mc_baldi_in

Input parameters:

- Number of iterations N
- Generator_Type
- TimeStep Number M
- Increment inc
- Confidence Value

Output parameters:

- Price P
- Error Price σ_P
- Delta δ
- Error delta σ_δ
- Price Confidence Interval: IC_P =[Inf Price, Sup Price]
- Delta Confidence Interval: IC_δ =[Inf Delta, Sup Delta]

Description:

Computation for a Knock-In Barrier Option, Call or Put - Up or Down - In, of its Price and its Delta with a Monte Carlo or Quasi-Monte Carlo method. In the case of Monte Carlo simulation, the algorithm also provides an estimation for the integration error and a confidence interval.

The underlying asset price evolves according to the Black and Scholes model, that is:

$$dS_u = S_u((r-d)du + \sigma dB_u), \quad S_t = s$$

then

$$S_T = s \exp\left((r - d - \frac{\sigma^2}{2})\theta\right) \exp(\sigma B_\theta)$$

where S_T denotes the spot at maturity T, s is the initial spot, θ is the time to maturity.
For the barriers we use the following notations:
L and U denote respectively the lower and upper barriers.

$$\tau_L = \inf\{u > t; S_u \leq L(u)\}$$

$$\tau_U = \inf\{u > t; S_u \geq U(u)\}$$

are the hitting time on the barriers.

Remarks: In PREMIA we only consider constant barriers. But following algorithms could be adapted to time-dependent barriers.

The Price of a single barrier option at t is:

$$P = E\left[\exp(-r\theta) f(S_T, K, R, \tau)\right]$$

where f denotes the payoff of the option, R the rebate and τ the hitting time on the barrier.
The Delta is given by:

$$\delta = \frac{\partial}{\partial s} E[\exp(-r\theta) f(S_T, K, R, \tau)]$$

Estimators are expressed as:

$$\widetilde{P} = \frac{1}{N} \exp(-r\theta) \sum_{i=1}^{N} P(i)$$

where $P(i) = f(S_T(i), K, R, \tau(i))$

$$\widetilde{\delta} = \frac{1}{N} \exp(-r\theta) \sum_{i=1}^{N} \frac{\partial}{\partial s} P(i) = \frac{1}{N} \exp(-r\theta) \sum_{i=1}^{N} \delta(i)$$

The values for $P(i)$ and $\delta(i)$ are detailed for each option.

- **Down-In Put**: The payoff is $(K - S_T)^+ 1_{\tau_L \leq T} + R 1_{\tau_L > T}$.
- **Up-In Put**: The payoff is $(K - S_T)^+ 1_{\tau_U \leq T} + R 1_{\tau_U > T}$.

- **Down-In Call**: The payoff is $(S_T - K)^+ 1_{\tau_L \leq T} + R 1_{\tau_L > T}$.
- **Up-In Call**: The payoff is $(S_T - K)^+ 1_{\tau_U \leq T} + R 1_{\tau_U > T}$.

Algorithm:

This algorithm is taken from [2] and allows to numerically compute the price and the delta of single Knock-In Barrier Options with a Monte Carlo method. The issue, as it is discussed in there, is to provide a good approximation of the first time τ at which the price of the underlying stock reaches the barrier. If such a time is observed to be less or equal to the maturity, the option is activated, its value being equal to a pre-specified rebate otherwise. One could numerically determine the first time at which the stock price is observed to cross the barrier by a crude simulation, i.e. through $k^* \cdot h$, where h stands for the time step increment and k^* denotes the first step the underlying asset price has been outside the boundary (here, it is supposed that 0 is the starting time). Numerical tests show that this method does not perform well because the stock price is checked at discrete instants through simulations and the barrier might have been hit without being detected, giving rice to an over-estimation of the exit time and thus to a non trivial error for the estimate of the option price.

The algorithm (there) from [1] allows to improve the performance of the crude Monte Carlo method, by giving a careful estimation of τ as follows. When the stock price is observed to stay inside the boundary either at step $k-1$ and k, an accurate approximation p_k^h of the probability that the underlying asset price crosses the barrier during the time interval $((k-1)h, kh)$ is computed and a bernoulli r.v. with parameter p_k^h is generated: if it is observed to be equal to 1, then the process is supposed to have gone out, so that the exit time can be approximated by kh, otherwise the $(k+1)^{\text{th}}$ step is considered, unless $k = M$, i.e. the maturity has been reached.

♣ **Function exit-probability-in**

/* Compute the probability that the spot has crossed the barrier during the time interval */

For a constant barrier, we can compute exactly this probability through the conditional law of a Brownian Bridge.

Probability that the spot crosses the barrier is the same that the maximum (minimum) of B_u over $[t_k, t_{k+1}]$ conditionally to B_k and B_{k+1} is greater (smaller) than the barrier.

The law of the maximum (minimum) is given by: (see Lookback options)

♣ Function MC-InBaldi-97

/* Value to construct the confidence interval */
For example if the confidence value is equal to 95% then the value z_α used to construct the confidence interval is 1.96. This parameter is taken into account only for MC simulation and not for QMC simulation.

/*Initialisation*/
The variables giving the price, the delta and the corresponding variances are initialised. The coefficients `z`, `rloc`, `sigmaloc` and `sigmat` are used in order to generate the the underlying asset prices starting at s and $s + \varepsilon$, at the discretisation times.
/* Maximum Size of the random vector we need in the simulation */
The size equals $2M$. This parameter is used in case of QMC simulation.
Justification
/*Coefficient for the computation of the exit probability*/
The constant `rap` is used to compute the local probability of exit from the barrier.

- /*MC sampling*/

Initialization of the simulation: generator type, dimension, size N of the sample.
/* Test after initialization for the generator */
Test if the dimension of the simulation $2M$ is compatible with the selected generator. (See remarks on QMC simulation, especially on dimension of low-discrepancy sequences). For Monte Carlo simulation, we never have any problem with the dimension because successive random numbes from generators are independent.
Definition of a parameter which exprimes if we realize a MC or QMC simulation. Some differences then appear in the algorithm for simulation of a gaussian variable and in results in the simulation.

/* MC simulation */
/* Begin N iterations */

- In this cycle, at step i the paths $\ln S^{(i)}(s)$ and $\ln S^{(i)}(s+\varepsilon)$, starting at s and $s + \varepsilon$, are simulated. Thus, it starts by initialising the variable `time` giving the current value of the discretization time. Since the paths really simulated are given by the logarithm of the underlying asset price starting at s and $s + \varepsilon$, their current values are set in the variables `lnspot` and `lnspot_increment`.
Notice that the process starting at $\ln(s + \varepsilon)$ is equal to the process starting

at $\ln s$ added by $\ln(1+\varepsilon/s)$, which is a constant denoted as `increment`.
/*Up and Down barrier at time */
Since the paths really simulated are given by the logarithm of the underlying asset price, the considered barrier is the logarithm of the starting barrier `l`.
/*Inside = 0 if the path reaches the barrier*/
`inside` and `inside_increment` are boolean variables initialised to 1, switching to 0 when the corresponding path is observed to exit from the barrier.

-/*Simulation of the i-th path until its exit if it does*/
In this cycle, the processes are both simulated at the discretisation times kh, whose current name is `time`, until $k = M$ or the corresponding value of the flag is changed, i.e. until `inside`$= 0$ or `inside_increment`$= 0$.
At each step k, a variable, called `correction_active`, is introduced in order to ensure that both paths are generated by means of the same sample. `correction_active` is firstly equal to 0 and its value switches to 1 whenever a path is observed to exit whereas the other one does not behave in the same way.
Values of the old and new simulated points and of the barrier are put in the variables `lastlnspot`, `lnspot`, `lastlnspot_increment`, `lnspot_increment`, `lastbarrier`, `barrier` respectively .

/* Simulation of a gaussian variable according to the generator type, that is Monte Carlo. */
Call to the appropriate function to generate a standard gaussian variable. See the part about simulation of random variables for explanations on this point. We just recall that for a MC simulation, we use the Gauss-Abramovitz algorithm.

/*Check if the i-th path has reached the barrier at time*/
- The variable `upordown` is defined to be equal to 0 if the considered barrier is an upper one, `upordown` being equal to 1 in the case of a lower barrier.
- `lnspot` is compared with `barrier`: if the path is outside the barrier, the corresponding value of `inside` is set equal to 0 and the exit time `exit-time` set as the current time.
- Otherwise, we need to simulate whether `lnspot` has crossed the barrier between time and time-1.
Rejection method : We take an uniform variable u and we compare it to the exit probability p computed with the function exit-probability.
$u < p$ means that the barrier was crossed, and then `inside`is set to 0.
$u > p$ that the barrier was not crossed.

If u was simulated, `correction_active` is set equal to 1, and thus u will not be simulated an other time for the incremented path.

- The same algorithm is applied to the incremented path with only one difference: if u was already simulated for the spot path, we use the same value, else we now generate u.

/*Inside=0 means that the payoff does not nullify
Inside=1 means that the payoff is equal to the rebate*/
- We introduce the variable `time-now` which defines at what time we quit the previous cycle. `lnspot` and `lnspot_increment` were both simulated until this time.
`time-now` is theoretically the same that `time` at the exit of the cycle, but if $k = M$ we have that $M * h$ is not exactly t because of computing error whithin iterations.
- We now compute `price_sample` and `price_sample_increment` for this cycle.
The i^{th} path has been generated until its exit, if it has done, or $k = M$, so that the price provided by the sample can be computed.
If `inside`= 0 then the boundary has been reached at exit-time.
If this is not the case, the following property is used:

$$\mathbb{E}_{0,s}[e^{-rt} f(S_t)\, \mathbf{1}_{\tau \leq t}] = \mathbb{E}_{0,s}\Big[e^{-rt'}\, \mathbf{1}_{\tau \leq t} \mathbb{E}_{t',S_{t'}}[e^{-r(t-t')} f(S_t)]\Big]$$

where $f(x)$ denotes $(x-K)_+$ or $(K-x)_+$ according to the case of a call or put option respectively. Since $\mathbb{E}_{u,S_u}[e^{-r(t-u)} f(S_t)]$, with $u < t$ and $S_u > 0$ denotes the price of a standard call/put option (without barriers), it can be exactly computed by means of closed formulas (Black and Scholes), so that `price_sample` is set equal to this quantity evaluated in `time`now and exp(lnspot).
Instead if `inside` is equal to 1, i.e. the path has never gone out, `price_sample` becomes equal to the rebate, denoted as `rebate`, discounted by `\exp(-r*t)`.
- By using a similar procedure, `price_sample_increment` is computed.

- /*Delta*/
The delta of the sample $\delta(i)$ is computed (recall that `increment`= $\ln(1+\varepsilon/s)$ so that $\varepsilon \sim$ `increment*s`:that is why the variation of the price sample is divided by `increment*s`).

- /*Sum*/
The partial sums of the observed `price_sample` and `delta_sample` are com-

puted.

-/*Sum of Squares*/

The partial sums of the squares of the observed `price_sample` and `delta_sample` are computed and will be used to evaluate the empirical variances.

/* End N iterations */

- /*Price*/

The price estimator is:

$$P = \frac{1}{N} \sum_{i=1}^{N} P(i)$$

The error estimator is σ_P with :

$$\sigma_P^2 = \frac{1}{N-1} \left(\frac{1}{N} \sum_{i=1}^{N} P(i)^2 - P^2 \right)$$

- /*Delta*/

The delta is computed according to the case of a put or call option.

$$\delta = \frac{1}{N} \sum_{i=1}^{N} \delta(i)$$

The error estimator is σ_δ with:

$$\sigma_\delta^2 = \frac{1}{N-1} \left(\frac{1}{N} \sum_{i=1}^{N} \delta(i)^2 - \delta^2 \right)$$

- /* Price Confidence Interval */

The confidence interval is given as:

$$IC_P = [P - z_\alpha \sigma_P; P + z_\alpha \sigma_P]$$

with z_α computed from the confidence value.

- /* Delta Confidence Interval */

The confidence interval is given as:

$$IC_\delta = [\delta - z_\alpha \sigma_\delta; \delta + z_\alpha \sigma_\delta]$$

with z_α computed from the confidence value.

/* QMC simulation */

We just give a description of the points which differ from MC simulation. Otherwise algorithm follows the same steps.

/* Begin N iterations */

/* index is used to select value in the random vector */

We use the variable `index` to keep successive random values in the simulated vector.

/* Simulation of a uniform vector of size simulation-dim*/

/* For QMC simulation, we have to simulate the full vector at the beginning of each cycle i even if we will not use all the values in the case where we reach the barrier before maturity */

Independent uniform variables are necessary to simulate the path and the crossing of the barrier, then we need the next point for each dimension of a multi-dimensional low-discrepancy at each step i.

/* The first uniform value will be used to generate a gaussian variable */

/*Simulation of i-th path until its exit if it does*/

/* Simulation of a gaussian variable according to the QMC generator.*/

/* Uniform value comes from the uniform vector generated at the beginning of each cycle i*/

We recall that we use an inverse function for QMC simulation. We use the first coordinate of the selected low-discrepancy sequence.

The algorithm goes on as for a MC simulation, except for simulation of the crossing of the barrier: we keep the next (with `index`) uniform value in the vector previously obtained.

We don't compute variance and confidence interval because they don't work for QMC simulation.

References

[1] P.BALDI L.CARAMELLINO M.G.IOVINO. Pricing complex barrier options with general features using sharp large deviation estimate. *Proceedings of the MCQMC Conference, Calremont (LA), USA*, 1999. 3

[2] P.BALDI L.CARAMELLINO M.G.IOVINO. Pricing general barrier options: a numerical approach using sharp large deviations. *Mathematical Finance*, 9(4), 1999. 3