**Pricing of a STANDARD LONG BARRIER CERTIFICATES with MONTE CARLO SIMULATION**

The "CQproj\_ws.xlsm" worksheet allows you to price a STANDARD LONG BARRIER CERTIFICATE using the Monte Carlo method, which is implemented in **Excel** and **VBA** (in VBA, I created: “Foglio1 (Barrier Certificate)” to make computation; the module “Genaration\_module” to implement method for generating uniform random numbers; the module “MC\_MeanVar” to compute mean and standard error of the simulation using the Knuth’s algorithm; the module “price\_certificate” for the pricing algorithm) .

A "**Long Barrier Certificate**" is a Bonus certificate which offers the investor the possibility of obtaining an additional return in exchange for assuming the risk of loss of the invested capital. It considers the value of the underlying asset (in our case the BNP PARIBAS SA stock) in a certain period, i.e., the Valuation Period of the Barrier Event. The Barrier event occurs if, at least once during the valuation Period, the value of the underlying goes below the Barrier value; it does not occur otherwise. Consequentially, the value of the Bonus that will be paid upon expiry will be given by:

* (Initial Percentage x Initial Reference Value x Multiple)

if the barrier event does not occur

* Min [Cap Barrier Amount; (Final Reference Value x Multiple)]

if the barrier event occurs

MODEL ASSUMPTIONS

The following simplifications have been adopted for the drafting of this project:

1. The pricing date is the issue date (April 8, 2022) (we want to price the certificates on the issuing date, but of course, it is possible to price it on a random day between the issue date and the expiration date)
2. The issuer is not risk-free (the credit risk is estimated through the Duffie – Singleton model)
3. All days between the issue date and the final valuation date have been counted, with all days counted as working days (to simplify our model)
4. The final valuation date is the same as the expiry date, instead of 4 days before (to simplify our model)
5. The models’ parameters are constant over time (to simplify the model)

THEORETICAL BACKGROUND

* *Linear Congruential Random Number Generator*

The linear congruential generator is a good U (0,1) random number generator that, given a positive integer x and a positive integer divisor m, the remainder of the integer division of x by m, is an ‘unpredictable’ integer between 0 and m - 1. Therefore, we can construct a U (0,1) random number generator using the following idea:

1. Fix two positive integers a and m
2. Fix an integer initialization seed x0
3. Build the recurrent sequence: for each k = 1, 2, ….

Xk = (axk-1) mod m (which represents the remainder of the integer division)

Uk = xk / m

The xks are a sequence of integer numbers in [0, m -1].

The uks are a sequence of real numbers in [0,1].

In our algorithm, we will choose the Park and Miller m and a, with m = 2147483647 and a = 16807.

* *Box – Muller method*

The BM method thakes a sample from a bivariate independent standard normal distribution, each component of wich is thus a univariate standard normal.

The algorithm is based on the following two properties of the bivariate independent standard normal distribution: if Z = (Z1, Z2) has this distribution, then

1. R2 = Z12 + Z22 is exponentially distributed with mean 2
2. Given R2, the point (Z1, Z2) is uniformly distributed on the circle of radius R centered at the origin.

We can use these properties to build the algorithm:

1. Generate independent U1, U2 ∼ U (0,1)
2. Transform U1 in an exponential sample by R2 = -2log(U1)
3. Transform U2 in a random uniform angle between 0 and 2𝛑 by 𝛂 = 2𝛑 U2
4. The corresponding point on the circle of radius R centred at the origin has coordinates Z1 = Rcos 𝛂 and Z2 = Rsin 𝛂

If Z1 and Z2 are two independent N (0,1) variates (e.g., generated by the Box- Muller algorithm), then by combining them in a deterministic way we get a unique variate form a N (0,1).

Therefore, the Box – Muller algorithm can be used in two ways:

* If we need a bivariate N (0,1) independent sample, then we use it in the standard way
* If we need a univariate N (0,1) sample, we combine the output
* *Cholesky Factorization*

A multivariate normal random variable Z ∼N (𝛍, 𝜮) is completely defined by its vector of means and its covariance matrix 𝜮.

For the linear transformation property, given a m x m matrix A, the linear transformation X = AZ is a multivariate normal, with mean vector A 𝛍 and covariance matrix A 𝜮AT. We can use this property to transform an independent multivariate normal Z ∼N (0, Im) (where Im is the identity m x m matrix) into the desired X ∼N( 𝛍, 𝜮). If A is a m x m matrix, by the linear transformation property AZ has mean 0 and covariance matrix A Im AT = AAT. So, our task is to find a matrix A satisfying AAT = 𝜮. Once found, we simulate Z and transform it into the desired X by X = 𝛍 + AZ.

Recall that 𝜮 is a strictly positive defined matrix, then we can use Cholesky factorization, that gives us a lower triangular matrix A such that AAT = 𝜮.

Because of lower triangularity of A, the final transformation X = 𝛍 + AZ becomes

X1 = 𝛍1 + a11Z1

X1 = 𝛍1 + a21Z1 + a22Z2

…

Xm = 𝛍m + am1Z1 + am2Z2 + … + + ammZm

Hence, we obtain the component of X by a transformation of the first component of Z, the second by a transformation of the first two components of Z and so on.

* *Stock evolution*

In our model we assume that the underlying stock price is log-normally distributed and follows a Geometric Brownian Motion under the probability measure Q. For this reason, for the adoption of the Euler scheme, we use the logarithmic values instead of the normal ones for our values (log-linearization), for example for the value of the underlying L and for that of the strike/barrier K. Then, if L = logS, with S as the price of the underlying stock, we have that:

dL(t) = [ r – 0.5V(t)] dt + √[V(t)] dWs(t)

where

L(t) = log-value of S at time t

r = interest rate

dt = time step

dWs(t) = differential of the Wiener process that drives S

V(t) = squared volatility

* *Volatility evolution*

In our model, we assume that the volatily that drives the stock process is not constant, but it is driven by a stochastic process that varies over time following the Heston model. In particular, the variation of the squared volatility V(t) is given by:

dV(t) = K [θ – V(t)] dt + Λ √[V(t)] dWv(t)

where

V(t) = squared volatility at time t

K [θ – V(t)] = mean reversion factor

Θ = long term expected squared volatility

K = speed of mean reversion

Λ = volatility of volatility

dWv(t) = differential of the Wiener process that drives V

* *Interest rate evolution*

In our model, we assume that the interest rate, used for the stock Q-dynamics and for the discount factor, follow a stochastic process. In particular, the interest rate process can be defined by

* + Vasicek model: Q-dynamics given by

dr(t) = ɑ [𝝲 – r(t)] dt + σ dWr(t)

where

r(t) = interest rate at time t

ɑ = speed of mean reversion

𝝲 = long term interest rate

σ = volatility of interest rate

dWr(t) = differential of the Wiener process that drives r

* + CIR model: Q-dynamics given by

dr(t) = ɑ [𝝲 – r(t)] dt + σ √[r(t)] dWr(t)

* *Credit risk evolution*

In our model, we assume that the certificate issuer is not risk-free, then we are subject to credit risk, whose evolution is driven by the Duffie-Singleton model as follows:

dλ(t) = a [b - λ(t)] dt + c √[λ(t)] dWλ(t)

where

λ(t) = credit risk value at time t

a = speed of mean reversion

b = long term credit risk value

c = volatility of credit risk

* *Euler scheme*

To simulate the value of the stock, the interest rate, the volatility, the credit risk and then the value of our certificate, we adopt the Euler scheme. Within this framework, each (one-dimensional stochastic) process is discretized and evaluated at each time step “dt”, and jointly simulated, to consider the eventual correlations between the Wieners that drives all the processes. The scheme uses the same idea of the first order Taylor expansion for deterministic processes, and then it converges to the” exact solution” as dt goes to 0. The approximation of the Euler scheme induces an approximation error on the final Monte Carlo estimate. As in the case of the approximation of the stochastic discount factor by the trapezoidal rule, the approximation error is difficult to estimate, and it interacts with the Monte Carlo error. Thus, the Euler approximation is a source of bias on the standard Monte Carlo estimation error.

* *Knuth algorithm*

Knuth's numerically stable one-pass algorithm is a method for computing the mean and variance of a set of numbers in a single pass, without storing all the individual values. This algorithm is particularly useful in Monte Carlo simulations, where large numbers of random samples are generated and statistical measures like mean, and variance need to be computed efficiently. The key idea behind the algorithm is to update the mean and variance incrementally as new values are encountered, rather than storing all the values and calculating the statistics at the end.

Here is a step-by-step description of Knuth's algorithm:

1. Initialize the variables:

- `k` to 1 (number of samples encountered so far)

- `mean` to 0 (running mean)

- `numVar` to 0 (sum of squares of differences from the mean)

2. For each new sample value `x`:

- Increment `k` by 1 (this is done automatically in the main “for loop” in the “price\_certificate” module when the function is called).

- define `delta` as `x - mean`.

- define `mean` as `mean + delta /k`.

- define ` numVar ` as ` numVar + delta \* (x - mean) `.

3. At the end of the simulation, you can compute the mean (which, in our case, represents the price of the certificate) and the standard error (the square root of the variance) using the following formulas:

- Mean (= price): `mean`

- standard error: `√ ((numVar / (n - 1)) / n) `

This algorithm has several advantages. It is computationally efficient because it requires only a single pass through the data, and is also numerically stable, meaning that it avoids some of the numerical precision issues that can arise when computing variance using traditional methods.

By using Knuth's algorithm, you can progressively update the mean and variance as new samples are generated in a Monte Carlo simulation, without needing to store all the individual values. This makes it a valuable tool for efficient and accurate statistical analysis in such simulations.

COMMENT ON THE RESULTS

|  |  |  |
| --- | --- | --- |
| Simulations | Price | Std Error |
| 10 | 108,9633 | 0,33 |
| 100 | 102,1431 | 2,0123 |
| 1000 | 102,3236 | 0,6139 |
| 10000 | 102,3885 | 0,1906 |
| 100000 | 102,7134 | 0,0579 |
| 200000 | 102,6828 | 0,0444 |
| 300000 | 102,6962 | 0,0336 |

We ran the algorithm with an increasing number of simulations, starting from 10 simulations up to 300,000 simulations. As we can see from the results obtained in the table above, keeping fixed the values of the parameters defined in "Barrier Certificate" of the “CQproj.xlsm” work sheet, the price of the certificate tends to converge around a value of 102.7, as the simulations increase. Of course, the results obtained also depend, for example, on the values attributed to the various parameters, on the randomness of the random variables, on the correlation between the Wieners’ and on the volatility of each model, but we will comment only with respect to the Monte Carlo simulation process. It should be noted that, with a relatively low number of simulations (i.e., below 1000), the values obtained are of little significance. With 100 simulations, for example, we obtain a value that deviates by 0.5% from the value of 102.7, but with a simulation error of 201%. By progressively increasing the number of simulations, it is possible to observe how the price tends to converge towards a certain value, and at the same time the error tends to decrease, reaching an error of 3.36% with 300,000 simulations. Clearly, this implies that the value of the error will further tend to decrease as the number of simulations increases and the price will get closer and closer to its "exact" value. This demonstrates the validity of the central limit theorem: the greater the number of simulations, the more precise and closer to the true value will be the result obtained. This means that by repeating the process many times, you will get results that are normally distributed around the true value, which we could call, 𝛍.