COMPUTATIONAL FINANCE PROJECT1

Kai Chen, Rutgers University 01/30/2018

Introduction

This paper reports the main work of Project1 for Computational Finance (Math 16:642:623). In this project, some classical numerical methods are implemented by Monte Carlo simulation to approximate the price of a European-style call and put option. Numerical solutions are then compared with the closed-form solution obtained from Black–Scholes–Merton formula and a benchmark given by the Excel-VBA spreadsheets of Haug.

Preliminary

In this project, we assume that the stock price is geometric Brownian motion and all the needed variables are constant:

- Volatility $\sigma = 0.3$
- Initial asset price $S(0) = 100$
- Risk-free interest rate $r = 0.05$
- Dividend yield $d = 0.02$
- Strike $K = 110$
- Maturity $T = 1$ year

Closed-form Formula

Since dividend paying is considered when pricing the option, we use the generalized Black-Scholes-Merton pricing formula. The dividend payment paid over the time period $[t, t + dt]$ is modeled as $dS_t dt$, where S_t is the generated stock price process at time t for some constant dividend yield d.

Then the arbitrage-free call option price implied by Black-Scholes-Merton model is:

$$C(S_t, t) = e^{-r(T-t)}[S_t e^{(r-d)(T-t)} N(d_1) - KN(d_2)]$$

where,

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}[\ln(\frac{S_t}{K}) + (r - d + \frac{1}{2}\sigma^2)(T-t)]$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

The put option price is given by:

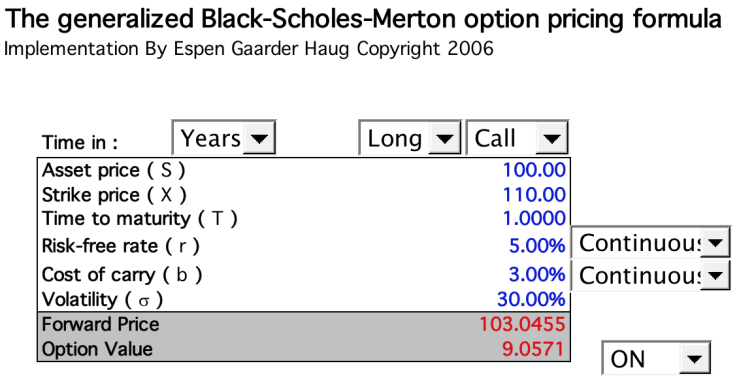
$$P(S_t, t) = e^{-r(T-t)}[KN(-d_2) - S_t e^{(r-d)(T-t)} N(-d_1)]$$

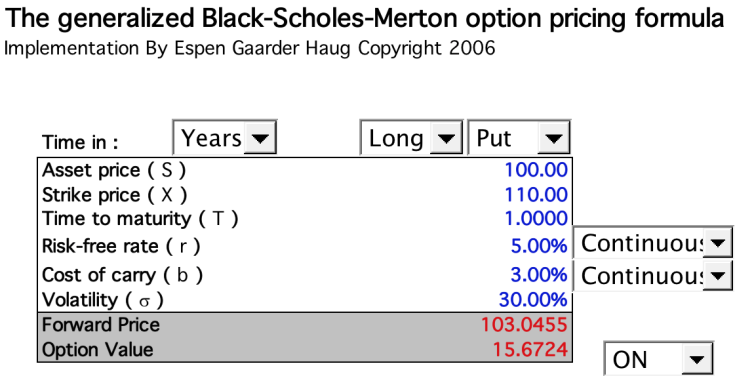
The prices of call and put calculated by our program are:

$$c = 9.05705, \quad p = 15.6724$$

Benchmark

This result is identical to the benchmark that we use, as is shown in the following screen-shots:





Numerical Methods

1. Single-step Exact SDE Solution

In this question, the SDE for stock price is:

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

This equation has an exact solution

$$S(T) = S(0)exp((r - \sigma^2/2)T + \sigma W(T))$$

where $W(t) = \sqrt{T}Z$ and Z is a standard normal random variable.

Then the value of a call option is given as:

$$V = e^{-rT}E_{\mathcal{Q}}[(S(T) - K)^+]$$

Similarly, the put option price can be obtained only by changing the payoff function.

Here, we choose $I = 10,000$ paths to do Monte Carlo simulation to get 10000 different value V^j where $j = 1, 2, ..., 10000$. The mean of all the values could be an estimator of the approximated price:

$$\hat{V} := \frac{1}{n}\sum_{j=1}^{n} V^j$$

The results generated by our program are:

$$c_{(b)} = 9.16408, \quad p_{(b)} = 15.4631$$

2. Euler Solution of SDE for Spot

For this question, the SDE of stock price is the same as section 1:

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

However, here we take 252 time steps per year to simulate the process instead of solving it by treating it as a single step process.

Euler method uses the relation:

$$\hat{X}(t_{i+1}) = \hat{X}(t_i) + a(t_i, \hat{X}(t_i))(t_{i+1} - t_i) + b(t_i, \hat{X}(t_i))(W(t_{i+1}) - W(t_i))$$

with initial condition $\hat{X}(0) = X(0)$, where the Brownian motion increments may be generated using:

$$W(t_{i+1}) - W(t_i) = \sqrt{t_{i+1} - t_i}Z_{i+1}$$

The procedure of Monte Carlo simulation for Euler method could be shown in the following function written in pseudo-code:

```
1     for (j = 0; j < numOfPaths; j++)
2     {
3         St = S0;
4         for (i = 0; i < timeSteps; i++)
5             {UPDATE St;}
6         thisPayoff = St - strike;
7         thisPayoff = thisPayoff > 0 ? thisPayoff : 0;
8         result += thisPayoff;
9     }
10    mean = (result / numOfPaths) * exp(-rfRate*maturity);
11    return mean;
```

Using Euler method that is described above, we can update S_t at every step. The approximated prices are:

$$c_{(c)} = 9.17204, \quad p_{(c)} = 15.6766$$

3. Euler Solution of SDE for Log Spot

Assume now the SDE for stock price process is turned into:

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

Now, we can treat $\ln(S(t))$ as a whole and use the same iterative formula given in section 2 with some constant term added. Thus, the only place that needs to be changed in Monte Carlo algorithm is how we update S_t. The final results are:

$$c_{(d)} = 9.16355, \quad p_{(d)} = 15.6766$$

4. Milstein Solution of SDE for Spot

Another first-order finite difference method is Milstein method. The problem here is still modeled by the SDE given in section 2. The relation used in this method is:

$$\hat{X}(t_{i+1}) = \hat{X}(t_i) + a(t_i, \hat{X}(t_i))(t_{i+1} - t_i) + b(t_i, \hat{X}(t_i))(W(t_{i+1}) - W(t_i)) \\ + \frac{1}{2} b_x(t_i, \hat{X}(t_i)) b(t_i, \hat{X}(t_i))((W(t_{i+1}) - W(t_i))^2 - (t_{i+1} - t_i))$$

where, $b_x = \frac{db}{dx} = \sigma$, other terms remain the same.

By applying this relation formula to updating the S_t, with Monte Carlo simulation, we get the results:

$$c_{(e)} = 9.22993, \quad p_{(e)} = 15.6729$$

Question

- When the payoff is $(S(T) - K)^+$, we do not need to simulate the whole paths and what we only need is to simulate is the final price. The reason is that it is a path-independent derivative so we can calculate the option price using the martingale property under risk-neutral measure.
- When it comes to a continuously monitored European-style Asian call option, it is necessary to simulate the whole paths, since now this option is path-dependent everywhere and information at any time t could influence the option price.
- However, if the Asian option is discretely monitored European-style, we only need to simulate the price at those discrete points, since the option price is determined by payoff at those points.

Error Analysis

Error Formula

According to the note, error of a method can be calculated with the formula:

$$Error = \sqrt{Variance/nPaths}$$

We approximate option prices with different seeds to have different paths for each method. Then apply the above formula. The screen-shot of the errors is shown in the following picture:

```
-------------------------------------------------------
****************** Error Analysis **********************
Single-step method Error = 0.0301072
Euler for spot Error = 0.0300736
Euler for log spot Error = 0.0301086
Milstein method Error = 0.0301559
```

In this analysis, we run each method for 100 times and then calculate the variance of the prices. We can see that all the methods have pretty small errors.

Convergence Analysis

In this part of analysis, we change the time steps into different values and then plot the approximated prices and theoretical prices into one figure to check whether the method converges. The data is shown in the following table. (we use call as an example)

Time Steps	Euler for spot	Euler for log spot	Milstein
252	9.30997	8.93313	9.17395
500	8.93131	9.03406	9.13222
1000	9.08113	9.15	9.09408

As we can see from the table, all the three methods are converging to the theoretical value: 9.05705, however with some oscillations.