Call option's pricing through Monte Carlo simulations

Sebastiano Monti, ID: 2052399

(Dated: May 21, 2022)

This report aims at describing an important technique that is commonly used in order to price options: Monte Carlo simulation. In particular, different implementations of this method have been used for different purposes. Firstly, we were interested in visualizing a certain number of randomly generated trajectories, then we used a high number of one- and multiple-step Monte Carlo simulations in order to price vanilla call options and other path dependent calls, like Asian and lookback options.

BRIEF THEORETICAL DESCRIPTION

Monte Carlo simulations in finance are used to calculate the value of an option with multiple sources of uncertainty. In our case of study, option's pricing through Monte Carlo simulations relies on a risk neutral evaluation, with the source of uncertainty represented by the price of a particular underlying S_t at any discrete time before the maturity T, with an initial stock price S_0. The price of the underlying is modeled in such a way that it follows a geometric Brownian motion with drift constant μ and volatility σ in the following general way.

$$dS_t = \mu S_t dt + \sigma S_t dW_t \tag{1}$$

Where dW_t represents a risk neutral probability measure and it is found in practice through random sampling from a normal distribution.

For enough simulated trajectories, the value of a European option, computed through Monte Carlo method, should be the same as the one predicted by the Black-Scholes' model, since the underlying random process is the same.

Let us remember that the call option's price of the Black-Scholes' model is given by the following formulas.

$$c_{BS} = S_0\mathcal{N}(d_1) - Ke^{-rT}\mathcal{N}(d_2) \tag{2}$$

Where K is the strike price of the underlying, r the interest rate and $\mathcal{N}(x)$ represents the cumulative function of a normal randomly distributed variable. d_1, d_2 are given by:

$$d_1 = \frac{\log\left(\frac{S_0}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} \quad d_2 = \frac{\log\left(\frac{S_0}{K}\right) + \left(r - \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} \tag{3}$$

Once the random path of the underlying has been generated, the discounted payoff of a vanilla European call option is given by the following.

$$c_{Eu} = (S_T - K)^+ e^{-rT} \tag{4}$$

Asian and lookback option prices depend on the past history of the underlying, namely, the trajectories of respective paths. In particular, their respective discounted payoffs can be computed by the following relations.

$$c_{Asian} = \left(\frac{1}{T}\int_0^T S_t dt - K\right)^+ e^{-rT} \tag{5}$$

$$c_{Lookback} = \left(S_T - min_{t\leq T}\left(S_t\right)\right)e^{-rT} \tag{6}$$

In order to price the options in case of multiple simulated trajectories, it is then appropriate to take the average of the discounted payoffs associated to each trajectory.

METHODS

All the formulas of the previous section have been implemented through some functions into a VBA script, using Excel. The entire code is shown in last section of this report.

In this way, just by fixing some values for S_0, K, T, r, σ, the dividend yield q, the number of time steps n and the number of randomly generated trajectories m, it is possible to recall the VBA functions and compute the call options pricing for each considered case. In particular, the chosen values for the above parameters are resumed in Table I.

S_0 [\$]	100
K [\$]	99
T [$years$]	1
r	1%
σ	20%
q	0

TABLE I. Fixed values of the parameters.

As first thing, an ensemble of $m = 100$ trajectories has been plotted, using $n = 252$ time steps (total number of open market days in the maturity time of one year) through the functions defined as `GBMPath` and `GBMSimulation`. These pieces of code were taken from site [2].

After having checked the randomness of the plotted paths, we proceeded with the pricing of a vanilla European call option by simulating $m = 5000$ trajectories of the underlying through a 1-step ($n = 1$) Monte Carlo method and compared the difference with a multiple-step Euler-scheme based simulation with same number of generated paths and $n = 252$. The prices of the call are given by the VBA functions `MCDynPrice` and `AveMCDynPrice`, that automatically calculate the discounted payoffs of each trajectory using Equation (4) and perform the average between each computed price. These functions have been written by properly modifying the code taken from source [2].

In order to check the validity of these results, the prices so obtained have been compared with the respective price computed through the Black-Scholes' formula, implemented in function `BSCall`, used also for past reports.

At this point, Asian options have been priced by modifying again the original code, in order to use the price written in Equation (5), which depends on the entire trajectory of the underlying. The VBA functions that have been used for this purpose are named `MCAsianCallPrice` and `MCAsianCallPriceSym`.

Finally, by modifying again the code, the functions `MCLookbackPrice` and `MCLookbackPriceSym` have been written in order to price lookback options, which price is written in equation (6).

RESULTS

Relying on the parameters' values described in Table I, for the underlying we obtained the trajectories shown in Figure 1 and a call option Black-Scholes' price equal to:

$$c_{BS} = 8,92\,\$ \tag{7}$$

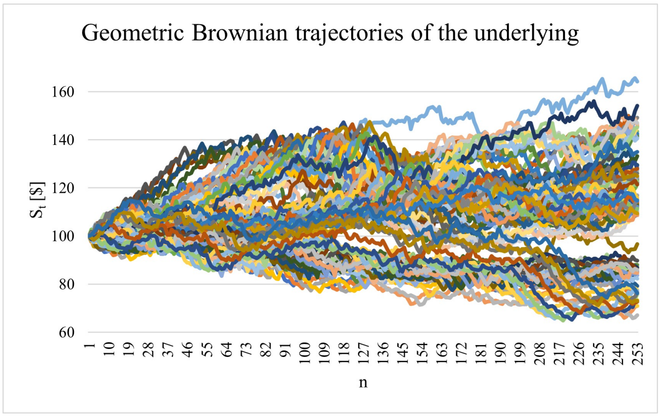

FIG. 1. Plot of $m = 100$ trajectories for the underlying with $n = 252$ discrete time steps.

The Monte Carlo dynamic price of the call, for this limited-sized simulation, is equal to $16,66\,\$$, which is a pretty bad estimation compared to the theoretical result 7.

Increasing the number of simulations to $m = 5000$, the situation significantly improves. The results of 10 runs of the simulations for the various options are listed in Tables II, III.

1-step vanilla [\$]	252-steps vanilla [\$]
8,98	9,10
8,72	9,33
9,31	9,35
9,08	9,43
8,70	9,34
8,74	9,35
8,85	9,33
9,25	9,33
8,88	9,38
9,08	9,22

TABLE II. Results of 10 subsequent 1-step and 252-steps Monte Carlo simulations for vanilla call options.

Asian [\$]	Lookback [\$]
5,91	15,04
5,63	15,12
5,71	15,58
5,63	14,79
5,66	15,13
5,76	15,44
5,73	14,81
5,51	15,11
5,53	15,52
5,66	15,37

TABLE III. Results of 10 subsequent 252-steps Monte Carlo simulations for Asian and lookback options.

CONCLUSIONS

The randomness of the simulated trajectories can be visually checked from Figure 1 by observing the absence of any trend between the single paths, that can thus be considered as independent.

From Table II it can be observed that the prices generated by 1-step and 252-steps Monte Carlo simulations are a bit different and, in particular, the 1-step results are more consistent with the Black-Scholes' result obtained in (7). The reason for this difference may be attributed to the fact that the subdivision of the time to maturity in multiple steps introduces a source of discretisation error in the evaluation of the final call price.

[1] Hull, John C. *Options futures and other derivatives*. Pearson Education India, 2003.
[2] https://stackoverflow.com/questions/43764341/plotting-trajectories-of-gbm-with-vba

VBA CODE

```
Function GBMPath(s As Double, t As Double, z As
    Double, r As Double, q As Double, n As Double)
    As Variant
Dim dt, e, dlns, SimVar() As Double
ReDim SimVar(n + 1)
dt = t / n
SimVar(0) = s
For i = 1 To n
Randomize
   e = WorksheetFunction.NormSInv(Rnd())
   dlns = (r - q - z ^ 2 / 2) * dt + z * e * dt ^
       0.5
       SimVar(i) = SimVar(i - 1) * Exp(dlns)
Next i
GBMPath = SimVar()
End Function

Sub GBMSimulation()
Dim i As Long, j As Long, m As Long, n As Double, s
    As Double, t As Double, z As Double, r As
    Double, q As Double
With Sheets("Sheet1")
   m = .Range("A2").Value
   n = .Range("B2").Value
   s = .Range("C2").Value
   t = .Range("D2").Value
   z = .Range("E2").Value
   r = .Range("F2").Value
   q = .Range("G2").Value
End With
ReDim Arr(m)
For i = 0 To m
   Arr(i) = GBMPath(s, t, z, r, q, n)
Next i
For i = 0 To m
   For j = 0 To n
       Debug.Print Arr(i)(j)
       Cells(i + 4, j + 1) = Arr(i)(j)
   Next j
Next i
End Sub

Function MCDynPrice(s As Double, t As Double, z As
    Double, r As Double, q As Double, n As Double,
    k As Double) As Variant
Dim dt, e, dlns, SimVar() As Double
ReDim SimVar(n + 1)
dt = t / n
SimVar(0) = s
For i = 1 To n
Randomize
   e = WorksheetFunction.NormSInv(Rnd())
   dlns = (r - q - z ^ 2 / 2) * dt + z * e * dt ^
       0.5
       SimVar(i) = SimVar(i - 1) * Exp(dlns)
Next i
MCDynPrice = (WorksheetFunction.Max((SimVar(n) -
    k), 0)) * Exp(-r * t)
End Function
```

```
Sub AveMCDynPrice()
Dim i As Long, j As Long, m As Long, n As Double, s
    As Double, t As Double, z As Double, r As
    Double, q As Double, k As Double, ave As Double
With Sheets("Sheet2")
   m = .Range("A2").Value
   n = .Range("B2").Value
   s = .Range("C2").Value
   t = .Range("D2").Value
   z = .Range("E2").Value
   r = .Range("F2").Value
   q = .Range("G2").Value
   k = .Range("H2").Value
End With
ReDim Arr(m)
For i = 0 To m
   Arr(i) = MCDynPrice(s, t, z, r, q, n, k)
Next i
ave = (WorksheetFunction.Sum(Arr())) / m
Debug.Print ave
Cells(2, 9) = ave

End Sub

Function MCAsianCallPrice(s As Double, t As Double,
    z As Double, r As Double, q As Double, n As
    Double, k As Double) As Variant
Dim dt, e, dlns, SimVar(), aus As Double
ReDim SimVar(n + 1)
dt = t / n
SimVar(0) = s
For i = 1 To n
Randomize
   e = WorksheetFunction.NormSInv(Rnd())
   dlns = (r - q - z ^ 2 / 2) * dt + z * e * dt ^
       0.5
       SimVar(i) = SimVar(i - 1) * Exp(dlns)
Next i
aus = WorksheetFunction.Sum(SimVar()) / (n + 1)
MCAsianCallPrice = (WorksheetFunction.Max((aus -
    k), 0)) * Exp(-r * t)
End Function

Sub MCAsianCallPriceSym()
Dim i As Long, j As Long, m As Long, n As Double, s
    As Double, t As Double, z As Double, r As
    Double, q As Double, k As Double, ave As Double
With Sheets("Sheet4")
   m = .Range("A2").Value
   n = .Range("B2").Value
   s = .Range("C2").Value
   t = .Range("D2").Value
   z = .Range("E2").Value
   r = .Range("F2").Value
   q = .Range("G2").Value
   k = .Range("H2").Value
End With
ReDim Arr(m)
For i = 0 To m
   Arr(i) = MCAsianCallPrice(s, t, z, r, q, n, k)
Next i
ave = (WorksheetFunction.Sum(Arr())) / m
Debug.Print ave
Cells(2, 9) = ave
```

```
End Sub

Function MCLookbackPrice(s As Double, t As Double,
    z As Double, r As Double, q As Double, n As
    Double) As Variant
Dim dt, e, dlns, SimVar(), aus As Double
ReDim SimVar(n + 1)
dt = t / n
SimVar(0) = s
For i = 1 To n
Randomize
    e = WorksheetFunction.NormSInv(Rnd())
    dlns = (r - q - z ^ 2 / 2) * dt + z * e * dt ^
        0.5
        SimVar(i) = SimVar(i - 1) * Exp(dlns)
Next i
aus = SimVar(0)
For i = 1 To n
    If (SimVar(i) < aus) Then
        aus = SimVar(i)
    End If
Next i

MCLookbackPrice = (SimVar(n) - aus) * Exp(-r * t)
End Function

Sub MCLookbackPriceSym()
Dim i As Long, j As Long, m As Long, n As Double, s
    As Double, t As Double, z As Double, r As
    Double, q As Double, ave As Double
With Sheets("Sheet5")
    m = .Range("A2").Value
    n = .Range("B2").Value
    s = .Range("C2").Value
    t = .Range("D2").Value
    z = .Range("E2").Value
    r = .Range("F2").Value
    q = .Range("G2").Value
End With
ReDim Arr(m)
For i = 0 To m
    Arr(i) = MCLookbackPrice(s, t, z, r, q, n)
Next i
ave = (WorksheetFunction.Sum(Arr())) / m
Debug.Print ave
Cells(2, 9) = ave

End Sub

'* Black Scholes call european option *

Function BSCall(Stock As Double, Exercise As
    Double, Rate As Double, Sigma As Double, Time
    As Double) As Double
Dim d1 As Double, d2 As Double
    With Application
        d1 = (.Ln(Stock / Exercise) + (Rate +
            (Sigma ^ 2) / 2) * Time) / (Sigma *
            Sqr(Time))
        d2 = (.Ln(Stock / Exercise) + (Rate -
            (Sigma ^ 2) / 2) * Time) / (Sigma *
            Sqr(Time))
        BSCall = Stock * .Norm_S_Dist(d1, True) -
            Exercise * Exp(-Rate * Time) *
            .Norm_S_Dist(d2, True)
    End With
End Function

'* Black Scholes put european option *

Function BSPut(Stock As Double, Exercise As Double,
    Rate As Double, Sigma As Double, Time As
    Double) As Double
Dim d1 As Double, d2 As Double
    With Application
        d1 = (.Ln(Stock / Exercise) + (Rate +
            (Sigma ^ 2) / 2) * Time) / (Sigma *
            Sqr(Time))
        d2 = (.Ln(Stock / Exercise) + (Rate -
            (Sigma ^ 2) / 2) * Time) / (Sigma *
            Sqr(Time))
        BSPut = Exercise * Exp(-Rate * Time) *
            .Norm_S_Dist(-d2, True) - Stock *
            .Norm_S_Dist(-d1, True)
    End With
End Function
```