Source | Model | Option
| Model_Option | Help on mc methods | Archived Tests

mc_outbaldi

This algorithm is taken from [1] and allows to numerically compute the price and the delta of double Knock-Out 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 barriers. If such a time is observed to be less or equal to the maturity, the option is nullified, being equal to a pre-specified rebate, and is activated otherwise. One could numerically determine the first time at which the stock price is observed to cross the barriers 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 dicrete instants through simulations and the barriers 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 a 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 $k = N$, i.e. the maturity has been reached.

/*Initialisation*/

The variables giving the price, the delta and the corresponding variances are initialised. The coefficients `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.

/*Coefficient for the computation of the exit probability*/
The constant `rap` is used to compute the local probability of exit from the barriers.

/*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 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 barriers are set in the variables `up` and `low` as the the logarithm of the starting upper and lower barrier respectively.

/*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 barriers.

/*Simulation of the i-th path until its exit if it does*/
In this cicle, the processes are both 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`$=0$. 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 barriers at time*/
If the paths starting at s and $s+\varepsilon$ have not yet reached the boundary, i.e. the corresponding value of `inside` and `inside_increment` are equal to 0, `lnspot` and `lnspot_increment` are compared with the barriers: if the path is outside the barriers, the corresponding value of `inside` and `inside_increment` is set equal to 0 and the exit times turns out to be equal to `time`. Moreover, in such a case the price of the samples, `price_sample` and `price_sample_increment`, are set equal to `rebate`, discounded by `\exp(-r*time)`.

/*Check if the i-th path has reached the barriers during (time-1, time)*/
If "`((inside)&&(inside_increment))`" is true, no path has reached the boundary. In such a case, the local exit probabilities `proba` and `proba_increment` are computed by means of `proba_barrierout` and a uniform r.v. `uniform` is generated: if `(uniform<proba)` and/or `(uniform<proba_increment)` then (the path has gone out, so that) `inside` and/or `inside_increment` becomes equal to 0 and `price_sample` and/or `price_sample_increment` are set equal to `rebate`, discounded by `\exp(-r*time)`.
If "`((inside)&&(!inside_increment))`" is true, the path starting at s has not reached the boundary whereas the path starting at $s+\varepsilon$ had. Thus, the local exit probability `proba` is computed by means of `proba_barrierout` and a uniform r.v. `uniform` is generated: if `(uniform<proba)` then (the path has gone out, so that) `inside` becomes equal to 0 and `price_sample` set equal to `rebate`, discounded by `\exp(-r*time)`.
If "`((!inside)&&(inside_increment))`" is true, the path starting at s has reached the boundary whereas the path starting at $s+\varepsilon$ had not. Thus, the local exit probability `proba_increment` is computed by means of `proba_barrierout` and a uniform r.v. `uniform` is generated: if `(uniform<proba_increment)` then (the path has gone out, so that) `inside_increment` becomes equal to 0 and `price_sample_increment` set equal to `rebate`, discounded by `\exp(-r*time)`.
At the end of the `while`-cicle, if `inside` and/or `inside_increment` are not changed, then the path has not reached the boundary: the option is activated and `price_sample` and/or `price_sample_increment` can be computed as usual.

/*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 general barrier options: a numerical approach using sharp large deviations. *To appear in Mathematical Finance (1999)*, 1999. 1