1	2nd Control error	2nd Antithetic error	2nd Control antithetic
k=,9	-0,127649	-0,132558	-0,131288
k=,95	-0,247972	-0,256033	-0,25356
k=1	-0,305608	-0,316139	-0,311566
k=1,05	-0,098349	-0,118117	-0,11036
k=1,1	0,672139	0,647599	0,659629
Squared Sum	0,632624	0,616404	0,625892

Focusing on the benchmark, i.e. the control antithetic method, for $k < 1$ the first order approximation formula performs better. As for $k > 1$ the second one performs better. The squared sum of errors is much lower in the second order formula. So, the loss that the second formula has for $k < 1$ is offset for the gain for $k > 1$. Note that in the other methods the same results are obtained.

Table 9: Comparative of 1st order (%) errors with Monte Carlo simulations for T=0,5.

T=0,5	1st Control error	1st Antithetic error	1st Control antithetic error
k=,9	0,324441	0,280563	0,27997
k=,95	0,083264	0,030771	0,030022
k=1	-0,400622	-0,465008	-0,465945
k=1,05	-1,207722	-1,28132	-1,282483
k=1,1	-2,411998	-2,501712	-2,503141
Squared sum	7,54902	8,196239	8,206867

Table 10: Comparative of 2nd order errors (%) with Monte Carlo simulations for T=0,5.

T=0,5	2nd Control error	2nd Antithetic error	2nd Control antithetic
k=,9	-0,409753	-0,453954	-0,454552
k=,95	-0,453952	-0,506728	-0,50748
k=1	-0,400622	-0,465008	-0,465945
k=1,05	-0,198824	-0,271688	-0,272839
k=1,1	0,216577	0,129164	0,127772
Squared sum	0,620905	0,769578	0,772025

For $T = 0.5$, the results are the same as for $T = 0.1$. For $k < 1$, the first order approximation formula performs better, whereas for $k > 1$ the second one is better. The squared sum of errors is much lower in the second order formula. Again, the same results are obtained in all the methods.

Table 11: Comparative of 1st order errors (%) with Monte Carlo simulations for T=1.

T=1	1st Control error	1st Antithetic error	1st Control antithetic error
k=,9	0,324913	0,3298	0,327409
k=,95	-0,004293	0,003721	0,000948
k=1	-0,51903	-0,508157	-0,511364
k=1,05	-1,254317	-1,241146	-1,244835
k=1,1	-2,239728	-2,224874	-2,2291
Squared sum	6,964672	6,857513	6,887192

Table 12: Comparative of 2nd order (%) errors with Monte Carlo simulations for T=1.

T=1	2nd Control error	2nd Antithetic error	2nd Control antithetic error
k=,9	-0,586095	-0,581163	-0,583576
k=,95	-0,595843	-0,587781	-0,59057
k=1	-0,51903	-0,508157	-0,511364
k=1,05	-0,327188	-0,314138	-0,317793
k=1,1	0,014312	0,028839	0,024706
Squared sum	1,075185	1,040975	1,05243

For $T = 1$, the results are again the same. For $k < 1$,the first order approximation formula performs better. And, for $k > 1$, the second one performs better. Again, the second formula is more accurate than the first one.

Note that, for each T, the inflection point is below $k = 1$ for the first order formula. So, the minimum error can be achieved using a k between 0.90 and 1. Whereas for the second formula, the inflection point is above $k = 1$. This gives the idea that the minimum error can be achieved using a k between 1 and 1.1. This is in the line of the results, where the first approximation formula outperformed the second one for $k < 1$, and the other way around for $k > 1$.

7 Conclusions

To sum up the results,

- All of the variance reduction techniques decrease the volatility of the simulations.
- The volatility of Monte Carlo simulations using control and antithetic techniques together is always lower than only using the antithetic technique.
- For $T = 0.1$ the most accurate variance reduction technique is the control antithetic method. For $T = 0.5$ and $T = 1$ the most accurate variance reduction technique is the control method.
- In all cases, the first approximation always performs better than the second one for $k < 1$. The second approximation always performs better than the first one for $k > 1$. For $k = 1$ the errors are always exactly the same in both formulas.
- In all cases, the squared sums of errors of the second approximation formula are much lower than the squared sums of errors of the first one, meaning that the gain in accuracy of the second formula for $k > 1$ offsets the lose in accuracy for $k < 1$. So, it can be concluded that the second order approximation formula is better than the first one.

It is worth reflecting that, using another option, rather than an European call to control the floating Asian options, would have been more effective. One next step could be to use a geometric Asian option. Recall that a closed formula is needed to perform the control variate method. The closed formula for the geometric Asian option was shown in equation(12). The handicap of using a geometric Asian option is that a geometric Asian option is harder to simulate and calculate than an European option. The benefit of using a geometric Asian option is that its correlation with a floating Asian option is higher, so the volatility of the simulations would have been reduced even more. As a result, the estimates of the floating Asian options would have been even more accurate.

References

[1] E. Alòs and J. A. León, "A note on the implied volatility of floating strike Asian options," *Decisions in Economics and Finance*, pp. 1–16, 2019.

[2] I. Nelken, *The Handbook of Exotic Options: Instruments, Analysis, and Applications*. Irwin Professional Pub., 1996.

[3] J. Hull, *Options, Futures, and Other Derivatives*. Pearson Education, 9th ed., 2018.

[4] P. P. Boyle, "Options: A Monte Carlo approach," *Journal of Financial Economics*, vol. 4, no. 3, pp. 323–338, 1977.

[5] A. Hirsa, *Computational Methods in Finance*. Chapman and Hall/CRC Financial Mathematics Series, Taylor & Francis, 2012.

8 Appendix

```
import numpy as np
import math
from scipy.stats import norm
import matplotlib.pyplot as plt
import pandas as pd
import time

M=10000000
sigma=0.5
s0=100
N=252

def BS1(k,T):
    x=np.log(s0)
    K=k*s0
    xstar=np.log(K)
    s=0.5
    sigma=s/math.sqrt(3)
    dplus=((x-xstar)/(sigma*(math.sqrt(T))))+(sigma/2)*math.sqrt(T)
    dminus=((x-xstar)/(sigma*(math.sqrt(T))))-(sigma/2)*math.sqrt(T)
    return((np.exp(x))*norm.cdf(dplus)-K*norm.cdf(dminus))

def BS2(k,T):
    x=np.log(s0)
    K=k*s0
    xstar=np.log(K)
    s=0.5
    sigma=(s/math.sqrt(3))-(math.sqrt(3)/40)*(k-1)
    dplus=((x-xstar)/(sigma*(math.sqrt(T))))+(sigma/2)*math.sqrt(T)
    dminus=((x-xstar)/(sigma*(math.sqrt(T))))-(sigma/2)*math.sqrt(T)
    return((np.exp(x))*norm.cdf(dplus)-K*norm.cdf(dminus))

def euro(k):
    european=max(st[-1]-BS2(k,T),0)
    return(european)

def BSeuro(k,T):
    x=np.log(s0)
    xstar=np.log(BS2(k,T))
    dplus=((x-xstar)/(sigma*(math.sqrt(T))))+(sigma/2)*math.sqrt(T)
    dminus=((x-xstar)/(sigma*(math.sqrt(T))))-(sigma/2)*math.sqrt(T)
    return((np.exp(x))*norm.cdf(dplus)-BS2(k,T)*norm.cdf(dminus))

def fasia(k):
    fasian=max(st[-1]-k*np.mean(st),0)
    return(fasian)

def fasia2(k):
    fasian2=max(st2[-1]-k*np.mean(st2),0)
    return(fasian2)

T=.1
d=T/N
```

```
mcfasia_90_01=[];mcfasia_95_01=[];mcfasia_100_01=[];mcfasia_105_01=[];mcfasia_110_01=[];
mcfasia2_90_01=[];mcfasia2_95_01=[];mcfasia2_100_01=[];mcfasia2_105_01=[];mcfasia2_110_01=[];
mceuro_90_01=[];mceuro_95_01=[];mceuro_100_01=[];mceuro_105_01=[];mceuro_110_01=[];

for j in range(int(M)):
    st=[s0]
    a=s0
    st2=[s0]
    a2=s0
    random=np.random.normal(0,1,252)
    random2=-random
    for i in range(N-1):
        a=a*math.exp((-(sigma**2)/2)*d+sigma*random[i]*math.sqrt(d))
        st.append(a)
        a2=a2*math.exp((-(sigma**2)/2)*d+sigma*random2[i]*math.sqrt(d))
        st2.append(a2)

    mcfasia_90_01.append(fasia(0.9));mcfasia_95_01.append(fasia(0.95));mcfasia_100_01.append(fasia(1));
    mcfasia_105_01.append(fasia(1.05));mcfasia_110_01.append(fasia(1.1))

    mcfasia2_90_01.append(fasia2(0.9));mcfasia2_95_01.append(fasia2(0.95));mcfasia2_100_01.append(fasia2(1));
    mcfasia2_105_01.append(fasia2(1.05));mcfasia2_110_01.append(fasia2(1.1))

    mceuro_90_01.append(euro(.9));mceuro_95_01.append(euro(.95));mceuro_100_01.append(euro(1));
    mceuro_105_01.append(euro(1.05));mceuro_110_01.append(euro(1.1))

mcanti_90_01=(np.array(mcfasia_90_01)+np.array(mcfasia2_90_01))/2
mcanti_95_01=(np.array(mcfasia_95_01)+np.array(mcfasia2_95_01))/2
mcanti_100_01=(np.array(mcfasia_100_01)+np.array(mcfasia2_100_01))/2
mcanti_105_01=(np.array(mcfasia_105_01)+np.array(mcfasia2_105_01))/2
mcanti_110_01=(np.array(mcfasia_110_01)+np.array(mcfasia2_110_01))/2

c_90_01=-np.cov(mcfasia_90_01,mceuro_90_01)[0,1]/np.var(mceuro_90_01)
c_95_01=-np.cov(mcfasia_95_01,mceuro_95_01)[0,1]/np.var(mceuro_95_01)
c_100_01=-np.cov(mcfasia_100_01,mceuro_100_01)[0,1]/np.var(mceuro_100_01)
c_105_01=-np.cov(mcfasia_105_01,mceuro_105_01)[0,1]/np.var(mceuro_105_01)
c_110_01=-np.cov(mcfasia_110_01,mceuro_110_01)[0,1]/np.var(mceuro_110_01)

fasiacontrol_90_01 =mcfasia_90_01+c_90_01*(np.array(mceuro_90_01)-BSeuro(.9,.1))
fasiacontrol_95_01 =mcfasia_95_01+c_95_01*(np.array(mceuro_95_01)-BSeuro(.95,.1))
fasiacontrol_100_01 =mcfasia_100_01+c_100_01*(np.array(mceuro_100_01)-BSeuro(1,.1))
fasiacontrol_105_01 =mcfasia_105_01+c_105_01*(np.array(mceuro_105_01)-BSeuro(1.05,.1))
fasiacontrol_110_01 =mcfasia_110_01+c_110_01*(np.array(mceuro_110_01)-BSeuro(1.1,.1))

c2_90_01=-np.cov(mcanti_90_01,mceuro_90_01)[0,1]/np.var(mceuro_90_01)
c2_95_01=-np.cov(mcanti_95_01,mceuro_95_01)[0,1]/np.var(mceuro_95_01)
c2_100_01=-np.cov(mcanti_100_01,mceuro_100_01)[0,1]/np.var(mceuro_100_01)
c2_105_01=-np.cov(mcanti_105_01,mceuro_105_01)[0,1]/np.var(mceuro_105_01)
c2_110_01=-np.cov(mcanti_110_01,mceuro_110_01)[0,1]/np.var(mceuro_110_01)

fasiacontrol2_90_01 =mcanti_90_01+c2_90_01*(np.array(mceuro_90_01)-BSeuro(.9,.1))
fasiacontrol2_95_01 =mcanti_95_01+c2_95_01*(np.array(mceuro_95_01)-BSeuro(.95,.1))
fasiacontrol2_100_01 =mcanti_100_01+c2_100_01*(np.array(mceuro_100_01)-BSeuro(1,.1))
fasiacontrol2_105_01 =mcanti_105_01+c2_105_01*(np.array(mceuro_105_01)-BSeuro(1.05,.1))
fasiacontrol2_110_01 =mcanti_110_01+c2_110_01*(np.array(mceuro_110_01)-BSeuro(1.1,.1))
```

```
meanfasia_90_01=np.mean(mcfasia_90_01);meanfasia_95_01=np.mean(mcfasia_95_01)
meanfasia_100_01=np.mean(mcfasia_100_01);meanfasia_105_01=np.mean(mcfasia_105_01)
meanfasia_110_01=np.mean(mcfasia_110_01)

stdfasia_90_01=np.std(mcfasia_90_01)/math.sqrt(M);stdfasia_95_01=np.std(mcfasia_95_01)/math.sqrt(M)
stdfasia_100_01=np.std(mcfasia_100_01)/math.sqrt(M);stdfasia_105_01=np.std(mcfasia_105_01)/math.sqrt(M)
stdfasia_110_01=np.std(mcfasia_110_01)/math.sqrt(M)

meanfasiac_90_01=np.mean(fasiacontrol_90_01);meanfasiac_95_01=np.mean(fasiacontrol_95_01)
meanfasiac_100_01=np.mean(fasiacontrol_100_01);meanfasiac_105_01=np.mean(fasiacontrol_105_01)
meanfasiac_110_01=np.mean(fasiacontrol_110_01)

stdfasiac_90_01=np.std(fasiacontrol_90_01)/math.sqrt(M);stdfasiac_95_01=np.std(fasiacontrol_95_01)/math.sqrt(M)
stdfasiac_100_01=np.std(fasiacontrol_100_01)/math.sqrt(M);stdfasiac_105_01=np.std(fasiacontrol_105_01)/math.sqrt(M)
stdfasiac_110_01=np.std(fasiacontrol_110_01)/math.sqrt(M)

meananti_90_01=np.mean(mcanti_90_01);meananti_95_01=np.mean(mcanti_95_01)
meananti_100_01=np.mean(mcanti_100_01);meananti_105_01=np.mean(mcanti_105_01)
meananti_110_01=np.mean(mcanti_110_01)

stdanti_90_01=np.std(mcanti_90_01)/math.sqrt(M);stdanti_95_01=np.std(mcanti_95_01)/math.sqrt(M)
stdanti_100_01=np.std(mcanti_100_01)/math.sqrt(M);stdanti_105_01=np.std(mcanti_105_01)/math.sqrt(M)
stdanti_110_01=np.std(mcanti_110_01)/math.sqrt(M)

meanfasiac2_90_01=np.mean(fasiacontrol2_90_01);meanfasiac2_95_01=np.mean(fasiacontrol2_95_01)
meanfasiac2_100_01=np.mean(fasiacontrol2_100_01);meanfasiac2_105_01=np.mean(fasiacontrol2_105_01)
meanfasiac2_110_01=np.mean(fasiacontrol2_110_01)

stdfasiac2_90_01=np.std(fasiacontrol2_90_01)/math.sqrt(M);stdfasiac2_95_01=np.std(fasiacontrol2_95_01)/math.sqrt(M)
stdfasiac2_100_01=np.std(fasiacontrol2_100_01)/math.sqrt(M);stdfasiac2_105_01=np.std(fasiacontrol2_105_01)/math.sqrt(M)
stdfasiac2_110_01=np.std(fasiacontrol2_110_01)/math.sqrt(M)

T=.5
d=T/N

mcfasia_90_05=[];mcfasia_95_05=[];mcfasia_100_05=[];mcfasia_105_05=[];mcfasia_110_05=[];
mcfasia2_90_05=[];mcfasia2_95_05=[];mcfasia2_100_05=[];mcfasia2_105_05=[];mcfasia2_110_05=[];
mceuro_90_05=[];mceuro_95_05=[];mceuro_100_05=[];mceuro_105_05=[];mceuro_110_05=[];

for j in range(int(M)):
    st=[s0]
    a=s0
    st2=[s0]
    a2=s0
    random=np.random.normal(0,1,252)
    random2=-random
    for i in range(N-1):
        a=a*math.exp((-(sigma**2)/2)*d+sigma*random[i]*math.sqrt(d))
        st.append(a)
        a2=a2*math.exp((-(sigma**2)/2)*d+sigma*random2[i]*math.sqrt(d))
        st2.append(a2)

    mcfasia_90_05.append(fasia(0.9));mcfasia_95_05.append(fasia(0.95));mcfasia_100_05.append(fasia(1));
    mcfasia_105_05.append(fasia(1.05));mcfasia_110_05.append(fasia(1.1))
```

```
    mcfasia2_90_05.append(fasia2(0.9));mcfasia2_95_05.append(fasia2(0.95));mcfasia2_100_05.append(fasia2(1));
    mcfasia2_105_05.append(fasia2(1.05));mcfasia2_110_05.append(fasia2(1.1))

    mceuro_90_05.append(euro(.9));mceuro_95_05.append(euro(.95));mceuro_100_05.append(euro(1));
    mceuro_105_05.append(euro(1.05));mceuro_110_05.append(euro(1.1))

mcanti_90_05=(np.array(mcfasia_90_05)+np.array(mcfasia2_90_05))/2
mcanti_95_05=(np.array(mcfasia_95_05)+np.array(mcfasia2_95_05))/2
mcanti_100_05=(np.array(mcfasia_100_05)+np.array(mcfasia2_100_05))/2
mcanti_105_05=(np.array(mcfasia_105_05)+np.array(mcfasia2_105_05))/2
mcanti_110_05=(np.array(mcfasia_110_05)+np.array(mcfasia2_110_05))/2

c_90_05=-np.cov(mcfasia_90_05,mceuro_90_05)[0,1]/np.var(mceuro_90_05)
c_95_05=-np.cov(mcfasia_95_05,mceuro_95_05)[0,1]/np.var(mceuro_95_05)
c_100_05=-np.cov(mcfasia_100_05,mceuro_100_05)[0,1]/np.var(mceuro_100_05)
c_105_05=-np.cov(mcfasia_105_05,mceuro_105_05)[0,1]/np.var(mceuro_105_05)
c_110_05=-np.cov(mcfasia_110_05,mceuro_110_05)[0,1]/np.var(mceuro_110_05)

fasiacontrol_90_05 =mcfasia_90_05+c_90_05*(np.array(mceuro_90_05)-BSeuro(.9,.5))
fasiacontrol_95_05 =mcfasia_95_05+c_95_05*(np.array(mceuro_95_05)-BSeuro(.95,.5))
fasiacontrol_100_05 =mcfasia_100_05+c_100_05*(np.array(mceuro_100_05)-BSeuro(1,.5))
fasiacontrol_105_05 =mcfasia_105_05+c_105_05*(np.array(mceuro_105_05)-BSeuro(1.05,.5))
fasiacontrol_110_05 =mcfasia_110_05+c_110_05*(np.array(mceuro_110_05)-BSeuro(1.1,.5))

c2_90_05=-np.cov(mcanti_90_05,mceuro_90_05)[0,1]/np.var(mceuro_90_05)
c2_95_05=-np.cov(mcanti_95_05,mceuro_95_05)[0,1]/np.var(mceuro_95_05)
c2_100_05=-np.cov(mcanti_100_05,mceuro_100_05)[0,1]/np.var(mceuro_100_05)
c2_105_05=-np.cov(mcanti_105_05,mceuro_105_05)[0,1]/np.var(mceuro_105_05)
c2_110_05=-np.cov(mcanti_110_05,mceuro_110_05)[0,1]/np.var(mceuro_110_05)

fasiacontrol2_90_05 =mcanti_90_05+c2_90_05*(np.array(mceuro_90_05)-BSeuro(.9,.5))
fasiacontrol2_95_05 =mcanti_95_05+c2_95_05*(np.array(mceuro_95_05)-BSeuro(.95,.5))
fasiacontrol2_100_05 =mcanti_100_05+c2_100_05*(np.array(mceuro_100_05)-BSeuro(1,.5))
fasiacontrol2_105_05 =mcanti_105_05+c2_105_05*(np.array(mceuro_105_05)-BSeuro(1.05,.5))
fasiacontrol2_110_05 =mcanti_110_05+c2_110_05*(np.array(mceuro_110_05)-BSeuro(1.1,.5))

meanfasia_90_05=np.mean(mcfasia_90_05);meanfasia_95_05=np.mean(mcfasia_95_05)
meanfasia_100_05=np.mean(mcfasia_100_05);meanfasia_105_05=np.mean(mcfasia_105_05)
meanfasia_110_05=np.mean(mcfasia_110_05)

stdfasia_90_05 =np.std(mcfasia_90_05)/math.sqrt(M);stdfasia_95_05=np.std(mcfasia_95_05)/math.sqrt(M)
stdfasia_100_05 =np.std(mcfasia_100_05)/math.sqrt(M);stdfasia_105_05=np.std(mcfasia_105_05)/math.sqrt(M)
stdfasia_110_05 =np.std(mcfasia_110_05)/math.sqrt(M)

meanfasiac_90_05=np.mean(fasiacontrol_90_05);meanfasiac_95_05=np.mean(fasiacontrol_95_05)
meanfasiac_100_05=np.mean(fasiacontrol_100_05);meanfasiac_105_05=np.mean(fasiacontrol_105_05)
meanfasiac_110_05=np.mean(fasiacontrol_110_05)

stdfasiac_90_05 =np.std(fasiacontrol_90_05)/math.sqrt(M);stdfasiac_95_05=np.std(fasiacontrol_95_05)/math.sqrt(M)
stdfasiac_100_05 =np.std(fasiacontrol_100_05)/math.sqrt(M);stdfasiac_105_05=np.std(fasiacontrol_105_05)/math.sqrt(M)
stdfasiac_110_05 =np.std(fasiacontrol_110_05)/math.sqrt(M)

meananti_90_05=np.mean(mcanti_90_05);meananti_95_05=np.mean(mcanti_95_05)
meananti_100_05=np.mean(mcanti_100_05);meananti_105_05=np.mean(mcanti_105_05)
meananti_110_05=np.mean(mcanti_110_05)
```

```
stdanti_90_05=np.std(mcanti_90_05)/math.sqrt(M);stdanti_95_05=np.std(mcanti_95_05)/math.sqrt(M)
stdanti_100_05=np.std(mcanti_100_05)/math.sqrt(M);stdanti_105_05=np.std(mcanti_105_05)/math.sqrt(M)
stdanti_110_05=np.std(mcanti_110_05)/math.sqrt(M)

meanfasiac2_90_05=np.mean(fasiacontrol2_90_05);meanfasiac2_95_05=np.mean(fasiacontrol2_95_05)
meanfasiac2_100_05=np.mean(fasiacontrol2_100_05);meanfasiac2_105_05=np.mean(fasiacontrol2_105_05)
meanfasiac2_110_05=np.mean(fasiacontrol2_110_05)

stdfasiac2_90_05=np.std(fasiacontrol2_90_05)/math.sqrt(M);stdfasiac2_95_05=np.std(fasiacontrol2_95_05)/math.sqrt(M)
stdfasiac2_100_05=np.std(fasiacontrol2_100_05)/math.sqrt(M);stdfasiac2_105_05=np.std(fasiacontrol2_105_05)/math.sqrt(M)
stdfasiac2_110_05=np.std(fasiacontrol2_110_05)/math.sqrt(M)

T=1
d=T/N

mcfasia_90_1=[];mcfasia_95_1=[];mcfasia_100_1=[];mcfasia_105_1=[];mcfasia_110_1=[];
mcfasia2_90_1=[];mcfasia2_95_1=[];mcfasia2_100_1=[];mcfasia2_105_1=[];mcfasia2_110_1=[];
mceuro_90_1=[];mceuro_95_1=[];mceuro_100_1=[];mceuro_105_1=[];mceuro_110_1=[];

for j in range(int(M)):
    st=[s0]
    a=s0
    st2=[s0]
    a2=s0
    random=np.random.normal(0,1,252)
    random2=-random
    for i in range(N-1):
        a=a*math.exp((-(sigma**2)/2)*d+sigma*random[i]*math.sqrt(d))
        st.append(a)
        a2=a2*math.exp((-(sigma**2)/2)*d+sigma*random2[i]*math.sqrt(d))
        st2.append(a2)

    mcfasia_90_1.append(fasia(0.9));mcfasia_95_1.append(fasia(0.95));mcfasia_100_1.append(fasia(1));
    mcfasia_105_1.append(fasia(1.05));mcfasia_110_1.append(fasia(1.1))

    mcfasia2_90_1.append(fasia2(0.9));mcfasia2_95_1.append(fasia2(0.95));mcfasia2_100_1.append(fasia2(1));
    mcfasia2_105_1.append(fasia2(1.05));mcfasia2_110_1.append(fasia2(1.1))

    mceuro_90_1.append(euro(.9));mceuro_95_1.append(euro(.95));mceuro_100_1.append(euro(1));
    mceuro_105_1.append(euro(1.05));mceuro_110_1.append(euro(1.1))

mcanti_90_1=(np.array(mcfasia_90_1)+np.array(mcfasia2_90_1))/2
mcanti_95_1=(np.array(mcfasia_95_1)+np.array(mcfasia2_95_1))/2
mcanti_100_1=(np.array(mcfasia_100_1)+np.array(mcfasia2_100_1))/2
mcanti_105_1=(np.array(mcfasia_105_1)+np.array(mcfasia2_105_1))/2
mcanti_110_1=(np.array(mcfasia_110_1)+np.array(mcfasia2_110_1))/2

c_90_1=-np.cov(mcfasia_90_1,mceuro_90_1)[0,1]/np.var(mceuro_90_1)
c_95_1=-np.cov(mcfasia_95_1,mceuro_95_1)[0,1]/np.var(mceuro_95_1)
c_100_1=-np.cov(mcfasia_100_1,mceuro_100_1)[0,1]/np.var(mceuro_100_1)
c_105_1=-np.cov(mcfasia_105_1,mceuro_105_1)[0,1]/np.var(mceuro_105_1)
c_110_1=-np.cov(mcfasia_110_1,mceuro_110_1)[0,1]/np.var(mceuro_110_1)

fasiacontrol_90_1 =mcfasia_90_1+c_90_1*(np.array(mceuro_90_1)-BSeuro(.9,1))
fasiacontrol_95_1 =mcfasia_95_1+c_95_1*(np.array(mceuro_95_1)-BSeuro(.95,1))
fasiacontrol_100_1 =mcfasia_100_1+c_100_1*(np.array(mceuro_100_1)-BSeuro(1,1))
```

```
fasiacontrol_105_1 =mcfasia_105_1+c_105_1*(np.array(mceuro_105_1)-BSeuro(1.05,1))
fasiacontrol_110_1 =mcfasia_110_1+c_110_1*(np.array(mceuro_110_1)-BSeuro(1.1,1))

c2_90_1=-np.cov(mcanti_90_1,mceuro_90_1)[0,1]/np.var(mceuro_90_1)
c2_95_1=-np.cov(mcanti_95_1,mceuro_95_1)[0,1]/np.var(mceuro_95_1)
c2_100_1=-np.cov(mcanti_100_1,mceuro_100_1)[0,1]/np.var(mceuro_100_1)
c2_105_1=-np.cov(mcanti_105_1,mceuro_105_1)[0,1]/np.var(mceuro_105_1)
c2_110_1=-np.cov(mcanti_110_1,mceuro_110_1)[0,1]/np.var(mceuro_110_1)

fasiacontrol2_90_1 =mcanti_90_1+c2_90_1*(np.array(mceuro_90_1)-BSeuro(.9,1))
fasiacontrol2_95_1 =mcanti_95_1+c2_95_1*(np.array(mceuro_95_1)-BSeuro(.95,1))
fasiacontrol2_100_1 =mcanti_100_1+c2_100_1*(np.array(mceuro_100_1)-BSeuro(1,1))
fasiacontrol2_105_1 =mcanti_105_1+c2_105_1*(np.array(mceuro_105_1)-BSeuro(1.05,1))
fasiacontrol2_110_1 =mcanti_110_1+c2_110_1*(np.array(mceuro_110_1)-BSeuro(1.1,1))

meanfasia_90_1=np.mean(mcfasia_90_1);meanfasia_95_1=np.mean(mcfasia_95_1)
meanfasia_100_1=np.mean(mcfasia_100_1);meanfasia_105_1=np.mean(mcfasia_105_1)
meanfasia_110_1=np.mean(mcfasia_110_1)

stdfasia_90_1 =np.std(mcfasia_90_1)/math.sqrt(M);stdfasia_95_1=np.std(mcfasia_95_1)/math.sqrt(M)
stdfasia_100_1 =np.std(mcfasia_100_1)/math.sqrt(M);stdfasia_105_1=np.std(mcfasia_105_1)/math.sqrt(M)
stdfasia_110_1 =np.std(mcfasia_110_1)/math.sqrt(M)

meanfasiac_90_1=np.mean(fasiacontrol_90_1);meanfasiac_95_1=np.mean(fasiacontrol_95_1)
meanfasiac_100_1=np.mean(fasiacontrol_100_1);meanfasiac_105_1=np.mean(fasiacontrol_105_1)
meanfasiac_110_1=np.mean(fasiacontrol_110_1)

stdfasiac_90_1 =np.std(fasiacontrol_90_1)/math.sqrt(M);stdfasiac_95_1=np.std(fasiacontrol_95_1)/math.sqrt(M)
stdfasiac_100_1 =np.std(fasiacontrol_100_1)/math.sqrt(M);stdfasiac_105_1=np.std(fasiacontrol_105_1)/math.sqrt(M)
stdfasiac_110_1 =np.std(fasiacontrol_110_1)/math.sqrt(M)

meananti_90_1=np.mean(mcanti_90_1);meananti_95_1=np.mean(mcanti_95_1)
meananti_100_1=np.mean(mcanti_100_1);meananti_105_1=np.mean(mcanti_105_1)
meananti_110_1=np.mean(mcanti_110_1)

stdanti_90_1=np.std(mcanti_90_1)/math.sqrt(M);stdanti_95_1=np.std(mcanti_95_1)/math.sqrt(M)
stdanti_100_1=np.std(mcanti_100_1)/math.sqrt(M);stdanti_105_1=np.std(mcanti_105_1)/math.sqrt(M)
stdanti_110_1=np.std(mcanti_110_1)/math.sqrt(M)

meanfasiac2_90_1=np.mean(fasiacontrol2_90_1);meanfasiac2_95_1=np.mean(fasiacontrol2_95_1)
meanfasiac2_100_1=np.mean(fasiacontrol2_100_1);meanfasiac2_105_1=np.mean(fasiacontrol2_105_1)
meanfasiac2_110_1=np.mean(fasiacontrol2_110_1)

stdfasiac2_90_1 =np.std(fasiacontrol2_90_1)/math.sqrt(M);stdfasiac2_95_1=np.std(fasiacontrol2_95_1)/math.sqrt(M)
stdfasiac2_100_1 =np.std(fasiacontrol2_100_1)/math.sqrt(M);stdfasiac2_105_1=np.std(fasiacontrol2_105_1)/math.sqrt(M)
stdfasiac2_110_1 =np.std(fasiacontrol2_110_1)/math.sqrt(M)

data1 = { '1st Order':[BS1(.9,.1),BS1(.95,.1),BS1(1,.1),BS1(1.05,.1),BS1(1.1,.1)],
          '2st Order':[BS2(.90,.1),BS2(.95,.1),BS2(1,.1),BS2(1.05,.1),BS2(1.1,.1)],
          "MC floating asian":[meanfasia_90_01,meanfasia_95_01,meanfasia_100_01,meanfasia_105_01,meanfasia_110_01],
          "Std floating asian" :[ stdfasia_90_01 , stdfasia_95_01 , stdfasia_100_01 , stdfasia_105_01 , stdfasia_110_01 ],
          "MC floating asian
                control" :[meanfasiac_90_01,meanfasiac_95_01,meanfasiac_100_01,meanfasiac_105_01,meanfasiac_110_01],
          "Std floating asian control" :[ stdfasiac_90_01 , stdfasiac_95_01 , stdfasiac_100_01 , stdfasiac_105_01 , stdfasiac_110_01 ],
          "MC floating asian antitetic" :[meananti_90_01,meananti_95_01,meananti_100_01,meananti_105_01,meananti_110_01],
          "Std floating asian antitetic" :[ stdanti_90_01 , stdanti_95_01 , stdanti_100_01 , stdanti_105_01 , stdanti_110_01 ],
```

```
        "MC floating asian control antitetic":[meanfasiac2_90_01,meanfasiac2_95_01,meanfasiac2_100_01,
        meanfasiac2_105_01,meanfasiac2_110_01],
        "Std floating asian control antitetic":[stdfasiac2_90_01,stdfasiac2_95_01,stdfasiac2_100_01,stdfasiac2_105_01,
        stdfasiac2_110_01],

        "Error 1st (%) control":[100*(1-BS1(.9,.1)/meanfasiac_90_01),
                    100*(1-BS1(.95,.1)/meanfasiac_95_01),
                    100*(1-BS1(1,.1)/meanfasiac_100_01),
                    100*(1-BS1(1.05,.1)/meanfasiac_105_01),
                    100*(1-BS1(1.1,.1)/meanfasiac_110_01)],
        "Error 2st (%) control":[100*(1-BS2(.9,.1)/meanfasiac_90_01),
                    100*(1-BS2(.95,.1)/meanfasiac_95_01),
                    100*(1-BS2(1,.1)/meanfasiac_100_01),
                    100*(1-BS2(1.05,.1)/meanfasiac_105_01),
                    100*(1-BS2(1.1,.1)/meanfasiac_110_01)],
        "Error 1st (%) anti":[100*(1-BS1(.9,.1)/meananti_90_01),
                    100*(1-BS1(.95,.1)/meananti_95_01),
                    100*(1-BS1(1,.1)/meananti_100_01),
                    100*(1-BS1(1.05,.1)/meananti_105_01),
                    100*(1-BS1(1.1,.1)/meananti_110_01)],
        "Error 2st (%) anti":[100*(1-BS2(.9,.1)/meananti_90_01),
                    100*(1-BS2(.95,.1)/meananti_95_01),
                    100*(1-BS2(1,.1)/meananti_100_01),
                    100*(1-BS2(1.05,.1)/meananti_105_01),
                    100*(1-BS2(1.1,.1)/meananti_110_01)],
      "Error 1st (%) control anti":[100*(1-BS1(.9,.1)/meanfasiac2_90_01),
                    100*(1-BS1(.95,.1)/meanfasiac2_95_01),
                    100*(1-BS1(1,.1)/meanfasiac2_100_01),
                    100*(1-BS1(1.05,.1)/meanfasiac2_105_01),
                    100*(1-BS1(1.1,.1)/meanfasiac2_110_01)],
        "Error 2st (%) control anti":[100*(1-BS2(.9,.1)/meanfasiac2_90_01),
                    100*(1-BS2(.95,.1)/meanfasiac2_95_01),
                    100*(1-BS2(1,.1)/meanfasiac2_100_01),
                    100*(1-BS2(1.05,.1)/meanfasiac2_105_01),
                    100*(1-BS2(1.1,.1)/meanfasiac2_110_01)],
        }

df1 = pd.DataFrame(data1,
                    index=['k=.9', 'k=.95', 'k=1', 'k=1.05','k=1.1'],
                    columns=["1st Order",'2st Order',"MC floating asian","MC floating asian control",
                             "MC floating asian antitetic","MC floating asian control antitetic",
                             "Std floating asian", "Std floating asian control","Std floating asian antitetic",
                             "Std floating asian control antitetic",
                            "Error 1st (%) control","Error 1st (%) anti","Error 1st (%) control anti",
                             "Error 2st (%) control","Error 2st (%) anti","Error 2st (%) control anti"])
print("T=.1",df1,sep="\n")
print(sep="\n")

data2 = { '1st Order':[BS1(.9,.5),BS1(.95,.5),BS1(1,.5),BS1(1.05,.5),BS1(1.1,.5)],
          '2st Order':[BS2(.90,.5),BS2(.95,.5),BS2(1,.5),BS2(1.05,.5),BS2(1.1,.5)],
          "MC floating asian":[meanfasia_90_05,meanfasia_95_05,meanfasia_100_05,meanfasia_105_05,meanfasia_110_05],
          "Std floating asian":[stdfasia_90_05,stdfasia_95_05,stdfasia_100_05,stdfasia_105_05,stdfasia_110_05],
          "MC floating asian
                control":[meanfasiac_90_05,meanfasiac_95_05,meanfasiac_100_05,meanfasiac_105_05,meanfasiac_110_05],
          "Std floating asian control":[stdfasiac_90_05,stdfasiac_95_05,stdfasiac_100_05,stdfasiac_105_05,stdfasiac_110_05],
          "MC floating asian antitetic":[meananti_90_05,meananti_95_05,meananti_100_05,meananti_105_05,meananti_110_05],
```

```
        "Std floating asian antitetic":[stdanti_90_05,stdanti_95_05,stdanti_100_05,stdanti_105_05,stdanti_110_05],
        "MC floating asian control antitetic":[meanfasiac2_90_05,meanfasiac2_95_05,meanfasiac2_100_05,
       meanfasiac2_105_05,meanfasiac2_110_05],
        "Std floating asian control antitetic":[stdfasiac2_90_05,stdfasiac2_95_05,stdfasiac2_100_05,stdfasiac2_105_05,
        stdfasiac2_110_05],

         "Error 1st (%) control":[100*(1-BS1(.9,.5)/meanfasiac_90_05),
                    100*(1-BS1(.95,.5)/meanfasiac_95_05),
                    100*(1-BS1(1,.5)/meanfasiac_100_05),
                    100*(1-BS1(1.05,.5)/meanfasiac_105_05),
                    100*(1-BS1(1.1,.5)/meanfasiac_110_05)],
        "Error 2st (%) control":[100*(1-BS2(.9,.5)/meanfasiac_90_05),
                    100*(1-BS2(.95,.5)/meanfasiac_95_05),
                    100*(1-BS2(1,.5)/meanfasiac_100_05),
                    100*(1-BS2(1.05,.5)/meanfasiac_105_05),
                    100*(1-BS2(1.1,.5)/meanfasiac_110_05)],
         "Error 1st (%) anti":[100*(1-BS1(.9,.5)/meananti_90_05),
                    100*(1-BS1(.95,.5)/meananti_95_05),
                    100*(1-BS1(1,.5)/meananti_100_05),
                    100*(1-BS1(1.05,.5)/meananti_105_05),
                    100*(1-BS1(1.1,.5)/meananti_110_05)],
        "Error 2st (%) anti":[100*(1-BS2(.9,.5)/meananti_90_05),
                    100*(1-BS2(.95,.5)/meananti_95_05),
                    100*(1-BS2(1,.5)/meananti_100_05),
                    100*(1-BS2(1.05,.5)/meananti_105_05),
                    100*(1-BS2(1.1,.5)/meananti_110_05)],
         "Error 1st (%) control anti":[100*(1-BS1(.9,.5)/meanfasiac2_90_05),
                    100*(1-BS1(.95,.5)/meanfasiac2_95_05),
                    100*(1-BS1(1,.5)/meanfasiac2_100_05),
                    100*(1-BS1(1.05,.5)/meanfasiac2_105_05),
                    100*(1-BS1(1.1,.5)/meanfasiac2_110_05)],
        "Error 2st (%) control anti":[100*(1-BS2(.9,.5)/meanfasiac2_90_05),
                    100*(1-BS2(.95,.5)/meanfasiac2_95_05),
                    100*(1-BS2(1,.5)/meanfasiac2_100_05),
                    100*(1-BS2(1.05,.5)/meanfasiac2_105_05),
                    100*(1-BS2(1.1,.5)/meanfasiac2_110_05)],
          }

df2 = pd.DataFrame(data2,
                    index=['k=.9', 'k=.95', 'k=1', 'k=1.05','k=1.1'],
                    columns=["1st Order",'2st Order',"MC floating asian","MC floating asian control",
                             "MC floating asian antitetic","MC floating asian control antitetic",
                             "Std floating asian", "Std floating asian control","Std floating asian antitetic",
                             "Std floating asian control antitetic",
                            "Error 1st (%) control","Error 1st (%) anti","Error 1st (%) control anti",
                             "Error 2st (%) control","Error 2st (%) anti","Error 2st (%) control anti"])
print("T=.5",df2,sep="\n")
print(sep="\n")

data3 = { '1st Order':[BS1(.9,1),BS1(.95,1),BS1(1,1),BS1(1.05,1),BS1(1.1,1)],
          '2st Order':[BS2(.90,1),BS2(.95,1),BS2(1,1),BS2(1.05,1),BS2(1.1,1)],
          "MC floating asian":[meanfasia_90_1,meanfasia_95_1,meanfasia_100_1,meanfasia_105_1,meanfasia_110_1],
          "Std floating asian" :[ stdfasia_90_1 , stdfasia_95_1 , stdfasia_100_1 , stdfasia_105_1 , stdfasia_110_1 ],
          "MC floating asian control" :[meanfasiac_90_1,meanfasiac_95_1,meanfasiac_100_1,meanfasiac_105_1,meanfasiac_110_1],
          "Std floating asian control" :[ stdfasiac_90_1 , stdfasiac_95_1 , stdfasiac_100_1 , stdfasiac_105_1 , stdfasiac_110_1 ],
          "MC floating asian antitetic" :[meananti_90_1,meananti_95_1,meananti_100_1,meananti_105_1,meananti_110_1],
```

```
        "Std floating asian antitetic":[stdanti_90_1,stdanti_95_1,stdanti_100_1,stdanti_105_1,stdanti_110_1],
        "MC floating asian control antitetic":[meanfasiac2_90_1,meanfasiac2_95_1,meanfasiac2_100_1,
       meanfasiac2_105_1,meanfasiac2_110_1],
       "Std floating asian control antitetic":[stdfasiac2_90_1,stdfasiac2_95_1,stdfasiac2_100_1,stdfasiac2_105_1,
        stdfasiac2_110_1],

       "Error 1st (%) control":[100*(1-BS1(.9,1)/meanfasiac_90_1),
                    100*(1-BS1(.95,1)/meanfasiac_95_1),
                    100*(1-BS1(1,1)/meanfasiac_100_1),
                    100*(1-BS1(1.05,1)/meanfasiac_105_1),
                    100*(1-BS1(1.1,1)/meanfasiac_110_1)],
       "Error 2st (%) control":[100*(1-BS2(.9,1)/meanfasiac_90_1),
                    100*(1-BS2(.95,1)/meanfasiac_95_1),
                    100*(1-BS2(1,1)/meanfasiac_100_1),
                    100*(1-BS2(1.05,1)/meanfasiac_105_1),
                    100*(1-BS2(1.1,1)/meanfasiac_110_1)],
       "Error 1st (%) anti":[100*(1-BS1(.9,1)/meananti_90_1),
                    100*(1-BS1(.95,1)/meananti_95_1),
                    100*(1-BS1(1,1)/meananti_100_1),
                    100*(1-BS1(1.05,1)/meananti_105_1),
                    100*(1-BS1(1.1,1)/meananti_110_1)],
       "Error 2st (%) anti":[100*(1-BS2(.9,1)/meananti_90_1),
                    100*(1-BS2(.95,1)/meananti_95_1),
                    100*(1-BS2(1,1)/meananti_100_1),
                    100*(1-BS2(1.05,1)/meananti_105_1),
                    100*(1-BS2(1.1,1)/meananti_110_1)],
         "Error 1st (%) control anti":[100*(1-BS1(.9,1)/meanfasiac2_90_1),
                    100*(1-BS1(.95,1)/meanfasiac2_95_1),
                    100*(1-BS1(1,1)/meanfasiac2_100_1),
                    100*(1-BS1(1.05,1)/meanfasiac2_105_1),
                    100*(1-BS1(1.1,1)/meanfasiac2_110_1)],
       "Error 2st (%) control anti":[100*(1-BS2(.9,1)/meanfasiac2_90_1),
                    100*(1-BS2(.95,1)/meanfasiac2_95_1),
                    100*(1-BS2(1,1)/meanfasiac2_100_1),
                    100*(1-BS2(1.05,1)/meanfasiac2_105_1),
                    100*(1-BS2(1.1,1)/meanfasiac2_110_1)],
       }

df3 = pd.DataFrame(data3,
                   index=['k=.9', 'k=.95', 'k=1', 'k=1.05','k=1.1'],
                   columns= ["1st Order",'2st Order',"MC floating asian","MC floating asian control",
                             "MC floating asian antitetic","MC floating asian control antitetic",
                             "Std floating asian", "Std floating asian control","Std floating asian antitetic",
                             "Std floating asian control antitetic",
                            "Error 1st (%) control","Error 1st (%) anti","Error 1st (%) control anti",
                             "Error 2st (%) control","Error 2st (%) anti","Error 2st (%) control anti"])
print("T=1",df3,sep="\n")
print(sep="\n")
```