MATH 6767 Homework 5

Yuliang Li GTID# 903012703

Project Object and Option Analysis

In this project, we price an option, which is on the combination of S&P 500 and Russell 2000 indexes. We value the payoff of the option at the $\frac{R(T)+S(T)}{2}$, where $R(T)$ is the index of Russell 2000 at time T and $S(T)$ is the index of S&P 500 at time T.

Risk free rate (r): 1%
Time to expiration: 1 year (365 days)
Dividend yield of S&P 500: 1.93%
Dividend yield if Russell 2000: 1.56%
Initial spot price of S&P 500 (S_0): 1800
Initial spot price of Russell 2000 (R_0): 1120

Thus, the initial spot price of the combined index is $S_0 = \frac{(1800+1120)}{2} = 1460$.

Concluded from the payoff figure in BOA's instruction, we can replicate the option by the portfolio like:

Short $10/S_0$ vanilla European put options, whose strike price (index) is $K_p = 0.95S_0 = 1387$; long 2 digital call options, whose strike price is $K_d = S_0 = 1460$; long $10/S_0$ vanilla European call options, whose strike price is $K_c = 1.2S_0 = 1752$; A risk-free assets whose value is 10 at T.

The movements of underlying price and volatility are modeled by Heston Model.

The index's process is like

$$S_{i+1} = S_i + (r-d)S_i\Delta t + \sqrt{v_i\Delta t}S_i Z'$$

The volatility's process is modified by Milstein Method, which is

$$v_{i+1} = (\sqrt{v_i} + \frac{\eta}{2}\sqrt{\Delta t}Z)^2 - \lambda(v_i-\theta)\Delta t - \frac{\eta^2}{4}\Delta t$$

where Z, Z' is a random variables satisfy standard random distribution.

In this project, we price the option under two situations: all the stochastic processes in situation I are independent, and in situation II they are correlated. The correlation matrix is

$$\begin{bmatrix} 1 & 0.7 & -0.7 & -0.8 \\ 0.7 & 1 & -0.6 & -0.7 \\ -0.7 & -0.6 & 1 & 0.8 \\ -0.8 & -0.7 & 0.8 & 1 \end{bmatrix}$$

In a correlated process, we should get new standard Brownian motion terms,

which are different from those in correlated process. In two Geometric

Brownian Motion, the form of two stochastic processes are

$$d\begin{bmatrix} X_t \\ Y_t \end{bmatrix} = \begin{bmatrix} \mu_1 X_t \\ \mu_2 Y_t \end{bmatrix} dt + \begin{bmatrix} \sigma_1 X_t & 0 \\ \rho\sigma_2 Y_t & \sigma_2\sqrt{1-\rho^2}\, Y_t \end{bmatrix} \begin{bmatrix} dW_t^1 \\ dW_t^2 \end{bmatrix}$$

It's similar for Heston process. Thus, we need to decompose the corrlation

matrix to get a lower triangle martix for multiplying with the independent

standard brownian motion terms. In our project, we use Cholesky

decomposition to get it.

Project Structure

Main.cpp: The main file. Complete pricing the option under independent situation and correlated situation. The generation of correlation matrix and Cholesky decomposition is also completed to get the correlated process.

Simulator.h: Simulate the stochastic processes with generating random numbers. The independent and correlated Heston processes are simulated in this file. The Milstein method is also simulated.

MCEngine.h: Monte Carlo Engines, which price options with various payoffs by Monte Carlo Methods simulated by different stochastic processes.

OptionPay.h and **OptionPay.cpp**: calculate the payoff of vanilla option and down and in barrier option.

IO.h and **IO.cpp**: input file contains parameters for pricing and output the results to csv files.

Project Implementation and Programming Analysis

Sum of Payoff

The VanillaPay and DigitalPay are derived class of a base class Payoff. If we put the objects of them in the same function simultaneously, they will cause ambiguity. To solve the problem, the function clone() in the two classes will return a pointer to the object itself. The function clone() also is a virtual function in base class Payoff. The code is like,

```
Payoff* DigitalPay::clone()
{
        return new DigitalPay (*this);
}
```

Then we put all option's payoffs and their amounts in the class PayofSum, which use two vectors to store them. To store the payoffs in the vector, we pass the pointer, which is to object itself, to class PayforSum. The code is like,

```
// Payoff of each option
class PayforSum
{
        Payoff* OptPay;
        public:
        PayforSum(Payoff& optpay)
        {
                OptPay = optpay.clone();
        }
         double operator() (double spot)
         {
                 return (*OptPay)(spot);
         }
};

// sum of all payoff
class PayofSum
{
         vector<PayforSum> pos;
         vector<double> amt;// the amout of option
         public:
         void insert(Payoff& optpay, double amount)
         {
                 PayforSum pfs(optpay);
                 pos.push_back(pfs);
                 amt.push_back(amount);
         }
         double operator() (int i, double spot)
         {
                 return pos[i](spot) * amt[i];
         }
```

```
        double size()//the size of PayofSum
        {
                return amt.size();
        }
};
```

So we only need to pass the object of PayofSum to the Monte Carlo Engine function. Using this method, it is easy to add new options and put the new one in the Monte Carlo Engine.

Monte Carlo Engine

For all three options, there only need one Monte Carlo simulation for the class PayofSum. In this project, the function for Monte Carlo Engine is still a template function. The code for independent stochastic processes is like,

```
template <typename T, typename Simulator> // T is sum of payoff, Simulator is the simulator
double MonteCarloPricer_MS(int N, T p, std::vector<Simulator> s, double discount)
{
        if(N <= 0){ std::cerr << "\n\t illegal number of simulations "<< std::endl;exit(1);}
        double sum = 0.0;
        int s_size = s.size();
        double weight = 1.0 / double(s_size);
        //std::cout << weight << std::endl;
        for(int i = 0; i< N; ++i)
        {
                double z = 0;
                for(int j = 0; j < s_size; j++)
                {
                        z += weight * s[j]();
                }
                for(int j = 0; j < p.size(); j++)

                        sum += p(j, z);
                }
        }
        return discount*(sum/N);
}
```

We use function clock() in library ctime to evaluate the Monte Carlo Engine. The result is in Table 1. The number of simulation is 10000.

Table 1 The duration time of Monte Carlo Engine

Monte Carlo Engine for	Independent process	Correlated process

Duration Time (seconds)	0.9162	2.2793

The engine for correlated process cost more time because at each step, we have to multiply the correlation matrix (decomposition to lower triangle matrix) to the independent standard Brownian Motion.

Results and Financial Analysis

The price of the option is shown in Table 2.

Table 2 The price of BOA option

	Independent process	Correlated process
Option price (dollars)	10.4872	10.4928

The prices of each replicating part are in Table 3.

Table 3 The price of each replicationg part

	Short put option	Long call digital option	Long call option
Independent Process	-0.3540	0.0635	0.8771
Correlated Process	-0.4776	0.0584	1.0114

Considering some errors in our simulation and correlation matrix, BOA sell its security at 10 dollars each seems reasonable. However, there exists some risks that, if the errors are small enough, BOA price the security lower than reasonable price, that means BOA may suffer loses in selling it.