mc_parisianin

This algorithm is taken from [1] and allows to numerically compute the price and the delta of double Knock-In Parisian Barrier Options with a Monte Carlo method. The issue, as it is discussed in there, it to provide a good approximation of the first time H_D at which the price of the underlying stock stays outside the barriers uninterruptedly for longer than a pre-specified delay D. If such a time is observed to be less or equal to the maturity, the option is activated, it is nullified otherwise. One could numerically determine how much time the stock price is observed to stay outside the barrier by a crude simulation, i.e. through $k^* \cdot h$, where h stands for the time step increment and k^* denotes the number of consecutive times the underlying asset price has been outside the same barrier. Numerical tests show that this method does not perform well because the stock price is checked at dicrete instants through simulations and the barrier might have been hit without being detected, giving rice 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 H_D as follows. Setting g_u as the final time before u when a barrier is hit, with the constraint $g_u = u$ if no barrier is crossed before u, H_D turns out to be the first instant u such that $u - g_u \geq D$. Now, when the stock price is observed to stay outside a barrier either at step $k-1$ and k, an accurate approximation p_k^h of the probability that the underlying asset price comes back to the domain 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 g is updated and set equal to hk, otherwise the value of g does not change.

/*One forces N if necessary so that delay
!!!!!!!!! WARNING !!!!!!!
be greater than the time step increment h*/

The time step increment h is initialised; since the value D of the delay has to be greiter than h, when $D \leq h$ the number N of the dicretisation points of the time interval $[0, t]$ (t standing for the maturity date) is increased.

/*Initialisation*/
The variables giving the price, the delta and the corresponding variances are initialised. Moreover, since the path really simulated are given by the logarithm of the underlying asset price starting at s and $s+\varepsilon$, the considered starting points are actually the logarithm of the starting points. The coefficients `rloc` and `sigmaloc` are used in order to generate the processes at the discretisation times. Finally, 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`.

/*Coefficient for the computation of the exit probability*/
The constant `rap` is used to compute the local probability of crossing the barrier, which turns out to be an exit probability from a barrier.

/*MC sampling*/
In this cicle, 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 variables `gt`, `hd`, `gt_increment,hd_increment` and `lnspot`, giving the current values for g, H_D and the current position `lnspot` of the process.

/*Inside=0 if the path stays beyond the barrier*/
/*uninterruptedly for longer than delay*/
`inside` and `inside_increment` are boolean variables initialised to 1, switching to 0 when the corresponding value of H_D is greater than D, i.e. when the path stays beyond a barrier uninterruptedly for longer than delay.

/*Barrier at time*/
The upper and lower barriers are evaluated at time 0 in `up` and `low` respectively.

/*Simulation of i-th path until Inside=0*/
In this cicle, both processes are simulated at the discretisation times kh, whose current name is `time`, until $k=N$ 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 stay locally beyond the barriers whereas the other one does not behave in the same way. The value of the old and new simulated points and of the barriers are put in the variables `lastlnspot`, `lnspot,lastlnspot_increment`, `lnspot_increment,lastup`, `up`, `lastlow`, `low` respectively.

/*Check if the i-th path has reached the barrier at time*/
/*Otherwise there is no extinction*/

The goal is to estimate the current value for `gt` and `gt_increment`, by making use of the procedure summarised at the beginning, so that the payoff can be computed. Obviously, it is meaningful only if `inside` and `inside_increment` are not changed, i.e. equal to 0, otherwise the delay has been reached and the value of the option is known.

Therefore, suppose that the condition "`if inside`" holds. First of all, `lnspot` and `lastlnspot`, giving the simulated values of $\ln S^{(i)}_{kh}(s)$ and $\ln S^{(i)}_{(k-1)h}(s)$ respectively, are compared with the (logarithm of the) upper barrier: if `lnspot`$>$ `up` then

– if also `lastlnspot`$>$ `up` then the probability p^h_k is computed, `correction_active` is set equal to 1 and a bernoulli r.v. with parameter p^h_k is considered: a uniform r.v. `uniform` is generated and if `uniform`$< p^h_k$ then (the path has gone back and) `gt` is updated as the current `time`; if `uniform`$\geq p^h_k$ then (the path has never gone back and) `gt` is not changed;

– if `lastlnspot`$\leq$ `up` then the final time `gt` at which the barrier has been crossed is approximated through a suitable instant between kh and $(k-1)h$ (see Monte Carlo for Barrier Option : Algorithm for details).

The same procedure applies to `lnspot_increment` and `lastlnspot_increment`. It is worth to observe that if `lnspot_increment` and `lastlnspot_increment` are both observed to stay above the barrier then a uniform r.v. is needed; since the paths have to be simulated by means of the same sample, such a uniform r.v. must be taken as the same `uniform` used to compute `gt`, if it has been generated. Thus, if the condition "`!*correction_active`" is true, which means that `correction_active`$\neq 1$, a new uniform r.v. is considered; whereas if it is false, i.e. the correction has been yet activated, `uniform` does not change and turns out to be the same one which has been previously generated. Notice that, since $\varepsilon > 0$, `lnspot`$<$ `lnspot_increment` and `lastlnspot`$<$ `lastlnspot_increment`, so that `correction_active` has to be firstly activated to the path starting at s.

The comparisons are now made with the lower barrier, in a similar way. Indeed, the same procedure is applied by changing the role to `lnspot` and `lastlnspot` and to `lnspot_increment` and `lastlnspot_increment`, because in such a case `correction_active` has to be firstly activated to the path starting at $s + \varepsilon$.

Finally, if `lnspot`$\in$ (`low`, `up`), then `gt` is updated and becomes equal to `time`; similarly, if `lnspot_increment`$\in$ (`low`, `up`), then `gt_increment` is set equal to `time`. Once these values are known, `hd` and `hd_increment` can be computed and compared with `delay`: if they are greiter than `delay`, the corresponding values of `inside` and `inside_increment` change and the payoffs are computed in `price_sample` and `price_sample_increment`. If at the end of the cicle, `inside` and `inside_increment` are equal to 0, then `price_sample` and `price_sample_increment` are set equal to 0.

/*Delta*/
The delta of the sample 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 computed.

/*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.

/*Price*/
The price is numerically computed by averaging over the M observed `price_sample`. The variable `pterror_price` is such that the interval (`ptprice`$-$ `pterror_price`, `ptprice`$+$ `pterror_price`) represents the 95% confidence interval for `ptprice`.

/*Delta*/
The delta is computed according to the case of a put or call option. The variable `pterror_delta` is such that the interval (`ptdelta`$-$ `pterror_delta`, `ptdelta`$+$ `pterror_delta`) represents the 95% confidence interval for `ptdelta`.

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. 1