IEOR 221 Homework 8

Jannin Loïc

November 2, 2023

Pricing European Call Option

We use Monte Carlo simulation to price a European call option with the following parameters: $S_0 = 100.0$, $K = 100.0$, $T = 1.0$ year , $r = 6\%$, $q = 6\%$, $\sigma = 35\%$

Plain MC

This method consists of simulating a large number of potential outcomes for the stock price at time T. Subsequently, we compute the payoff of the call option for each final price and then take the average to obtain the derivative's average payoff. The price of the derivative is the present value of this average payoff, discounted with the risk-free rate, assuming a risk-free world for the pricing.

For this simulation, we assume that the price follows the following geometric Brownian motion:

$$dS = (r - q)S\,dt + \sigma S\,dz$$

Where dz is a Wiener process. Following this model, we can assume that:

$$S(t + \Delta t) - S(t) = (r - q)S(t)\,\Delta t + \sigma S(t)\sqrt{\Delta t}\,\epsilon$$

Written differently, ϵ follows a standard normal distribution and:

$$S(t + \Delta t) = S(t)exp[(r - q - \frac{\sigma^2}{2})\Delta t + \sigma\sqrt{\Delta t}\,\epsilon]$$

For a call option, the derivative price is then given by:

$$f = e^{-rT}\max(S - K, 0)$$

In our case, $S = \hat{S} = \mathbf{E}_t(S_T) = \frac{1}{n}\sum_{k=1}^{n} S^k_{\text{simulated}}(T)$

As advised by Pr.Zhang, we chose $n_{steps} = 100$, $n_{simulation} = 4000$ and we have the following result:

Price of the call option: $13.50

Variance of the call option: 611.22

The code used for this simulation is the following:

```
import numpy as np

def option_payoff(S, K, option_type):
    payoff = 0.0
    if option_type == "call":
        payoff = max(S - K, 0)
    elif option_type == "put":
        payoff = max(K - S, 0)
    return payoff

S0 = 100
K = 100
T = 1.0
r = 0.06
q = 0.06
sigma = 0.35
option_type = "call"
n_steps = 100 # number of steps
np.random.seed(102623) # make result reproducable
n_simulation = 4000 # number of simulation
dt = T/n_steps
sqrt_dt = np.sqrt(dt)
payoff = np.zeros((n_simulation), dtype = float)
step = range(0, int(n_steps), 1)

for i in range(0, n_simulation):
    ST = S0
    for j in step:
        epsilon = np.random.normal()
        ST *= np.exp((r - q - 0.5*sigma*sigma)*dt +
        sigma*epsilon*sqrt_dt)

    payoff[i] = option_payoff(ST, K, option_type)

option_variance = np.var(payoff*np.exp(-r*T))
option_price = np.mean(payoff)*np.exp(-r*T)
print(option_type + ' price =', round(option_price, 2))
print(option_type + ' variance =', round(option_variance,
    2))
```

Listing 1: Plain MC Code

Antithetic Method

The Antithetic Method involves generating pairs of potential outcomes for the stock price at time T, aimed at reducing variance in option pricing estimation. In this method, pairs of outcomes are simulated, employing the same pricing model assumptions.

This technique of generating negatively correlated pairs is designed to mitigate variance in the estimated option price, providing more accurate and stable results compared to using solely independent outcomes.

We have the following results:

Price of the call option: $12.97

Variance of the call option: 203.57

The code used for this simulation is the following:

```
import numpy as np

def option_payoff(S, K, option_type):
    payoff = 0.0
    if option_type == "call":
        payoff = max(S - K, 0)
    elif option_type == "put":
        payoff = max(K - S, 0)
    return payoff

S0 = 100
K = 100
T = 1.0
r = 0.06
q = 0.06
sigma = 0.35
option_type = "call"
n_steps = 100 # number of steps
np.random.seed(102623) # make result reproducable
n_simulation = 4000 # number of simulation
dt = T/n_steps
sqrt_dt = np.sqrt(dt)

payoff = np.zeros((n_simulation), dtype = float)
step = range(0, int(n_steps), 1)

for i in range(0, n_simulation):
    ST1 = S0
    ST2 = S0
    for j in step:
        epsilon = np.random.normal()
        ST1 *= np.exp((r - q - 0.5*sigma*sigma)*dt +
        sigma*epsilon*sqrt_dt)
```

```
        ST2 *= np.exp((r - q - 0.5*sigma*sigma)*dt -
        sigma*epsilon*sqrt_dt)

    payoff[i] = (option_payoff(ST1, K,
        option_type)+option_payoff(ST2, K, option_type))/2

option_variance = np.var(payoff*np.exp(-r*T))
option_price = np.mean(payoff)*np.exp(-r*T)

print(option_type + ' price =', round(option_price, 8))
print(option_type + ' variance =', round(option_variance,
    8))
```

Listing 2: Antithetic Method Code

Control Variate Method

This method involves introducing a known random variable, called the control variate, to reduce the variance in the estimation of the target variable—in this case, the option price.

In the context of option pricing, the stock price S at time T is used as the control variate. The expected value of the stock price at time T $E(S_T)$ under risk-neutral valuation is given by:

$$\mathbf{E}(S_T) = S_0 e^{(r-q)T}$$

Thus, a better estimate f_{call} of the option price is given by:

$$f_{call} = f^* - S_T^* + \mathbf{E}(S_T)$$

Where f^* and S_T^* are respectively the estimated payoff and the estimated stock price at t = T

Price of the call option: $13.27

Variance of the call option: 253.68

```python
import numpy as np

def option_payoff(S, K, option_type):
    payoff = 0.0
    if option_type == "call":
        payoff = max(S - K, 0)
    elif option_type == "put":
        payoff = max(K - S, 0)
    return payoff

S0 = 100
K = 100
T = 1.0
r = 0.06
q = 0.06
sigma = 0.35
option_type = "call"
n_steps = 100 # number of steps
np.random.seed(102623) # make result reproducable
n_simulation = 4000 # number of simulation
dt = T/n_steps
sqrt_dt = np.sqrt(dt)

payoff = np.zeros((n_simulation), dtype = float)
step = range(0, int(n_steps), 1)
risk_neutral_ST = S0*np.exp((r-q)*T)

for i in range(0, n_simulation):
    ST = S0
```

```
30
31      for j in step:
32          epsilon = np.random.normal()
33          ST *= np.exp((r - q - 0.5*sigma*sigma)*dt +
34          sigma*epsilon*sqrt_dt)
35
36      payoff[i] = option_payoff(ST, K, option_type) - ST +
            risk_neutral_ST
37
38
39  option_variance = np.var(payoff*np.exp(-r*T))
40  option_price = np.mean(payoff)*np.exp(-r*T)
41  print(option_type + ' price =', round(option_price, 8))
42  print(option_type + ' variance =', round(option_variance,
        8))
```

Listing 3: Control Variate Method Code

Results

In this last section we compare the results of the different methods. Let's start by computing the call price given by the BSM equations :

$$c = S_0 \cdot e^{-q \cdot T} \cdot N(d_1) - K \cdot e^{-r \cdot T} \cdot N(d_2)$$

And we have a price of : $**13.08**

Method	**Price**	**Error to BSM**
Plain MC	13.50	0.42
Antithetic	12.97	0.11
Control Variate	13.27	0.19

Table 1: Comparison of Option Prices and Errors

The first method presents the worst approximation of the BSM model, with a variance three times higher than the other two methods. The two other techniques yield similar results, with the antithetic method slightly more accurate.