

Report for OOPA Big Project

Zihao Song CMF
10/01/2019

This report, which contains the descriptions of work, figures, tables, plots I have done and C# code as well, is for 2018-2019 Semester one OOPA course.

Task 2.1 (Code set-up).

After finishing the task work, I summarize the structure of my C# code, which has 9 new classes, the GeneratePath.cs for generating a path of Monte Carlo;
HestonAsianoption.cs for calculating Heston Asian call and put price;
Hestonformula.cs for setting up Heston formula;
Hestoncalibrator.cs for calculating the error and providing the parameters after calibrating;
HestonLookbackoption.cs for calculating the Heston Look back option price;
HestonMC.cs for calculating Heston European option price by MC;
SolversAndIntegrators.cs for using it as a method to do the integration;
HestonRainbowoptionprice.cs for calculating the Heston Rainbow option price;
Hestonmodelparameter.cs for testing the interface;
(for all variables in classes and all methods are in XML document which is described at the last) in the project HestonModel and new projects solution for getting the correct answers. With revising the Heston.cs, I implement codes for the Interface and try to write my own test. Besides that, I provide the XML documentation and research a lot about topics related to this project and achieve them in my code, like pricing Rainbow options with payoff: Best of assets or cash: $\max(S_1, S_2, \dots, S_n, K)$ and variance reduction method in the Monte-Carlo algorithm by reading essay <Monte Carlo methods -A review of Variance Reduction Techniques > [1].

Task 2.2 (Heston formula).

By using the method provided in essay, firstly setting a barrier avoiding code duplication and defining the functions d , g , C , D , Φ and P , then using the result of each function step by step, we get the price when $t = 0$, under these conditions:

$r = 0.025$, $\theta = 0.0398$, $\kappa = 1.5768$, $\sigma = 0.5751$, $\rho = -0.5711$, $Nv = 0.0175$, $S = 100$, $K = 100$, Option exercise $T = 1, 2, 3, 4, 15$.

Below is the result of call and put option price:

```
C:\Users\Lenovo\Documents\GitHub\final-project-ytycmx\Code\ConsoleApp2\bin\Debug\ConsoleApp2.exe
These are from Heston formula call option:
7. 27417978294247
11. 7371414763384
15. 4789594840967
18. 7738224489144
43. 1700985199846
These are from Heston formula put option:
4. 80517098577573
6. 86008392640984
8. 25330811695197
9. 25756425251038
11. 8990263990818
```

Figure 1. Result of Heston formula

For the things we need to notice are:

1. We need firstly define the i which is for complex operation.
2. What needs to be considered is which method we need to use for integration, and how large the constant we choose for the integrating range.
3. Notation!!! When we define double a has fraction, it definitely should use double in both numerator and denominator, i.e. $a = 1/2$ is 0 in C#.
4. It is better to initialize separately the parameters in model and in the option, so we could use it more efficiently, i.e. initializing the model parameters outside the function calculating price and initializing the option parameters when using calculating function.
5. Extension packages in C# are quietly useful and high-efficient!

Task 2.3 (Heston Call/Put with Monte Carlo).

By using the method provided in essay, setting several barriers avoiding code duplication at the beginning, we need to generate the path, containing s (price) and y (volatility), in three steps using delta Z, getting at least 365 items, firstly. After achieving generating one path, we could use the payoff formula 100000 times and get the average. That is the price for MC European.

This is the price when $t = 0$, under these conditions:

$r = 0.1$, $\theta = 0.06$, $\kappa = 2$, $\sigma = 0.4$, $\rho = 0.5$, $Nv = 0.04$, $S = 100$, $K = 100$, option exercise $T = 1, 2, 3, 4, 15$.

Below is the result of MC European call and put option price:

```
C:\Users\Lenovo\Documents\GitHub\final-project-ytycmx\Code\ConsoleApp2\bin\Debug\ConsoleApp2.exe
These are from Heston European call option MC:
13. 683066431941
22. 3643249044
29. 9883289853792
36. 9158013644337
77. 4588278845067
These are from Heston European put option MC:
4. 25230153123793
4. 16770831081618
4. 26802281843985
3. 91522454960175
0. 113593398977297
```

Figure 2. Result of Heston MC

For the things we need to notice are:

1. Your algorithm only needs to run if the Feller condition $2 * k * \theta > \sigma * \sigma$ holds.
2. It is better to return all items when generating a path.
3. Parallel implementation of the Monte-Carlo algorithm is much quicker than circulation in circulation.

Task 2.4 (Checking Heston formula and Monte Carlo).

Experimentally (i.e. run the code) confirm that your option prices match between the Heston formula and Monte Carlo code. So under the condition of (2.3), we get:

```
C:\Users\Lenovo\Documents\GitHub\final-project-ytycmx\Code\ConsoleApp2\bin\Debug\ConsoleApp2.exe
These are from Heston formula call option:
13. 6297122472144
22. 4525046964915
29. 9949494337074
36. 6542682935841
78. 468847523295
These are from Heston formula put option:
4. 11345405081038
4. 32558000428972
4. 07677150187919
3. 68627289714804
0. 781863538137994
These are from Heston European call option MC:
13. 5379672941981
22. 6072198943861
29. 9167617484619
36. 7086978047661
78. 4123443899246
These are from Heston European put option MC:
4. 09803719451199
4. 34401869478793
4. 08049716987438
3. 78939481838226
0. 2473294303073
```

Figure 3. Difference of formula and MC

, which seems that the MC is quite similar with formula, and it works.

For the plot: data from 2.3

We plot the convergence in log-log scale as below:

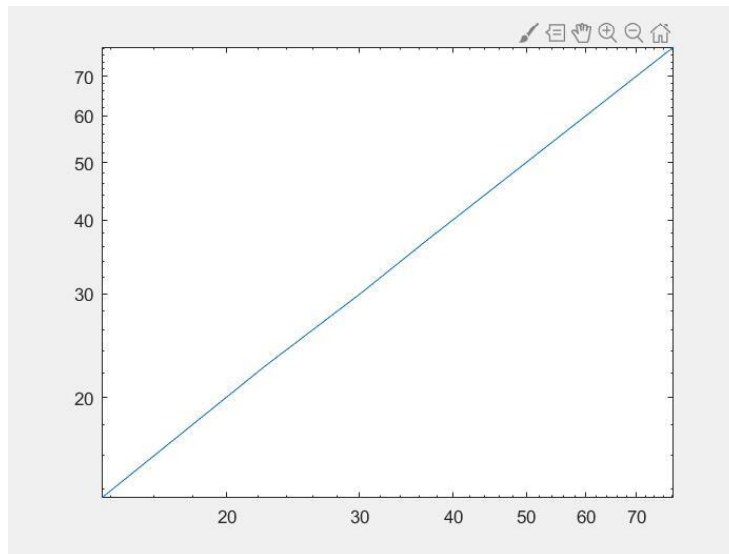


Figure 4.1 MC and formula call price convergence plot

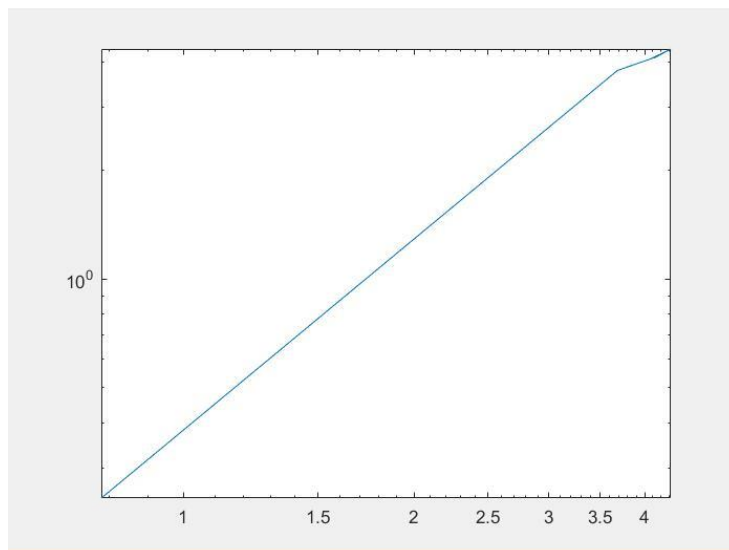


Figure 4.2 MC and formula put price convergence plot

Explain as below:

About why my experiments make sense, the plots would directly show that almost all prices calculated from formula could match those from MC quite well, and also from the theory, MC problem solved can be transformed into a randomly distributed number of features, such as the probability of occurrence of a random event, or the expected value of a random variable. The probability of random events is estimated by random sampling, or the numerical characteristics of random variables are estimated by the numerical features of the samples, and they are used as solutions to the problem. So, the experiment could be done well.

Task 2.5 (Calibration).

By using the codes professor provided, we need to check and modify certain lines, in order to make it useful for us. Besides that, we need to add several lines in Hestonformula.cs to get the

parameters from the Heston model.

Minimizing mean square error between model prices and market prices is the main thing in calibration, by using the Heston formula code from Task 2.2 and treating the initial asset price S and risk-free-rate r as given but the initial variance Nv as a model parameter. Thus, your minimizer should minimize over κ , θ , σ , ρ and Nv .

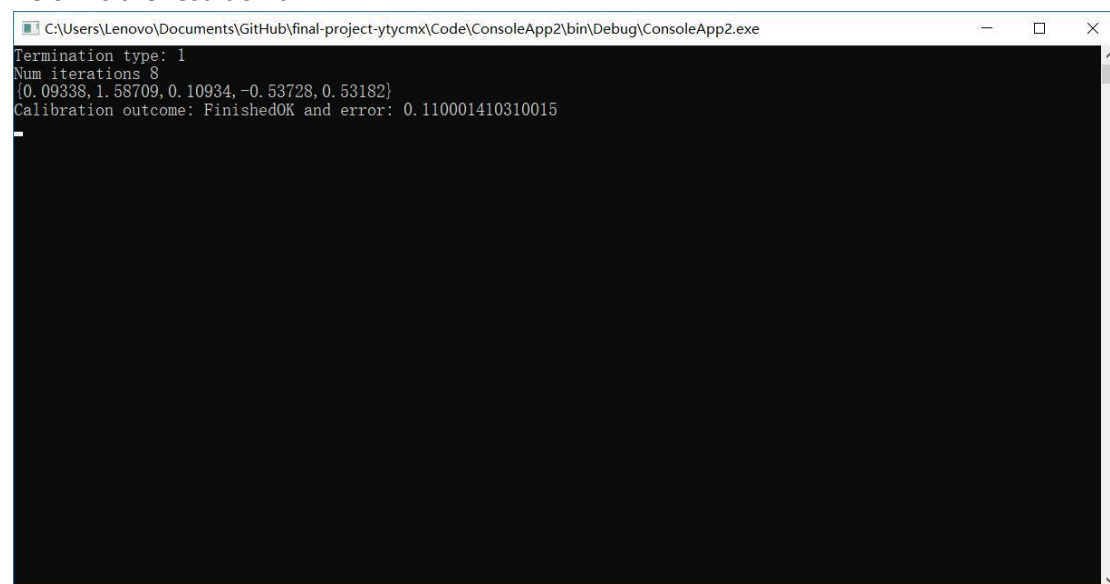
The error is provided when the accuracy is set to 0.001, maximum iterations to 1000, $S = 100$, $r = 2.5\%$, initializing $\theta = 0.0398$, $\kappa = 1.5768$, $\sigma = 0.5751$, $\rho = -0.5711$, $Nv = 0.0175$ and with market data from project requirements:

$K = 80, 90, 80, 100, 100$;

Option exercise $T = 1, 1, 2, 2, 1.5$;

Market call price = 25.72, 18.93, 30.49, 19.36, 16.58.

Below is the result of it:

A screenshot of a Windows console window titled "C:\Users\Lenovo\Documents\GitHub\final-project-ytycmx\Code\ConsoleApp2\bin\Debug\ConsoleApp2.exe". The console output shows the following text: "Termination type: 1", "Num iterations 8", "{0.09338, 1.58709, 0.10934, -0.53728, 0.53182}", and "Calibration outcome: FinishedOK and error: 0.110001410310015". The rest of the console window is black.

```
Termination type: 1
Num iterations 8
{0.09338, 1.58709, 0.10934, -0.53728, 0.53182}
Calibration outcome: FinishedOK and error: 0.110001410310015
```

Figure 5. Result of calibration

For the things we need to notice are:

1. This will take a lot more time than calibrating the Vasicek model from the workshop. This is because you need numerical integration every time you price an option using the Heston formula. Each step in the minimization algorithm will price the option at least six times.
2. There shows that it is better to separately initialize the parameters in model and option price, because we only need to calculate the parameters in model here.
3. I need to think about the meaning of calibration in the future!
4. Setting calibrated Params = new double[] { 0.0175, 1.5768, 0.0398, -0.5711, 0.5751 } is using for if there is no parameters transmitting inside.

Task 2.6 (Checking calibration).

As above, the method works, in three steps:

1. Initialize 5 parameters in the model and pass in the value;
2. Calculate the mean square error;
3. Using the system function `alglib.minlbfgsoptimize` and the result we get fifth times, it is easy

to find the five calibrated parameters and return them.

The result of checking is above and here we only discuss about why the calibration procedure fails:

1. When I test in my C# calibration class, if the parameters theta, kappa, sigma, rho, Nv cannot match the order, the system would get a ridiculous answer;
2. At beginning I did not separately initialize the parameters so that theta, kappa, sigma, rho, Nv could not get inside of my model, and it fails;
3. Over the Maximum steps was taken inside the algorithm;
4. Does not meet the Feller condition;
5. Some misquotation or other mistakes in calculation;
6. The minimizer laid on the local minimum.

Task 2.7 (Pricing Asian arithmetic option).

By using how to generating path in 2.3, we get over 100000 paths and using the payoff formula, we calculate the call and put price for Asian option.

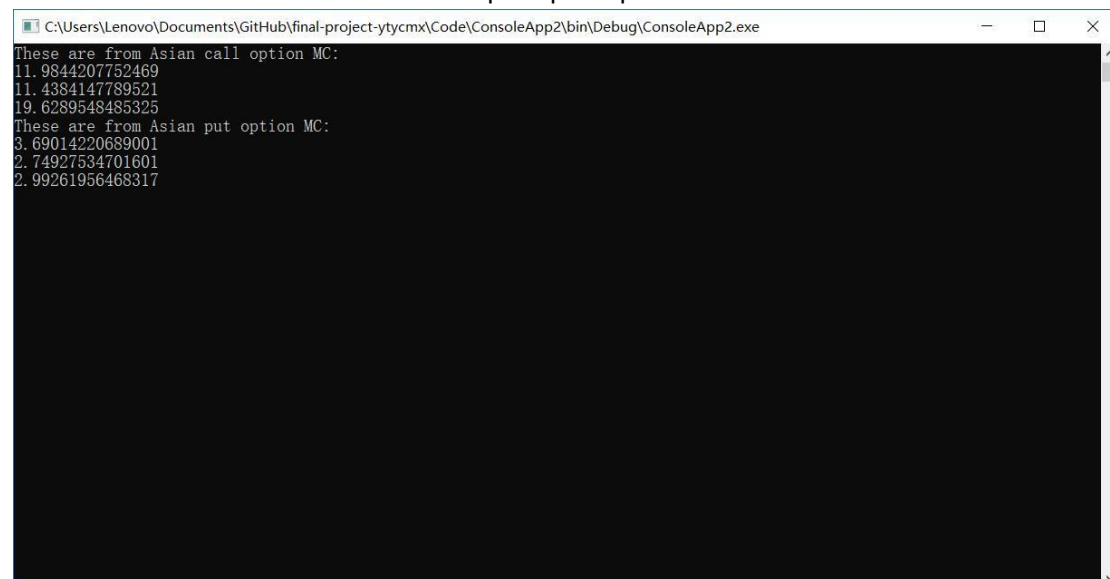
Before we getting start with calculation, we need to set up the barriers like:

1. Need at least one monitoring date for Asian option.
2. The first monitoring date must be positive.
3. Monitoring dates must be increasing.
4. Last monitoring dates must not be greater than the exercise time.

This is the price under these conditions:

$r = 0.1$, $\theta = 0.06$, $\kappa = 2$, $\sigma = 0.4$, $\rho = 0.5$, $Nv = 0.04$, $S = 100$, $K = 100$, option exercise $T = 1, 2, 3, T_1, \dots, T_m = \{0.75, 1.00\}, \{0.25, 0.50, 0.75, 1.00, 1.25, 1.50, 1.75\}, \{1.00, 2.00, 3.00\}$.

Below is the result of MC Asian call and put option price:



```
C:\Users\Lenovo\Documents\GitHub\final-project-ytycmx\Code\ConsoleApp2\bin\Debug\ConsoleApp2.exe
These are from Asian call option MC:
11. 9844207752469
11. 4384147789521
19. 6289548485325
These are from Asian put option MC:
3. 69014220689001
2. 74927534701601
2. 99261956468317
```

Figure 6. Result of Heston Asian option price

For the things we need to notice are:

1. We need to use the put option payoff formula, not the call-put parity, because it is used for European option.

2. We need to get the certain T_i item inside our generated path, which means we should convert the T_i into the number, i.e. we need to convert 0.75 in T_1 to 273 in path.
3. Parallel implementation of the Monte-Carlo algorithm is much quicker than circulation in circulation.

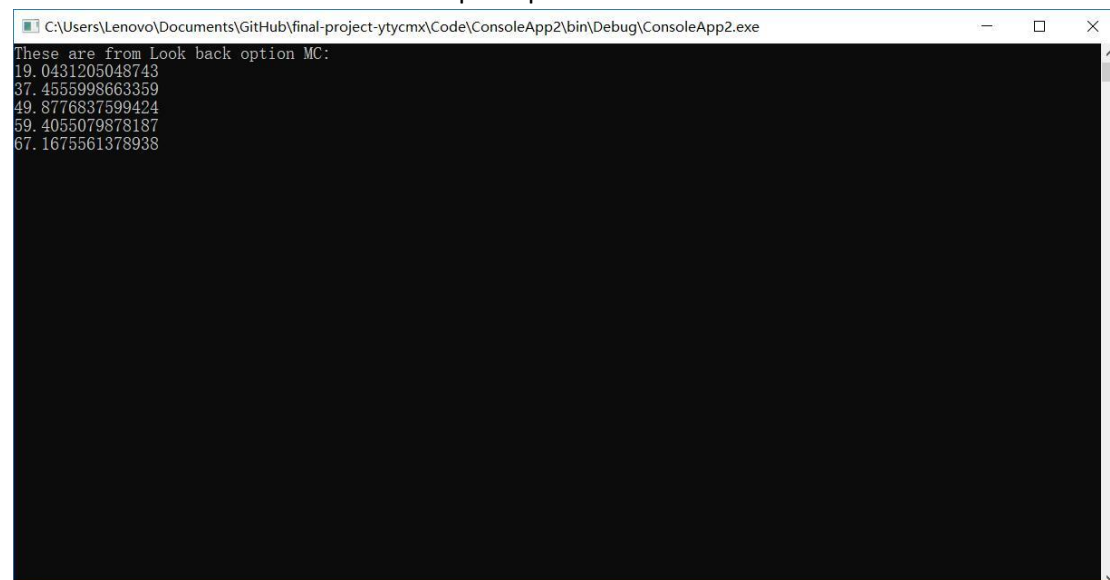
Task 2.8 (Pricing lookback option).

Similar as 2.3 and 2.7, we generate over 100000 paths and calculate the price by payoff formula. The only barrier is the Feller condition.

This is the price under these conditions:

$r = 0.1$, $\theta = 0.06$, $\kappa = 2$, $\sigma = 0.4$, $\rho = 0.5$, $N_v = 0.04$, $S = 100$, option exercise $T = 1, 3, 5, 7, 9$.

Below is the result of MC Look back option price:



```

C:\Users\Lenovo\Documents\GitHub\final-project-ytycmx\Code\ConsoleApp2\bin\Debug\ConsoleApp2.exe
These are from Look back option MC:
19. 0431205048743
37. 4555998663359
49. 8776837599424
59. 4055079878187
67. 1675561378938
  
```

Figure 7. Result of Heston Look back option price

For the things we need to notice are:

1. There is no call or put, only one option price.
2. Parallel implementation of the Monte-Carlo algorithm is much quicker than circulation in circulation.

Exploration.

1. Rainbow option:

Rainbow option is a derivative exposed to two or more sources of uncertainty, as opposed to a simple option that is exposed to one source of uncertainty, such as the price of underlying asset. Rainbow options are usually calls or puts on the best or worst of an underlying assets. Like basket option, which is written on a group of assets and pays out on a weighted-average

gain on the basket as a whole, a rainbow option also considers a group of assets, but usually pays out on the level of one of them. Alternatively, in a more complex scenario, the assets are sorted by their performance at maturity, for instance, a rainbow call with weights 50%, 30%, 20%, with a basket including FTSE 100, Nikkei and S&P 500 pays 50% of the best return (at maturity) between the three indices, 30% of the second best and 20% of the third best.

The options are often considered a correlation trade since the value of the option is sensitive to the correlation between the various basket components.

Rainbow options are used, for example, to value natural resources deposits. Such assets are exposed to two uncertainties—price and quantity.

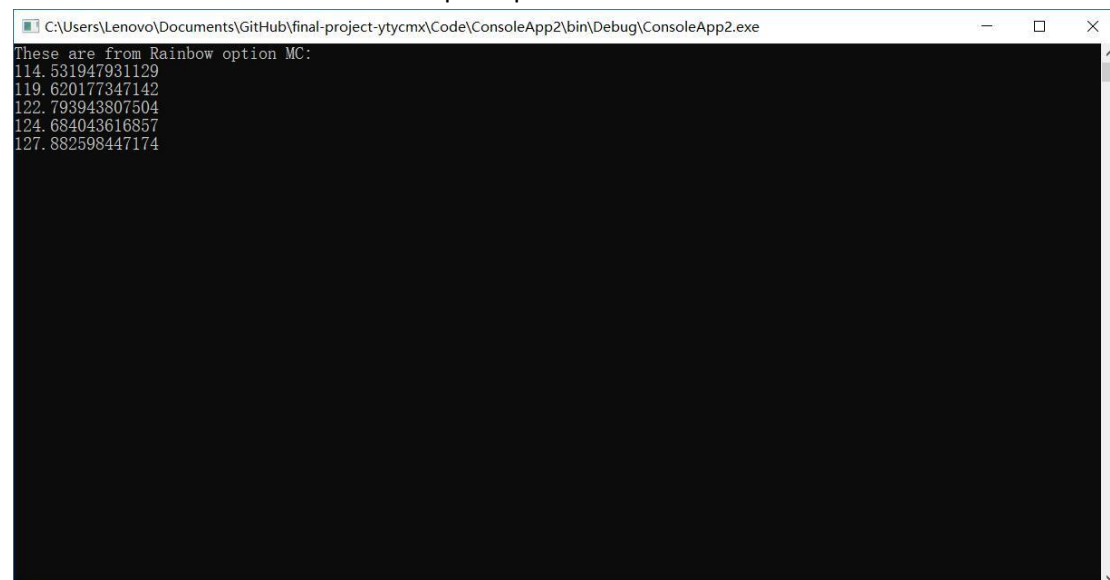
I use the Rainbow option payoff as below:

Best of assets or cash option, delivering the maximum of two risky assets and cash at expiry, i.e. $\max(S_1, S_2, \dots, S_n, K)$

This is the price under these conditions:

$r = 0.1$, $\theta = 0.06$, $\kappa = 2$, $\sigma = 0.4$, $\rho = 0.5$, $N_v = 0.04$, $S = 100$, $K = 100$, Option exercise $T = 1, 2, 3, 4, 15$, $N = 365$, $n = 100000$.

Below is the result of MC Rainbow option price:



```

C:\Users\Lenovo\Documents\GitHub\final-project-ytycmx\Code\ConsoleApp2\bin\Debug\ConsoleApp2.exe
These are from Rainbow option MC:
114.531947931129
119.620177347142
122.793943807504
124.684043616857
127.882598447174

```

Figure 8. Result of Heston Rainbow option price

2. variance reduction method in the Monte-Carlo algorithm:

When using simulation techniques to estimate a function f it requires a very high amount of trials to get a reasonable accuracy in the result. This is in most cases very expensive in terms of computational time. This is where the variance reduction techniques come to use. By reducing the variance of the simulation results the computational time could be greatly reduced because the number of simulations needed for an accurate result is a lot lower [1].

There are several different variance reduction techniques in financial theory, some that is easily explained by theory and mathematics and extremely hard to implement and some which are easy to implement and fairly easy to describe on a theory level [1].

We only talk about the Antithetic Variate and it is categorized into the strategy where the

output is adjusted and corrected. It is a variance reduction technique that uses the negative dependence between pairs of replications to reduce the variance, and the most used one is based on the observation U that is uniformly distributed over $[0, 1]$. If the variable U is then used for simulating a path, the complement $1-U$ or $-U$ can be used to simulate a second path without changing the law of simulated process.

These two are together called antithetic pair in the sense that the antithetic path would balance an unusually large or small value calculated from the first simulated path and therefore reduce the variance [1].

In the case of estimating the expected value I of the function $f(U)$:

$$I = E(f(U)) \text{ where } U \sim N(0, 1);$$

A crude Monte Carlo estimate would look like this:

$$I_N = \frac{1}{N} \sum_{i=1}^N f(U_i) \text{ where } U_i \sim N(0, 1);$$

And the corresponding antithetic variant would be:

$$I_N = \frac{1}{N} \sum_{i=1}^N \frac{f(U_i) + f(-U_i)}{2} \text{ where } U_i \sim N(0, 1).$$

Antithetic variate is quite easy to implement, all you actually need to do is run the same function again with the same input, but this time negative, and then calculate the mean of the results [1].

Thus, we could find this method is easy to reduce the variance in Monte-Carlo method and as [1] said it is widely used in industry and finance.

As for our algorithm, what need to change is when we calculate ΔZ_i in the second step, it should be modified as:

$$\Delta Z_{1,N}^k := \sqrt{\tau} \frac{x_{1,N}^k - x_{3,N}^k}{2},$$

$$\Delta Z_{2,N}^k := \sqrt{\tau} \left(\rho \frac{x_{1,N}^k - x_{3,N}^k}{2} + \sqrt{1 - \rho^2} \frac{x_{2,N}^k - x_{4,N}^k}{2} \right),$$

where $x_{i,N}^k \sim N(0, 1), i = 1, 2, 3, 4$.

For XML, I tried to get it by DOS, but failed as below:

```

Developer Command Prompt for VS 2017
Heston.cs(8, 7): error CS0246: 未能找到类型或命名空间名 "FinalAssignment5" (是否缺少 using 指令或程序集引用?)
Heston.cs(29, 74): error CS0246: 未能找到类型或命名空间名 "IHestonModelParameters" (是否缺少 using 指令或程序集引用?)
Heston.cs(30, 86): error CS0246: 未能找到类型或命名空间名 "IOptionMarketData<" (是否缺少 using 指令或程序集引用?)
Heston.cs(30, 104): error CS0246: 未能找到类型或命名空间名 "IEuropeanOption" (是否缺少 using 指令或程序集引用?)
Heston.cs(31, 74): error CS0246: 未能找到类型或命名空间名 "ICalibrationSettings" (是否缺少 using 指令或程序集引用?)
Heston.cs(29, 23): error CS0246: 未能找到类型或命名空间名 "IHestonCalibrationResult" (是否缺少 using 指令或程序集引用?)
Heston.cs(61, 56): error CS0246: 未能找到类型或命名空间名 "IHestonModelParameters" (是否缺少 using 指令或程序集引用?)
Heston.cs(62, 56): error CS0246: 未能找到类型或命名空间名 "IEuropeanOption" (是否缺少 using 指令或程序集引用?)
Heston.cs(92, 58): error CS0246: 未能找到类型或命名空间名 "IHestonModelParameters" (是否缺少 using 指令或程序集引用?)
Heston.cs(93, 58): error CS0246: 未能找到类型或命名空间名 "IEuropeanOption" (是否缺少 using 指令或程序集引用?)
Heston.cs(94, 58): error CS0246: 未能找到类型或命名空间名 "IMonteCarloSettings" (是否缺少 using 指令或程序集引用?)
Heston.cs(134, 55): error CS0246: 未能找到类型或命名空间名 "IHestonModelParameters" (是否缺少 using 指令或程序集引用?)
Heston.cs(135, 55): error CS0246: 未能找到类型或命名空间名 "IASianOption" (是否缺少 using 指令或程序集引用?)
Heston.cs(136, 55): error CS0246: 未能找到类型或命名空间名 "IMonteCarloSettings" (是否缺少 using 指令或程序集引用?)
Heston.cs(178, 58): error CS0246: 未能找到类型或命名空间名 "IHestonModelParameters" (是否缺少 using 指令或程序集引用?)
Heston.cs(179, 58): error CS0246: 未能找到类型或命名空间名 "IOption" (是否缺少 using 指令或程序集引用?)
Heston.cs(180, 58): error CS0246: 未能找到类型或命名空间名 "IMonteCarloSettings" (是否缺少 using 指令或程序集引用?)

C:\Users\Lenovo\Documents\GitHub\final-project-ytycmx\Code\HestonModel\csc Hestonformula.cs /doc:Heston.xml
Microsoft(R) Visual C# 编译器 版本 2.9.0.63208 (958f2354)
版权所有 (C) Microsoft Corporation. 保留所有权利。

Hestonformula.cs(6, 7): error CS0246: 未能找到类型或命名空间名 "MathNet" (是否缺少 using 指令或程序集引用?)
Hestonformula.cs(8, 7): error CS0246: 未能找到类型或命名空间名 "SolversAndIntegrators" (是否缺少 using 指令或程序集引用?)
Hestonformula.cs(19, 16): error CS0246: 未能找到类型或命名空间名 "Complex" (是否缺少 using 指令或程序集引用?)

C:\Users\Lenovo\Documents\GitHub\final-project-ytycmx\Code\HestonModel>

```

For the Chinese parts, it means it cannot find the type or named space, but actually I wrote “`using MathNet.Numerics.Integration; using System.Numerics; using SolversAndIntegrators;`” at the top of the Hestonformula.cs. Thus, I wonder what is wrong in fact.

Reference:

[1] Håkan Andersson, Monte Carlo methods -A review of Variance Reduction Techniques. Master of Science Thesis INDEK 2013:03, KTH Industrial Engineering and Management.

Appendix codes

Task 2.2 (Heston formula).

Hestonformula.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MathNet.Numerics.Integration;
using System.Numerics;
using SolversAndIntegrators;

namespace FinalAssignment1
{
    /// <summary>
    /// calculate Heston European option price by formula
    /// </summary>
    public class Hestonformula
    {
        public double u1 = 0.5;
        public double u2 = -0.5;
        public Complex i = new Complex(0.0, 1.0);

        public const int numModelParams = 5;

        private const int nvIndex = 0;
        private const int kIndex = 1;
        private const int thetaIndex = 2;
        private const int rhoIndex = 3;
        private const int sigmaIndex = 4;
```

```

    public double k;
    public double theta;
    public double rho;
    public double sigma;
    public double nv;
    public double phi;

    public Hestonformula()
    {
    }

    public Hestonformula(Hestonformula otherModel) :
        this(otherModel.nv, otherModel.k, otherModel.theta, otherModel.rho,
otherModel.sigma)
    {
    }

    public Hestonformula(double nv, double k, double theta, double rho,
double sigma)
    {
        this.nv = nv;
        this.k = k;
        this.theta = theta;
        this.rho = rho;
        this.sigma = sigma;
    }

    public Hestonformula(double[] paramsArray)
        : this(paramsArray[nvIndex], paramsArray[kIndex],
paramsArray[thetaIndex], paramsArray[rhoIndex], paramsArray[sigmaIndex])
    {
    }

    public double GetNv() { return nv; }
    public double GetK() { return k; }
    public double GetTheta() { return theta; }
    public double GetRho() { return rho; }
    public double GetSigma() { return sigma; }

    /// <summary>
    /// the formula for calculating

```

```

    /// </summary>
    /// <param name="S">Initial Stock Price</param>
    /// <param name="r">Risk Free Rate</param>
    /// <param name="T">Maturity</param>
    /// <param name="K">Strike Price</param>
    /// <returns>the final price of Euorpean option</returns>
    public double CalculateCallOptionPrice(double S, double K, double r,
double T)
    {

        double b1 = k - rho * sigma;
        double b2 = k;
        double a = k * theta;
        if (sigma <= 0 || T <= 0 || K <= 0 || S <= 0)
            throw new System.ArgumentException("Need sigma > 0, T > 0, K > 0
and S > 0.");

        Complex d1(double phi) { return Complex.Sqrt(Complex.Pow(rho * sigma
* phi * i - b1, 2.0) - sigma * sigma * (2.0 * u1 * phi * i - phi * phi)); }
        Complex d2(double phi) { return Complex.Sqrt(Complex.Pow(rho * sigma
* phi * i - b2, 2.0) - sigma * sigma * (2.0 * u2 * phi * i - phi * phi)); }
        Complex g1(double phi) { return (b1 - rho * sigma * phi * i -
d1(phi)) / (b1 - rho * sigma * phi * i + d1(phi)); }
        Complex g2(double phi) { return (b2 - rho * sigma * phi * i -
d2(phi)) / (b2 - rho * sigma * phi * i + d2(phi)); }
        Complex C1(double tau, double phi) { return r * phi * i * tau + (a /
Math.Pow(sigma, 2.0)) * ((b1 - rho * sigma * phi * i - d1(phi)) * tau - 2.0 *
Complex.Log((1.0 - g1(phi) * Complex.Exp(-tau * d1(phi))) / (1.0 -
g1(phi)))); }
        Complex C2(double tau, double phi) { return r * phi * i * tau + (a /
Math.Pow(sigma, 2.0)) * ((b2 - rho * sigma * phi * i - d2(phi)) * tau - 2.0 *
Complex.Log((1.0 - g2(phi) * Complex.Exp(-tau * d2(phi))) / (1.0 -
g2(phi)))); }
        Complex D1(double tau, double phi) { return ((b1 - rho * sigma * phi
* i - d1(phi)) / Math.Pow(sigma, 2.0)) * ((1.0 - Complex.Exp(-tau * d1(phi))) /
(1.0 - g1(phi) * Complex.Exp(-tau * d1(phi)))); }
        Complex D2(double tau, double phi) { return ((b2 - rho * sigma * phi
* i - d2(phi)) / Math.Pow(sigma, 2.0)) * ((1.0 - Complex.Exp(-tau * d2(phi))) /
(1.0 - g2(phi) * Complex.Exp(-tau * d2(phi)))); }
        Complex Phi1(double x, double phi) { return Complex.Exp(C1(T, phi) +
D1(T, phi) * nv + i * phi * x); }
        Complex Phi2(double x, double phi) { return Complex.Exp(C2(T, phi) +
D2(T, phi) * nv + i * phi * x); }
        Complex integrand1(double x, double phi) { return (Complex.Exp(-i *
phi * Math.Log(K)) * Phi1(x, phi)) / (i * phi); }

```

```

        Complex integrand2(double x, double phi) { return (Complex.Exp(-i *
phi * Math.Log(K)) * Phi2(x, phi)) / (i * phi); }

double P1(double x)
{

    Func<double, double> IntegrateFormula
    = (phi) => {
        Complex aj = integrand1(x, phi);
        return aj.Real;
    };
    CompositeIntegrator integrator = new CompositeIntegrator(4);
    /*double integral = integrator.Integrate(IntegrateFormula,
0.0001, 10000, 10000);*/
    double integral =
NewtonCotesTrapeziumRule.IntegrateComposite(IntegrateFormula, 0.0001, 10000,
10000);

    /*double integral =
DoubleExponentialTransformation.Integrate(IntegrateFormula, 0.01, 1e3, 1e-5);*/
    return 0.50 + (1 / Math.PI) * integral;
}
double P2(double x)
{

    Func<double, double> IntegrateFormula
    = (phi) => {
        Complex al = integrand2(x, phi);
        return al.Real;
    };
    CompositeIntegrator integrator = new CompositeIntegrator(4);
    /*double integral = integrator.Integrate(IntegrateFormula,
0.0001, 10000, 10000);*/
    double integral2 =
NewtonCotesTrapeziumRule.IntegrateComposite(IntegrateFormula, 0.0001, 10000,
10000);

    /*double integral =
DoubleExponentialTransformation.Integrate(IntegrateFormula, 0.01, 1e3, 1e-5);*/
    return 0.50 + (1 / Math.PI) * integral2;
}
return S * P1(Math.Log(S)) - Math.Exp(-r * T) * K * P2(Math.Log(S));

}

```

```

        public double GetPutFromCallPrice(double S, double K, double r, double
T, double callPrice)
        {
            return callPrice - S + K * Math.Exp(-r * T);
        }

        public double CalculatePutOptionPrice(double S, double K, double r,
double T)
        {
            if (sigma <= 0 || T <= 0 || K <= 0 || S <= 0)
                throw new System.ArgumentException("Need sigma > 0, T > 0, K > 0
and S > 0.");

            return GetPutFromCallPrice(S, K, r, T, CalculateCallOptionPrice(S,
K, r, T));
        }

        // For use in calibration
        public double[] ConvertHestonModelCalibrationParamsToArray()
        {
            double[] paramsArray = new double[Hestonformula.numModelParams];
            paramsArray[nvIndex] = nv;
            paramsArray[kIndex] = k;
            paramsArray[thetaIndex] = theta;
            paramsArray[rhoIndex] = rho;
            paramsArray[sigmaIndex] = sigma;
            return paramsArray;
        }
    }
}

```

Task 2.3 (Heston Call/Put with Monte Carlo).

GeneratePath.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

using MathNet.Numerics.Distributions;
using MathNet.Numerics.LinearAlgebra;

namespace FinalAssignment3
{
    /// <summary>
    /// this class is for generate MC paths
    /// </summary>
    public class GeneratePath
    {
        public double r;
        public double T;
        public double S;
        public int N;
        public double k;
        public double theta;
        public double rho;
        public double sigma;
        public double nv;
        public double tau;
        public int n;

        public GeneratePath(double S, double r, double T, double nv, double k,
double theta, double rho, double sigma, int N, int n)
        {
            if (sigma <= 0 || N <= 0)
                throw new System.ArgumentException("Need sigma >0, N > 0.");
            this.S = S;
            this.T = T;
            this.nv = nv;
            this.k = k;
            this.theta = theta;
            this.rho = rho;
            this.sigma = sigma;
            this.r = r;
            this.N = N;
            this.n = n;
        }

        /// <summary>
        /// this is for generate all the items on one path
        /// </summary>
        /// <param name="S">Initial Stock Price</param>

```

```

    /// <param name="r">Risk Free Rate</param>
    /// <param name="T">Maturity</param>
    /// <param name="nv">parameter in model</param>
    /// <param name="k">parameter in model</param>
    /// <param name="theta">parameter in model</param>
    /// <param name="rho">parameter in model</param>
    /// <param name="sigma">parameter in model</param>
    /// <param name="N">Number Of Time Steps</param>
    /// <param name="n">Number Of Trials</param>
    /// <returns>a double[] which contains all items we generated</returns>
    public static double[] Generateallpath(double S, double r, double T,
double nv, double k, double theta, double rho, double sigma, int N, int n)
    {
        double alpha = (4 * k * theta - sigma * sigma) / 8;
        double beta = -k / 2;
        double gamma = sigma / 2;
        int i;
        if (N <= 0 || S <= 0 || T <= 0)
            throw new System.ArgumentException("Need N, S, T > 0");
        double tau = T / N;
        double[] s = new double[N + 1];
        double[] y = new double[N + 1];
        s[0] = S;
        y[0] = Math.Sqrt(nv);
        double[] DeltaZ1 = new double[N+1];
        double[] DeltaZ2 = new double[N+1];
        double[] x1 = new double[N];
        double[] x2 = new double[N];
        Normal.Samples(x1, 0, 1);
        Normal.Samples(x2, 0, 1);
        for (i = 0; i < N; i++)
        {
            DeltaZ1[i+1] = Math.Sqrt(tau) * x1[i];
            DeltaZ2[i+1] = Math.Sqrt(tau) * (rho * x1[i] + Math.Sqrt(1 - rho
* rho) * x2[i]);
            y[i + 1] = (y[i] + gamma * DeltaZ2[i + 1]) / (2 * (1 - beta *
tau)) + Math.Sqrt(Math.Pow(y[i] + gamma * DeltaZ2[i + 1], 2.0) / (4 *
Math.Pow(1 - beta * tau, 2.0)) + alpha * tau / (1 - beta * tau));
            s[i + 1] = s[i] + r * s[i] * tau + y[i] * s[i] * DeltaZ1[i + 1];
        }
        return s;
    }
}

```



```
}
```

HestonMC.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MathNet.Numerics.Distributions;
using FinalAssignment3;

namespace FinalAssignment2
{
    /// <summary>
    /// how to use Monte Carlo
    /// </summary>
    public class HestonMC
    {
        /// <summary>
        /// For how to calculate the European option price by MC
        /// </summary>
        /// <param name="S">Initial Stock Price</param>
        /// <param name="r">Risk Free Rate</param>
        /// <param name="T">Maturity</param>
        /// <param name="nv">parameter in model</param>
        /// <param name="k">parameter in model</param>
        /// <param name="theta">parameter in model</param>
        /// <param name="rho">parameter in model</param>
        /// <param name="sigma">parameter in model</param>
        /// <param name="N">Number Of Time Steps</param>
        /// <param name="n">Number Of Trials</param>
        /// <returns>the price we generating from Monte Carlo method</returns>
        public double CalculateEuropeanCallOptionPrice(double S, double K,
double r, double T, double nv, double k, double theta, double rho, double
sigma, int N, int n)
        {
            if (S <= 0 || K <= 0 || T <= 0)
                throw new System.ArgumentException("Need S, K, T > 0");
            if (2 * k * theta <= sigma * sigma)
                throw new System.ArgumentException("need to meet Feller
condition");
            else
```

```

        {
            double[] callprice = new double[n];
            for (int i = 0; i < n; i++)
            {
                callprice[i] = GeneratePath.Generateallpath(S, r, T, nv, k,
theta, rho, sigma, N, n)[N];
            }
            double Price = 0.0;
            int j = 0;
            double[] aaa = new double[n];
            for (j = 0; j < n; j++)
            {
                Price = Price + Math.Exp(-r * T) * Math.Max(callprice[j] - K,
0);
                aaa[j] = Math.Max(callprice[j] - K, 0);
            }
            return Price/n;
        }
    }

    public double CalculatePutOptionPrice(double S, double K, double r,
double T, double nv, double k, double theta, double rho, double sigma, int N,
int n)
    {
        if (S <= 0 || K <= 0 || T <= 0)
            throw new System.ArgumentException("Need S, K, T > 0");
        if (2 * k * theta <= sigma * sigma)
            throw new System.ArgumentException("need to meet Feller
condition");
        return CalculateEuropeanCallOptionPrice(S, K, r, T, nv, k, theta,
rho, sigma, N, n) - S + K * Math.Exp(-r * (T));
    }
}
}

```

Task 2.4 (Checking Heston formula and Monte Carlo).

Using MATLAB:

```

>> A = [13.62971224, 22.45250469, 29.99494943, 36.65426829, 78.46884752]
>> B = [4.11345405, 4.32558000, 4.07677150, 3.68627289, 0.78186353]
>> C = [13.53796729, 22.60721989, 29.91676174, 36.70869780, 78.41234438]
>> D = [4.09803719, 4.34401869, 4.08049716, 3.78939481, 0.24732943]

```

```
>> loglog(A, C)
>> loglog(B, D)
```

Task 2.5 (Calibration).

Hestoncalibrator.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using FinalAssignment1;

namespace HestonModel
{
    public class CalibrationFailedException : Exception
    {
        public CalibrationFailedException()
        {
        }
        public CalibrationFailedException(string message)
            : base(message)
        {
        }
    }

    public enum CalibrationOutcome
    {
        NotStarted,
        FinishedOK,
        FailedMaxItReached,
        FailedOtherReason
    };

    public struct HMCallOptionMarketData
    {
        public double optionExercise;
        public double strike;
        public double marketMidPrice;
    }
}
```

```

/// <summary>
/// calculate the error, outcome type and the parameters we calibrated
/// </summary>
public class HestonCalibrator
{
    private const double defaultAccuracy = 10e-3;
    private const int defaultMaxIterations = 1000;
    private double accuracy;
    private int maxIterations;

    private LinkedList<HMCalloptionMarketData> marketOptionsList;
    private double r; // initial interest rate, this is observed, no need to
calibrate to options
    private double S;
    public double nv;
    public double k;
    public double theta;
    public double rho;
    public double sigma;
    private CalibrationOutcome outcome;

    private double[] calibratedParams;

    public HestonCalibrator()
    {
        accuracy = defaultAccuracy;
        maxIterations = defaultMaxIterations;
        marketOptionsList = new LinkedList<HMCalloptionMarketData>();
        r = 0.025;
        S = 100;
        calibratedParams = new double[] { 0.0175, 1.5768, 0.0398, -0.5711,
0.5751 };
    }

    public HestonCalibrator(double r, double S, double accuracy, int
maxIterations)
    {
        this.r = r;
        this.S = S;
        this.accuracy = accuracy;
        this.maxIterations = maxIterations;
        marketOptionsList = new LinkedList<HMCalloptionMarketData>();
        calibratedParams = new double[] { 0.0175, 1.5768, 0.0398, -0.5711,

```

```

0.5751 };
    }

    public void SetGuessParameters(double nv, double k, double theta, double
rho, double sigma)
    {
        Hestonformula m = new Hestonformula(nv, k, theta, rho, sigma);
        calibratedParams = m.ConvertHestonModelCalibrationParamsToArray();
    }

    public void AddObservedOption(double optionExercise, double strike,
double mktMidPrice)
    {
        HMCallOptionMarketData observedOption;
        observedOption.optionExercise = optionExercise;
        observedOption.strike = strike;
        observedOption.marketMidPrice = mktMidPrice;
        marketOptionsList.AddLast(observedOption);
    }

    private const int nvIndex = 0;
    private const int kIndex = 1;
    private const int thetaIndex = 2;
    private const int rhoIndex = 3;
    private const int sigmaIndex = 4;

    //now we figure out the difference
    /// <summary>
    /// calculate the mean square error
    /// </summary>
    /// <param name="m">the array contained the parameters</param>
    /// <returns>the mean square error between Model and Market</returns>
    public double CalcMeanSquareErrorBetweenModelAndMarket(Hestonformula m)
    {
        double meanSqErr = 0;
        foreach (HMCallOptionMarketData option in marketOptionsList)
        {
            double optionExercise = option.optionExercise;
            double strike = option.strike;
            double modelPrice = m.CalculateCallOptionPrice(S, strike, r,
optionExercise);
            double difference = modelPrice - option.marketMidPrice;
            meanSqErr += difference * difference;
        }
    }

```

```

    }
    return meanSqErr;
}

// Used by Alglib minimisation algorithm
public void CalibrationObjectiveFunction(double[] paramsArray, ref
double func, object obj)
{
    Hestonformula m = new Hestonformula(paramsArray);
    func = CalcMeanSquareErrorBetweenModelAndMarket(m);
}

public void Calibrate()
{
    outcome = CalibrationOutcome.NotStarted;
    double[] initialParams = new double[Hestonformula.numModelParams];
    calibratedParams.CopyTo(initialParams, 0); // a reasonable starting
gueses

    double epsg = accuracy;
    double epsf = accuracy; //1e-4;
    double epsx = accuracy;
    double diffstep = 1.0e-6;
    int maxits = maxIterations;
    double stpmax = 0.05;

    alglib.minlbfgsstate state;
    alglib.minlbfgsreport rep;
    alglib.minlbfgscreatef(1, initialParams, diffstep, out state);
    alglib.minlbfgssetcond(state, epsg, epsf, epsx, maxits);
    alglib.minlbfgssetstpmax(state, stpmax);

    // this will do the work
    alglib.minlbfgsoptimize(state, CalibrationObjectiveFunction, null,
null);

    double[] resultParams = new double[Hestonformula.numModelParams];
    alglib.minlbfgsresults(state, out resultParams, out rep);

    Console.WriteLine("Termination type: {0}", rep.terminationtype);
    Console.WriteLine("Num iterations {0}", rep.iterationscount);
    Console.WriteLine("{0}", alglib.ap.format(resultParams, 5));

    if (rep.terminationtype == 1 // relative function
improvement is no more than EpsF.
        || rep.terminationtype == 2 // relative step is no more

```

```

than EpsX.
        || rep.terminationtype == 4)
    {
        // gradient norm is no more than EpsG
        outcome = CalibrationOutcome.FinishedOK;
        // we update the 'initial parameters'
        calibratedParams = resultParams;
    }
    else if (rep.terminationtype == 5)
    { // MaxIts steps was taken
        outcome = CalibrationOutcome.FailedMaxItReached;
        // we update the 'initial parameters' even in this case
        calibratedParams = resultParams;
    }
    else
    {
        outcome = CalibrationOutcome.FailedOtherReason;
        throw new CalibrationFailedException("Heston model calibration
failed badly.");
    }
}

    public void GetCalibrationStatus(ref CalibrationOutcome calibOutcome,
ref double pricingError)
    {
        calibOutcome = outcome;
        Hestonformula m = new Hestonformula(calibratedParams);
        pricingError = CalcMeanSquareErrorBetweenModelAndMarket(m);
    }

    public Hestonformula GetCalibratedModel()
    {
        Hestonformula m = new Hestonformula(calibratedParams);
        return m;
    }
}
}

```

Task 2.7 (Pricing Asian arithmetic option).

HestonAsianoption.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MathNet.Numerics.Distributions;
using FinalAssignment3;

namespace FinalAssignment4
{
    /// <summary>
    /// calculate Heston Asian call and put Price
    /// </summary>
    public class HestonAsianoption
    {

        private void CheckAsianOptionInputs(IEnumerable<double> T, double
exerciseT)
        {
            if (T.Count() == 0)
                throw new System.ArgumentException("Need at least one monitoring
date for Asian option.");

            if (T.ElementAt(0) <= 0)
                throw new System.ArgumentException("The first monitoring date
must be positive.");

            for (int i = 1; i < T.Count(); ++i)
            {
                if (T.ElementAt(i-1) >= T.ElementAt(i))
                    throw new System.ArgumentException("Monitoring dates must be
increasing.");
            }

            if (T.ElementAt(T.Count()-1) > exerciseT)
                throw new System.ArgumentException("Last monitoring dates must
not be greater than the exercise time.");
        }
    }
}
```



```

    }
    /// <summary>
    ///
    /// </summary>
    /// <param name="S">Initial Stock Price</param>
    /// <param name="r">Risk Free Rate</param>
    /// <param name="T">Maturity</param>
    /// <param name="K">Strike Price</param>
    /// <param name="exerciseT">Monitoring Times</param>
    /// <param name="nv">parameter in model</param>
    /// <param name="k">parameter in model</param>
    /// <param name="theta">parameter in model</param>
    /// <param name="rho">parameter in model</param>
    /// <param name="sigma">parameter in model</param>
    /// <param name="N">Number Of Time Steps</param>
    /// <param name="n">Number Of Trials</param>
    /// <returns>the price of Asian option</returns>
    public double CalculateAsianCallOptionPrice(double S, double K, double
r, IEnumerable<double> T, double exerciseT, double nv, double k, double theta,
double rho, double sigma, int N, int n)
    {
        if (S <= 0 || K <= 0 || exerciseT <= 0)
            throw new System.ArgumentException("Need S, K, T > 0");

        CheckAsianOptionInputs(T, exerciseT);

        if (2.0 * k * theta <= sigma * sigma)
            throw new System.ArgumentException("need to meet Feller
condition");
        else
        {
            int M = T.Count();
            double tau = exerciseT / N;
            double[] givenprice = new double[M];
            double[] callprice = new double[n];
            double[] path = new double[N];
            int number;
            double priceitself;
            double price;
            double Price;
            for (int u = 0; u < n; u++)
            {
                path = GeneratePath.Generateallpath(S, r, exerciseT, nv, k,
theta, rho, sigma/*, tau*/ , N, n);

```

```

        for (int i = 0; i < M; i++)
        {
            number = (int)(T.ElementAt(i) / tau);
            givenprice[i] = path[number];
        }
        priceitself = givenprice.Sum();
        price = Math.Exp(-r * exerciseT) * Math.Max(priceitself / M -
K, 0);

        callprice[u] = price;
    }
    Price = callprice.Sum();
    return Price / n;
}
}

public double CalculateAsianPutOptionPrice(double S, double K, double r,
IEnumerable<double> T, double exerciseT, double nv, double k, double theta,
double rho, double sigma, int N, int n)
{
    if (S <= 0 || K <= 0 || exerciseT <= 0)
        throw new System.ArgumentException("Need S, K, T > 0");

    CheckAsianOptionInputs(T, exerciseT);

    if (2.0 * k * theta <= sigma * sigma)
        throw new System.ArgumentException("need to meet Feller
condition");
    else
    {
        int M = T.Count();
        double tau = exerciseT / N;
        double[] givenprice = new double[M];
        double[] callprice = new double[n];
        double[] path = new double[N];
        int number;
        double priceitself;
        double price;
        double Price;
        for (int u = 0; u < n; u++)
        {
            path = GeneratePath.Generateallpath(S, r, exerciseT, nv, k,
theta, rho, sigma/*, tau*/ , N, n);
            for (int i = 0; i < M; i++)
            {
                number = (int)(T.ElementAt(i) / tau);

```



```

    /// <param name="sigma">parameter in model</param>
    /// <param name="N">Number Of Time Steps</param>
    /// <param name="n">Number Of Trials</param>
    /// <returns>the price of look back option</returns>
    public double CalculateLookbackCallOptionPrice(double S, double r,
double T, double nv, double k, double theta, double rho, double sigma, int N,
int n)
    {
        if (S <= 0 || T <= 0)
            throw new System.ArgumentException("Need S, K, T > 0");
        if (2.0 * k * theta <= sigma * sigma)
            throw new System.ArgumentException("need to meet Feller
condition");
        else
        {
            double Priceitself = 0.0;
            double[] callprice = new double[n];
            for (int i = 0; i < n; i++)
            {
                callprice[i] = Math.Exp(-
r*T)*(GeneratePath.Generateallpath(S, r, T, nv, k, theta, rho, sigma/*, tau*/,
N, n)[N] - GeneratePath.Generateallpath(S, r, T, nv, k, theta, rho, sigma, N,
n).Min());
            }
            for (int j = 0; j < n; j++)
            {
                Priceitself = Priceitself + callprice[j];
            }
            return Priceitself / n;
        }
    }
}

```

solution.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using FinalAssignment1;
using FinalAssignment2;

```

```

using FinalAssignment4;
using FinalAssignment5;
using FinalAssignment6;
using FinalAssignment7;

namespace ConsoleApp2
{
    /// <summary>
    /// give certain value of all parameters and have the final results
    /// </summary>
    class Program
    {
        static void Main(string[] args)
        {
            double r = 0.10;
            double theta = 0.06;
            double k = 2.00;
            double sigma = 0.40;
            double rho = 0.5;
            double nv = 0.04;
            /*double r = 0.025;
            double theta = 0.0398;
            double k = 1.5768;
            double sigma = 0.5751;
            double rho = -0.5711;
            double nv = 0.0175;*/

            double S = 100;
            double T1 = 1.0;
            double T2 = 2.0;
            double T3 = 3.0;
            double T4 = 4.0;
            double T5 = 15.0;
            double K = 100.0;
            double C1, C2, C3, C4, C5, C6, C7, C8, C9, C10;
            double[] parameters = new double[5];
            parameters[0] = nv;
            parameters[1] = k;
            parameters[2] = theta;
            parameters[3] = rho;
            parameters[4] = sigma;
            Hestonformula r1 = new Hestonformula(parameters);
            Console.WriteLine("These are from Heston formula call option:");
        }
    }
}

```

```

C1 = r1.CalculateCallOptionPrice(S, K, r, T1);
Console.WriteLine(C1);
C2 = r1.CalculateCallOptionPrice(S, K, r, T2);
Console.WriteLine(C2);
C3 = r1.CalculateCallOptionPrice(S, K, r, T3);
Console.WriteLine(C3);
C4 = r1.CalculateCallOptionPrice(S, K, r, T4);
Console.WriteLine(C4);
C5 = r1.CalculateCallOptionPrice(S, K, r, T5);
Console.WriteLine(C5);
Console.WriteLine("These are from Heston formula put option:");
C6 = r1.CalculatePutOptionPrice(S, K, r, T1);
Console.WriteLine(C6);
C7 = r1.CalculatePutOptionPrice(S, K, r, T2);
Console.WriteLine(C7);
C8 = r1.CalculatePutOptionPrice(S, K, r, T3);
Console.WriteLine(C8);
C9 = r1.CalculatePutOptionPrice(S, K, r, T4);
Console.WriteLine(C9);
C10 = r1.CalculatePutOptionPrice(S, K, r, T5);
Console.WriteLine(C10);


int n = 100000;
int N = 365;
double tau1 = T1 / N;
double tau2 = T2 / N;
double tau3 = T3 / N;
double tau4 = T4 / N;
double tau5 = T5 / N;
double A1, A2, A3, A4, A5, A6, A7, A8, A9, A10;
HestonMC p1 = new HestonMC();
Console.WriteLine("These are from Heston European call option MC:");
A1 = p1.CalculateEuropeanCallOptionPrice(S, K, r, T1, nv, k, theta,
rho, sigma, N, n);
Console.WriteLine(A1);
A2 = p1.CalculateEuropeanCallOptionPrice(S, K, r, T2, nv, k, theta,
rho, sigma, N, n);
Console.WriteLine(A2);
A3 = p1.CalculateEuropeanCallOptionPrice(S, K, r, T3, nv, k, theta,
rho, sigma, N, n);
Console.WriteLine(A3);
A4 = p1.CalculateEuropeanCallOptionPrice(S, K, r, T4, nv, k, theta,
rho, sigma, N, n);

```

```

        Console.WriteLine(A4);
        A5 = p1.CalculateEuropeanCallOptionPrice(S, K, r, T5, nv, k, theta,
rho, sigma, N, n);
        Console.WriteLine(A5);
        Console.WriteLine("These are from Heston European put option MC:");
        A6 = p1.CalculatePutOptionPrice(S, K, r, T1, nv, k, theta, rho,
sigma, N, n);
        Console.WriteLine(A6);
        A7 = p1.CalculatePutOptionPrice(S, K, r, T2, nv, k, theta, rho,
sigma, N, n);
        Console.WriteLine(A7);
        A8 = p1.CalculatePutOptionPrice(S, K, r, T3, nv, k, theta, rho,
sigma, N, n);
        Console.WriteLine(A8);
        A9 = p1.CalculatePutOptionPrice(S, K, r, T4, nv, k, theta, rho,
sigma, N, n);
        Console.WriteLine(A9);
        A10 = p1.CalculatePutOptionPrice(S, K, r, T5, nv, k, theta, rho,
sigma, N, n);
        Console.WriteLine(A10);


        double t1 = 1.0;
        double t2 = 3.0;
        double t3 = 5.0;
        double t4 = 7.0;
        double t5 = 9.0;
        double K1, K2, K3, K4, K5;
        HestonLookbackoption w1 = new HestonLookbackoption();
        Console.WriteLine("These are from Look back option MC:");
        K1 = w1.CalculateLookbackCallOptionPrice(S, r, t1, nv, k, theta,
rho, sigma, N, n);
        Console.WriteLine(K1);
        K2 = w1.CalculateLookbackCallOptionPrice(S, r, t2, nv, k, theta,
rho, sigma, N, n);
        Console.WriteLine(K2);
        K3 = w1.CalculateLookbackCallOptionPrice(S, r, t3, nv, k, theta,
rho, sigma, N, n);
        Console.WriteLine(K3);
        K4 = w1.CalculateLookbackCallOptionPrice(S, r, t4, nv, k, theta,
rho, sigma, N, n);
        Console.WriteLine(K4);
        K5 = w1.CalculateLookbackCallOptionPrice(S, r, t5, nv, k, theta,
rho, sigma, N, n);

```

```

Console.WriteLine(K5);

/*double[] T11 = { 0.75, 1.00 };
double[] T22 = { 0.25, 0.50, 0.75, 1.00, 1.25, 1.50, 1.75 };
double[] T33 = { 1.00, 2.00, 3.00 };*/
IEnumerable<double> T11 = new double[] { 0.75, 1.00 };
IEnumerable<double> T22 = new double[] { 0.25, 0.50, 0.75, 1.00,
1.25, 1.50, 1.75 };
IEnumerable<double> T33 = new double[] { 1.00, 2.00, 3.00 };
double O1, O2, O3, O4, O5, O6;
HestonAsianoption v1 = new HestonAsianoption();
Console.WriteLine("These are from Asian call option MC:");
O1 = v1.CalculateAsianCallOptionPrice(S, K, r, T11, T1, nv, k,
theta, rho, sigma, N, n);
Console.WriteLine(O1);
O2 = v1.CalculateAsianCallOptionPrice(S, K, r, T22, T2, nv, k,
theta, rho, sigma, N, n);
Console.WriteLine(O2);
O3 = v1.CalculateAsianCallOptionPrice(S, K, r, T33, T3, nv, k,
theta, rho, sigma, N, n);
Console.WriteLine(O3);
Console.WriteLine("These are from Asian put option MC:");
O4 = v1.CalculateAsianPutOptionPrice(S, K, r, T11, T1, nv, k, theta,
rho, sigma, N, n);
Console.WriteLine(O4);
O5 = v1.CalculateAsianPutOptionPrice(S, K, r, T22, T2, nv, k, theta,
rho, sigma, N, n);
Console.WriteLine(O5);
O6 = v1.CalculateAsianPutOptionPrice(S, K, r, T33, T3, nv, k, theta,
rho, sigma, N, n);
Console.WriteLine(O6);

double r0 = 0.025;
double S0 = 100;
double theta1 = 0.0398;
double k1 = 1.5768;
double sigma1 = 0.5751;
double rho1 = -0.5711;
double nv1 = 0.0175;

Hestonformula model = new Hestonformula(nv1, k1, theta1, rho1,
sigma1);

```



```

double[] optionExerciseTimes = new double[] { 1, 1, 2, 2, 1.5 };
double[] optionStrikes = new double[] { 80, 90, 80, 100, 100 };
double[] prices = new double[] { 25.72, 18.93, 30.49, 19.36, 16.58};

HestonCalibrator calibrator = new HestonCalibrator(r0, S0, 1e-3,
1000);

calibrator.SetGuessParameters(nv1, k1, theta1, rho1, sigma1);
for (int i = 0; i < optionExerciseTimes.Length; ++i)
{
    calibrator.AddObservedOption(optionExerciseTimes[i],
optionStrikes[i], prices[i]);
}
calibrator.Calibrate();
double error = 0;
CalibrationOutcome outcome = CalibrationOutcome.NotStarted;
calibrator.GetCalibrationStatus(ref outcome, ref error);
Console.WriteLine("Calibration outcome: {0} and error: {1}",
outcome, error);

double U1, U2, U3, U4, U5;
HestonRainbowoptionprice e1 = new HestonRainbowoptionprice();
Console.WriteLine("These are from Rainbow option MC:");
U1 = e1.CalculateRainbowOptionPrice(S, K, r, T1, nv, k, theta, rho,
sigma, N, n);
Console.WriteLine(U1);
U2 = e1.CalculateRainbowOptionPrice(S, K, r, T2, nv, k, theta, rho,
sigma, N, n);
Console.WriteLine(U2);
U3 = e1.CalculateRainbowOptionPrice(S, K, r, T3, nv, k, theta, rho,
sigma, N, n);
Console.WriteLine(U3);
U4 = e1.CalculateRainbowOptionPrice(S, K, r, T4, nv, k, theta, rho,
sigma, N, n);
Console.WriteLine(U4);
U5 = e1.CalculateRainbowOptionPrice(S, K, r, T5, nv, k, theta, rho,
sigma, N, n);
Console.WriteLine(U5);
Console.ReadKey();
}
}
}

```

Exploration.

HestonRainbowoption.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using FinalAssignment3;

namespace FinalAssignment7
{
    /// <summary>
    /// calculate the Heston Rainbow option price
    /// </summary>
    public class HestonRainbowoptionprice
    {
        /// <summary>
        /// how to calculate the price by using the Monte Carlo and payoff
        formula
        /// </summary>
        /// <param name="S">Initial Stock Price</param>
        /// <param name="r">Risk Free Rate</param>
        /// <param name="T">Maturity</param>
        /// <param name="K">Strike Price</param>
        /// <param name="nv">parameter in model</param>
        /// <param name="k">parameter in model</param>
        /// <param name="theta">parameter in model</param>
        /// <param name="rho">parameter in model</param>
        /// <param name="sigma">parameter in model</param>
        /// <param name="N">Number Of Time Steps</param>
        /// <param name="n">Number Of Trials</param>
        /// <returns>calculate the price from the payoff formula</returns>
        public double CalculateRainbowOptionPrice(double S, double K, double r,
double T, double nv, double k, double theta, double rho, double sigma, int N,
int n)
        {
            if (S <= 0 || T <= 0)
                throw new System.ArgumentException("Need S, K, T > 0");
            if (2.0 * k * theta <= sigma * sigma)
                throw new System.ArgumentException("need to meet Feller
condition");
        }
    }
}
```

```

        else
        {
            double priceitself;
            double[] price = new double[n];
            for (int j = 0; j < n; j++)
            {
                price[j] = Math.Exp(-r * T) *
Math.Max(GeneratePath.Generateallpath(S, r, T, nv, k, theta, rho, sigma, N,
n).Max(), K);
            }
            priceitself = price.Sum();
            return priceitself / n;
        }
    }
}

```

About interface.

Hestonmodelparameter.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using HestonModel.Interfaces;
using HestonModel;

namespace HestonModel
{
    public class Hestonoption : IOption
    {
        private double T;
        public Hestonoption() { this.T = 1; }
        public Hestonoption(double T) { this.T = T; }
        public double Maturity{ get { return T; } }
    }

    public class Hestonparameter : IVarianceProcessParameters
    {
        private double k;
    }
}

```

```

private double theta;
private double sigma;
private double nv;
private double rho;
public Hestonparameter()
{
    this.k = 1.5768;
    this.theta = 0.0398;
    this.sigma = 0.5751;
    this.nv = 0.0175;
    this.rho = -0.5711;
}
public Hestonparameter(double k, double theta, double sigma, double nv,
double rho)
{
    this.k = k;
    this.theta = theta;
    this.sigma = sigma;
    this.nv = nv;
    this.rho = rho;
}
public double Kappa { get { return k; } }
public double Theta { get { return theta; } }
public double Sigma { get { return sigma; } }
public double V0 { get { return nv; } }
public double Rho { get { return rho; } }
}

}

public class HestonMCsetting : IMonteCarloSettings
{
    private int n;
    private int N;
    public HestonMCsetting()
    {
        this.n = 100000;
        this.N = 365;
    }
    public HestonMCsetting (int n, int N)
    {
        this.n = n;
        this.N = N;
    }
    public int NumberOfTrials { get { return n; } }
}

```

```

        public int NumberOfTimeSteps { get { return N; } }
    }

    public class HestonCalibrationSetting : ICalibrationSettings
    {
        private double accuracy;
        private int maximumNumberOfIterations;
        public HestonCalibrationSetting()
        {
            this.accuracy = 0.001;
            this.maximumNumberOfIterations = 1000;
        }
        public HestonCalibrationSetting (double accuracy, int
maximumNumberOfIterations)
        {
            this.accuracy = accuracy;
            this.maximumNumberOfIterations = maximumNumberOfIterations;
        }
        public double Accuracy { get { return accuracy; } }
        public int MaximumNumberOfIterations { get { return
maximumNumberOfIterations; } }
    }

    public class Hestonoptionmarketdata : IOptionMarketData<IOption>
    {
        private IOption option;
        private double Marketprice;
        public Hestonoptionmarketdata()
        {
            this.option = new Hestonoption(1);
            this.Marketprice = 25.72;
        }
        public Hestonoptionmarketdata(IOption option, double Marketprice)
        {
            this.option = option;
            this.Marketprice = Marketprice;
        }
        public IOption Option{ get { return option; } }
        public double Price { get { return Marketprice; } }
    }

    public class Hestonmodelparameter : IHestonModelParameters
    {
        private double S;

```

```

        private double r;
        private IVarianceProcessParameters varianceParameters;
        public Hestonmodelparameter()
        {
            this.S = 100;
            this.r = 0.1;
            this.varianceParameters = new Hestonparameter(2, 0.06, 0.4, 0.04,
0.5);
        }
        public Hestonmodelparameter(double S, double r,
IVarianceProcessParameters varianceParameters)
        {
            this.S = S;
            this.r = r;
            this.varianceParameters = varianceParameters;
        }
        public double InitialStockPrice { get { return S; } }
        public double RiskFreeRate { get { return r; } }
        public IVarianceProcessParameters VarianceParameters { get { return
varianceParameters; } }
    }

    public class HestonEuro : IEuropeanOption
    {
        private PayoffType type;
        private double K;
        private double T;
        public HestonEuro()
        {
            this.type = PayoffType.Call;
            this.K = 100;
            this.T = 1;
        }
        public HestonEuro(PayoffType type, double K, double T)
        {
            this.type = type;
            this.K = K;
            this.T = T;
        }
        public PayoffType Type { get { return type; } }
        public double StrikePrice { get { return K; } }
        public double Maturity { get { return T; } }
    }

```

```

public class HestonAsian : IAsianOption
{
    private IEnumerable<double> Tm;
    private PayoffType type;
    private double K;
    private double T;
    public HestonAsian()
    {
        this.Tm = new double[] { 0.75, 1.00};
        this.type = PayoffType.Call;
        this.K = 100;
        this.T = 1;
    }
    public HestonAsian(IEnumerable<double> Tm, PayoffType type, double K,
double T)
    {
        this.Tm = Tm;
        this.type = type;
        this.K = K;
        this.T = T;
    }
    public IEnumerable<double> MonitoringTimes { get { return Tm; } }
    public PayoffType Type { get { return type; } }
    public double StrikePrice { get { return K; } }
    public double Maturity { get { return T; } }
}
}

```

Heston.cs:

```

using System;
using System.Collections.Generic;
using HestonModel.Interfaces;
using HestonModel;
using FinalAssignment1;
using FinalAssignment2;
using FinalAssignment4;
using FinalAssignment5;

```

```

namespace HestonModel
{

```

```

    /// <summary>
    /// This class will be used for grading.

```

/// Don't remove any of the methods and don't modify their signatures. Don't change the namespace.

/// Your code should be implemented in other classes (or even projects if you wish), and the relevant functionality should only be called here and outputs returned.

/// You don't need to implement the interfaces that have been provided if you don't want to.

/// </summary>

public static class Heston

{

/// <summary>

/// Method for calibrating the heston model.

/// </summary>

/// <param name="guessModelParameters">Object implementing IHestonModelParameters interface containing the risk-free rate, initial stock price

/// and initial guess parameters to be used in the calibration.</param>

/// <param name="referenceData">A collection of objects implementing IOptionMarketData<IEuropeanOption> interface. These should contain the reference data used for calibration.</param>

/// <param name="calibrationSettings">An object implementing ICalibrationSettings interface.</param>

/// <returns>Object implementing IHestonCalibrationResult interface which contains calibrated model parameters and additional diagnostic information</returns>

public static IHestonCalibrationResult

CalibrateHestonParameters(IHestonModelParameters guessModelParameters,

IEnumerable<IOptionMarketData<IEuropeanOption>> referenceData,

ICalibrationSettings calibrationSettings)

{

HestonCalibrator calibrator = new

HestonCalibrator(guessModelParameters.RiskFreeRate,

guessModelParameters.InitialStockPrice,

calibrationSettings.Accuracy,

calibrationSettings.MaximumNumberOfIterations);

calibrator.SetGuessParameters(guessModelParameters.VarianceParameters.V0,

guessModelParameters.VarianceParameters.Kappa,

guessModelParameters.VarianceParameters.Theta,


```

guessModelParameters.VarianceParameters.Rho,

guessModelParameters.VarianceParameters.Sigma);
    foreach (IOptionMarketData<IEuropeanOption> M in referenceData)
    {
        calibrator.AddObservedOption(M.Option.StrikePrice,
M.Option.Maturity, M.Price);
    }
    calibrator.Calibrate();
    double error = 0;
    CalibrationOutcome outcome = CalibrationOutcome.NotStarted;
    calibrator.GetCalibrationStatus(ref outcome, ref error);
}

/// <summary>
/// Price a European option in the Heston model using the Heston formula. This should
be accurate to 5 decimal places
/// </summary>
/// <param name="parameters">Object implementing IHestonModelParameters
interface, containing model parameters.</param>
/// <param name="europeanOption">Object implementing IEuropeanOption interface,
containing the option parameters.</param>
/// <returns>Option price</returns>
public static double HestonEuropeanOptionPrice(IHestonModelParameters parameters,
                                                IEuropeanOption
europeanOption)
{
    Hestonformula r1 = new Hestonformula( parameters.VarianceParameters.V0,

parameters.VarianceParameters.Kappa,

parameters.VarianceParameters.Theta,

parameters.VarianceParameters.Rho,

parameters.VarianceParameters.Sigma);
    if (europeanOption.Type == PayoffType.Call)
    {
        return Math.Round(r1.CalculateCallOptionPrice(parameters.InitialStockPrice,
                                                        europeanOption.StrikePrice,
                                                        parameters.RiskFreeRate,
                                                        europeanOption.Maturity), 5);
    }
    else

```

```

        {
            return Math.Round(r1.CalculatePutOptionPrice(parameters.InitialStockPrice,
                                                         europeanOption.StrikePrice,
                                                         parameters.RiskFreeRate,
                                                         europeanOption.Maturity), 5);
        }
    }

    /// <summary>
    /// Price a European option in the Heston model using the Monte-Carlo method.
    Accuracy will depend on number of time steps and samples
    /// </summary>
    /// <param name="parameters">Object implementing IHestonModelParameters
    interface, containing model parameters.</param>
    /// <param name="europeanOption">Object implementing IEuropeanOption interface,
    containing the option parameters.</param>
    /// <param name="monteCarloSimulationSettings">An object implementing
    IMonteCarloSettings object and containing simulation settings.</param>
    /// <returns>Option price</returns>
    public static double HestonEuropeanOptionPriceMC(IHestonModelParameters
    parameters,
                                                         IEuropeanOption
    europeanOption,
                                                         IMonteCarloSettings
    monteCarloSimulationSettings)
    {
        HestonMC p1 = new HestonMC();
        if (europeanOption.Type == PayoffType.Call)
        {
            return p1.CalculateEuropeanCallOptionPrice(parameters.InitialStockPrice,

    europeanOption.StrikePrice,

    parameters.RiskFreeRate,

    europeanOption.Maturity,

    parameters.VarianceParameters.V0,

    parameters.VarianceParameters.Kappa,

    parameters.VarianceParameters.Theta,

    parameters.VarianceParameters.Rho,

```

```

parameters.VarianceParameters.Sigma,

monteCarloSimulationSettings.NumberOfTimeSteps,

monteCarloSimulationSettings.NumberOfTrials);
    }
    else
    {
        return p1.CalculatePutOptionPrice(parameters.InitialStockPrice,
                                           europeanOption.StrikePrice,
                                           parameters.RiskFreeRate,
                                           europeanOption.Maturity,
                                           parameters.VarianceParameters.V0,

parameters.VarianceParameters.Kappa,

parameters.VarianceParameters.Theta,

parameters.VarianceParameters.Rho,

parameters.VarianceParameters.Sigma,

monteCarloSimulationSettings.NumberOfTimeSteps,

monteCarloSimulationSettings.NumberOfTrials);
    }
}

    /// <summary>
    /// Price a Asian option in the Heston model using the
    /// Monte-Carlo method. Accuracy will depend on number of time steps and
samples</summary>
    /// <param name="parameters">Object implementing IHestonModelParameters
interface, containing model parameters.</param>
    /// <param name="asianOption">Object implementing IAsian interface, containing the
option parameters.</param>
    /// <param name="monteCarloSimulationSettings">An object implementing
IMonteCarloSettings object and containing simulation settings.</param>
    /// <returns>Option price</returns>
    public static double HestonAsianOptionPriceMC(IHestonModelParameters parameters,
                                                  IAsianOption asianOption,
                                                  IMonteCarloSettings
monteCarloSimulationSettings)

```

```

    {
        HestonAsianoption v1 = new HestonAsianoption();
        if (asianOption.Type == PayoffType.Call)
        {
            return v1.CalculateAsianCallOptionPrice(parameters.InitialStockPrice,
                                                    asianOption.StrikePrice,
                                                    parameters.RiskFreeRate,

asianOption.MonitoringTimes,

                                                    asianOption.Maturity,

parameters.VarianceParameters.V0,

parameters.VarianceParameters.Kappa,

parameters.VarianceParameters.Theta,

parameters.VarianceParameters.Rho,

parameters.VarianceParameters.Sigma,

monteCarloSimulationSettings.NumberOfTimeSteps,

monteCarloSimulationSettings.NumberOfTrials);
        }
        else
        {
            return v1.CalculateAsianPutOptionPrice(parameters.InitialStockPrice,
                                                    asianOption.StrikePrice,
                                                    parameters.RiskFreeRate,

asianOption.MonitoringTimes,

                                                    asianOption.Maturity,

parameters.VarianceParameters.V0,

parameters.VarianceParameters.Kappa,

parameters.VarianceParameters.Theta,

parameters.VarianceParameters.Rho,

parameters.VarianceParameters.Sigma,

```

```

monteCarloSimulationSettings.NumberOfTimeSteps,

monteCarloSimulationSettings.NumberOfTrials);
    }
}

/// <summary>
/// Price a lookback option in the Heston model using the
/// a Monte-Carlo method. Accuracy will depend on number of time steps and samples
</summary>
/// <param name="parameters">Object implementing IHestonModelParameters
interface, containing model parameters.</param>
/// <param name="maturity">An object implementing IOption interface and containing
option's maturity</param>
/// <param name="monteCarloSimulationSettings">An object implementing
IMonteCarloSettings object and containing simulation settings.</param>
/// <returns>Option price</returns>
public static double HestonLookbackOptionPriceMC(IHestonModelParameters
parameters,
                                IOption maturity,
                                IMonteCarloSettings
monteCarloSimulationSettings)
{
    HestonLookbackoption w1 = new HestonLookbackoption();
    return w1.CalculateLookbackCallOptionPrice(parameters.InitialStockPrice,
                                                parameters.RiskFreeRate,
                                                maturity.Maturity,

parameters.VarianceParameters.V0,

parameters.VarianceParameters.Kappa,

parameters.VarianceParameters.Theta,

parameters.VarianceParameters.Rho,

parameters.VarianceParameters.Sigma,

monteCarloSimulationSettings.NumberOfTimeSteps,

monteCarloSimulationSettings.NumberOfTrials);
    }
}
}

```