Report HW 1 – Computational Finance

Carlos J Arguello. cja2119

Algorithms to implement the pricing schemes:

General Approach:

- I defined 2 classes: Closed Formula Pricing (function member: Black Scholes), and Monte Carlo Pricing (function members: Closed formula, euler and Milstein functions contained in that class).
- For the Gaussian cdf and the random number generation I used gsl (gnu scientific libraries).

Closed Formula Pricing:

a. Black Scholes Evaluation: Not much to be said here, other than for the cumulant density functions I used gsl functions and libraries. It receives the necessary parameters and calculates Call and Put European options.

Monte Carlo pricing:

b. Closed formula: I used the analytical solution to the Geometrical Brownian Motion:

$$S(T)=So*\exp(-(r-d-\sigma^2/2)T+\sigma W(T))$$
$$W(T)=\text{sqrt}(T)*\mathbf{N(0,1)}$$

For **N(0,1)** I used a gsl function that generates pseudorandom numbers with a standard Gaussian distribution. I evaluate S(T) with as many paths as required. For the call option if S(T)-K > 0, I add it to a accumulating variable and then take the average. For the put call I use if K-S(T)> 0 instead.

c. Euler: I use the following dynamics:

$$dW(t) = \text{sqrt}(\Delta t) *\mathbf{N(0,1)}$$

$$dS(t_i)=S(t_i)((r-d)t_i+\sigma dW(t))$$

$$S(t_{i+1})=S(t_i)+dS(t_i)$$

And then if S(T)-K >0, add it to an accumulator, take the average and multiply by the discount rate: exp(-(r-d)T)

In my code, this function also contains the option to calculate log Spot:

$$dW(t) = \text{sqrt}(\Delta t) *\mathbf{N(0,1)}$$

$$d(\log S(t_i))= ((r-d- \sigma^2/2)t_i+\sigma dW(t))$$

$$\log S(t_{i+1}) = \log S(t_i) + (d \log S(t_i))$$

The algorithm is the same than in the Euler spot (obviously exponentiation is required to get rid of the log).

d. Milstein: I use the following dynamics:

$$dW(t) = \text{sqrt}(\Delta t) * \mathbf{N(0,1)}$$
$$dS(t_i) = S(t_i)((r-d)t_i + \sigma dW(t) + \sigma^2/2\,(dW(t)^2 - \Delta t)$$
$$S(t_{i+1}) = S(t_i) + dS(t_i)$$

So the algorithm is exactly the same, with the added quadratic term for the Wiener process.

Benchmarking:

With the parameters given in the Problem set I get (For the call option):

Trial/Method	Black S. (From my code)	Closed Formula/Error	Euler/Error	Log-Euler/Error	Milstein/Error
10000 paths 252 time steps	9.0571	9.354/3.2 %	9.2564/ 2.19%	9.2581/ 2.2%	9.266 /2.3%
20000 paths 252 time steps	N/A	9.2237 / 1.8%	9.158/ 1.02%	9.158	9.161/ 1.05%
50000 paths 252 time steps	N/A	9.147	9.1739/ 1.29%	9.1730/ 1.27%	9.174/ 1.29%

Convergence is slow with increasing number of paths as can be seen here. For the methods that rely on the dynamics of the process (Euler, Log-Euler, Milstein), there is a substantial improvement with twice the number of paths. The error increased slightly with 50000 paths(?) in the Euler and Milstein based methods. I think this is a product of error accumulation per iteration.

- **Is it necessary to compute the full paths?**

It depends whether a closed solution for the SDE of a given process exists or no. In the case of Geometrical Brownian Motion for example, since the SDE can be solved analytically, the computation of the full path is not necessary.

In cases where the dynamics of the process makes impossible to find an analytic solution, the price of the financial instrument can be found by computing full paths or solving the associated PDE.

If the payoff is one of a continuously monitored call option, then it depends on the path. And if the instrument depends on the path, then there is no alternative but to compute full paths.

Compared with the Excel spreadsheet by Gaarder Haug, My Black S. pricing is correct (the implementation of the