UNIVERSITY OF BIRMINGHAM

C++ for Finance
Assessment 2
Importance Sampling

1222781

Date: Mar 2017

School of Social Sciences
University of Birmingham
Birmingham, B15 2TT

Contents

1 Briefing

This C++ project is designed to assess Importance Sampling method in valuing European cal and put options. The program is object based and is written in ISO C++ 11 using DevC++ 5.11 compiler. The project contains files in the following categories :

Invocation chain

- main.cpp
- Application_Wrapper.cpp
- ErrorHandler.cpp

IO [Input and Output]

- GrandInput.cpp
- Input.cpp
- Output.cpp

Methods

- IS_Method.cpp
- MC_Method.cpp
- Method_Base.cpp

Processes

- GBM_Process.cpp
- IS_Process.cpp

Options

- EuroCall.cpp
- EuroPut.cpp
- OptionBase.cpp

Uncategorised

- Accumulator.cpp
- Valuation.cpp

libraries

- AnalyticalFormulae.cpp
- StopWatch.cpp
- lib_val.cpp
- rv_library.cpp
- utility.cpp

1.1 Copy Right

In order to construct this project different source files were used from the C++ for finance course files. The libraries are exact identical of the libraries given in the course. The other source file used are modified for the perpose of this program.

2 Importance Sampling

Importance Sampling is a estimation technique of part of distribution that has low probability. In our case, probability of a horizon price of 160 for current price of 100 in step of 1 day has a very low probability. Therefore, estimating the price of calls and puts using plain Monte Carlo estimation will have significant uncertainty (standard error).

3 Functionality

All the inputs of this program is hard-wried inside the IO files. In this program each Input object is information of an option (current price, volatility, interest rate, exercise price, validity, option type). However the GradInput object defines the input objects that are going to be assessed.

3.1 Valuation Object

The valuation object is the core of the program. The valuation object takes and input as its argument to understand the details of the option that it is assessing. Note that assessing an option is defined by finding it's value or estimating it's value and calculating estimation's accuracy. The object will then create an IS_method object and MC_method object. The job of IS_method object is to assess the option using Importance Sampling, and the job of MC_method is to assess the object using Monte Carlo method without any addition (plain).
Alongside the methods, the object will create the related option object (e.g for a call it will create a Erocall object). It also values the option using explicit solution to the Black-Scholes equation to benchmark against the estimation methods.
Once the Valuation object is constructed all these relevant objects are also constructed as private members of it. Then, when the Valuation object is requested to be ran (using its public run() function), the object will then supply the methods with the right option object and request the methods to run.
The Valuation object keeps the explicit solution as it's private member. However the solution to the methods will be stored within the method objects.
The valuation is also responsible to calculate the spped_up (efficiency gain) between the methods using each method's standard error and run time.

(a) Error Handling

If the input request for an option to be built that is not identified by the Valuation, the object will throw an error within its construction phase.

3.2 Methods

The method objects are polymorphic, they all have public functions to be ran and supply estimation information after running. When a method object is created, it creates an Accumulator object as its private member. The job of the accumulator is to accumulate the random possible future payoffs and estimate the value of the option using Monte Carlo Estimation.
The accumulator accumulates the payoffs when the method is ran. At that point the mthod supply the accumulator with M (default to be 100,000 - stored in the input option) random possible payoffs from the payoff probability distribution.
When a method is requested to be ran, the method will create a process object that is responsible for distribution of future payoffs.
The MC_method creates a GBM_Process object that takes the inputs and gives out random possible future prices. the method object use the possible future price to calculate the possible future payoff using the option's payoff calculator function.
The method objects meaure their ruuning time using StopWatch object. It stores the run time as a private member.

(a) Error Handling

The method objects are designed to throw an error if option value is requested from them before they are ran. This is to ensure that the user can not obtain wrong estimation value.

3.3 Output

The output object's responsibility is to store the outputs of the Valuation object and display such information to the console if asked. To set the output, the object take the Valuation object as it's input and extract output information from it using getter functions.

3.4 Application_Wrapper

The task of the Application_Wrapper object is to extract the inputs, supply the inputs to the Valuation object, run the Valuation object, Extract the outputs to the Output function, and requesting output to display the outputs to the user. It basically wrap everything up. It perform such action by calling it's run function. This object is also in charge of catching run time error for each valuation.

3.5 The main

The main displays the banner of the project and request the application wrapper to run. It will then prompt the user to press q to exit the application after the application wrapper has finished running.

4 Running

When the application is complied and ran, it will display information related the valuation of each input. figure 1 shows an example of the running.

```
\\\\\\\\\ Importance Sampling Project ///////////////////
\\\\\\\\\          By: 1222781          ///////////////////
>>>>>>>>>>>>>>>> Option Evaluator running <<<<<<<<<<<<<<<<
Call Option
Exercise Price:  140

Explicit Solution =  0.784965

IS Method Output
Option value    0.785308
se              0.000394868
Time taken      0.072

Plain Method Output
Option value    0.772977
se              0.0133345
Time taken      0.011

speed up is     174.225
>>>>>>>>>>>>>>>> Option Evaluator running <<<<<<<<<<<<<<<<
Call Option
Exercise Price:  160

Explicit Solution =  0.158954

IS Method Output
Option value    0.158991
se              4.74142e-005
Time taken      0.071

Plain Method Output
Option value    0.162684
se              0.00610741
Time taken      0.011

speed up is     2570.58
```

Figure 1: Screen-shot of the console

5 Results

The output of the program is summarised into tables.

European Call Option			
Exercise Price	140	160	180
Explicit	0.784965	0.158954	0.0286429
Plain MC	0.77 (0.01) $[0.019]$	0.161 (0.006) $[0.017]$	0.028 (0.002) $[0.012]$
IS MC	0.7853 (0.0004) $[0.12]$	0.15899 (0.00005) $[0.111]$	0.028647 (0.000005) $[0.111]$
Speed_up	176.483	2391.43	34547.2

European Put Option			
Exercise Price	40	50	60
Explicit	1.59921×10^{-6}	0.000333342	0.0112929
Plain MC	0 0 $[0.016]$	0.0003 (1×10^{-4}) $[0.017]$	0.0109 (0.0007) $[0.016]$
IS MC	1.6001×10^{-6} (1×10^{-9}) $[0.107]$	3.3336×10^{-4} (3×10^{-7}) $[0.108]$	1.1304×10^{-2} (1×10^{-5}) $[0.116]$
Speed_up	0(N/A)	20997.5	624.189

6 Conclusion

As expected the result demonstrates that the importance sampling is an efficient method of calculating OTM options with higher accuracy.

7 Source Codes

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//   main.cpp
//   By : 1222781
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "Application_Wrapper.h"
#include "Errorhandler.h"

#include <stdexcept>
#include <iostream>
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//   main()
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

int main(int argc, char *argv[])
{
   std::cout << "\\\\\\\\\\\\\\\\\\ Importance Sampling Project //////////////////   " << std::endl;
   std::cout << "\\\\\\\\\\\\\\\\\\          By: 1222781          //////////////////   " << std::endl;

    ErrorHandler the_handler;

    try
    {
        Application_Wrapper app;
        app.run();
    }
    catch(const std::runtime_error & e)
    {
        the_handler.HandleRunTimeError(e);
    }
    catch (...)
    {
        the_handler.HandleUnknownError();
    }

    return the_handler.PauseAndReturn();
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//   end of file
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 1: main.cpp.

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//   ErrorHandler.h
//   By : 1222781
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ErrorHandlerH
#define ErrorHandlerH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include <stdexcept>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//   class ErrorHandler
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ErrorHandler
{
    public:
```

```
18        explicit ErrorHandler();
19          void HandleRunTimeError(const std::runtime_error & e);
20          void HandleUnknownError();
21          long PauseAndReturn();
22  };
23
24  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
25  #endif
26  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
27  //   End
28  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 2: ErrorHandler.h

```
1   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2   // ErrorHandler.cpp
3   //   By : 1222781
4   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6   #include "ErrorHandler.h"
7
8   #include "utility.h"
9
10  #include <stdexcept>
11  #include <iostream>
12
13  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
14  // constructor
15  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
16  ErrorHandler::ErrorHandler()
17  {
18
19  }
20
21  void ErrorHandler::HandleRunTimeError(const std::runtime_error & e)
22  {
23      std::cout << "Error caught:  " << e.what() << std::endl;
24  }
25
26  void ErrorHandler::HandleUnknownError()
27  {
28      std::cout << "Unknown error caught" << std::endl;
29  }
30
31  long ErrorHandler::PauseAndReturn()
32  {
33     return ut::PauseAndReturn();
34  }
35
36  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
37  // end
38  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 3: ErrorHandler.cpp

```
1   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2   //   Application_Wrapper.h
3   //   By : 1222781
4   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5   #ifndef Application_WrapperH
6   #define Application_WrapperH
7   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
8
9   class Input;
10  class Output;
```

```
11 class StopWatch;
12 class Valuation;
13
14 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
15 //  class Application_Wrapper
16 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
17
18 class Application_Wrapper
19 {
20     public:
21         Application_Wrapper();
22         ~Application_Wrapper();
23
24         void run();
25
26     private:
27
28 };
29
30 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
31 #endif
32 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
33 //  End
34 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 4: Application_Wrapper.h

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  //  Application_Wrapper.cpp
3  //  By : 1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6  #include "Application_Wrapper.h"
7
8  #include "GrandInput.h"
9  #include "Input.h"
10 #include "Output.h"
11
12 #include "Valuation.h"
13
14 #include "ErrorHandler.h"
15
16 #include <vector>
17 #include <stdexcept>
18
19 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
20 // constructor
21 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
22
23 Application_Wrapper::Application_Wrapper()
24 {
25
26 }
27
28 Application_Wrapper::~Application_Wrapper()
29 {
30
31 }
32
33 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
34 // run
35 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
36
37 void Application_Wrapper::run()
38 {
39    ErrorHandler the_handler;
40
```

```
41    GrandInput ginp;  //loading inputs from the depository
42    std::vector<Input> inputs =  ginp.GetVec();
43
44    double imp_size = inputs.size();
45
46     for (double i = 0; i < imp_size; ++i)
47    {
48        Input inp = inputs[i];
49        Output out;
50
51       try
52        {
53            Valuation val(inp, out);
54          val.run();
55        }
56        catch(const std::runtime_error & e)
57        {
58            the_handler.HandleRunTimeError(e);
59        }
60        catch(...)
61        {
62            the_handler.HandleUnknownError();
63        }
64
65
66       out.SetOutput(val);
67       out.DoOutput();
68    }
69
70 }
71
72 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
73 // end
74 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 5: Application_Wrapper.cpp

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  //  Input.h
3  //  By : 1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5  #ifndef InputH
6  #define InputH
7  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
8
9  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
10 //  class Input
11 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
12
13 class Input
14 {
15     public:
16         explicit Input(double X, char o_type);
17
18         double GetS0() const {return S_0_;}
19         double Getr() const {return r_;}
20         double Getsig() const {return sig_;}
21         double GetX() const {return X_;}
22         double GetT() const {return T_;}
23         long GetM() const {return M_;}
24         char GetOptionType() const {return o_type_;}
25
26     private:
27         double S_0_;
28         double r_;
29         double sig_;
30
```

```
31          double X_;
32          double T_;
33
34          long M_;      // number of sample paths
35
36          char o_type_;
37  };
38
39  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
40  #endif
41  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
42  //  End
43  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 6: Input.h

```
1   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2   // Input.cpp
3   //  By : 1222781
4   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6   #include "Input.h"
7
8   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
9   // interface
10  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
11
12  Input::Input(double X, char o_type)
13  {
14     S_0_ = 100; //Current asset price
15     r_ = 0.05;  //Interest Rate
16     sig_ = 0.2; //Volatility
17
18     X_ = X;     //Exercise
19     T_ = 1;     //Expiery
20
21     M_ = 100000; //Number of payoffs for MC
22
23
24     o_type_ = o_type;     // c for call, p for put
25
26  }
27
28  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
29  // end
30  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 7: Input.cpp

```
1   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2   //  GrandInput.h
3   //  By : 1222781
4   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5   #ifndef GrandInputH
6   #define GrandInputH
7   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
8
9   class Input;
10
11  #include <vector>
12
13  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
14  //  class GrandInput
15  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
16
```

```
class GrandInput
{
    public:
        explicit GrandInput();

        std::vector<Input> GetVec() const {return inp_vec_;}

    private:
        std::vector<Input> inp_vec_;
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 8: GrandInput.h

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// GrandInput.cpp
//  By : 1222781
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "GrandInput.h"
#include "Input.h"

#include <vector>
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// interface
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

GrandInput::GrandInput()
{
   char put = 'p';
   char call = 'c';

   // creating three call inputs
   for (double x = 140; x <= 180; x += 20)
   {
       Input imp(x, call);
       inp_vec_.push_back(imp);
   }

   //creating three put inputs
   for (double x = 40; x <= 60; x += 10)
   {
       Input imp(x, put);
       inp_vec_.push_back(imp);
   }

}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// end
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 9: GrandInput.cpp

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Output.h
//  By : 1222781
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef OutputH
#define OutputH
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include <string>

class Valuation;
class StopWatch;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class Output
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class Output
{
    public:
        void SetOutput(Valuation & val);
        void DoOutput();
        void OutputBanner(std::string strg);
        void OutputCounter(long j, long M, long interval);

    private:
        double expl_ ;
      double is_se_ ;
      double is_val_ ;
      double is_t_ ;

      double pl_se_ ;
      double pl_val_ ;
      double pl_t_ ;

      double speed_up_ ;
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 10: Output.h

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// Output.cpp
//  By : 1222781
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "Output.h"

#include "Valuation.h"
#include "StopWatch.h"

#include "utility.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// interface
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void Output::SetOutput(Valuation & val)
{
   expl_ = val.Get_expl();        //Explicit Solution

    is_se_ = val.Get_IS_SE();
    is_val_ = val.Get_IS_Value();
    is_t_ = val.Get_IS_Time();

    pl_se_ = val.Get_PL_SE();
    pl_val_ = val.Get_PL_Value();
    pl_t_ = val.Get_PL_Time();
```

```
28
29      speed_up_ = val.Get_Speed_up();
30  }
31
32  void Output::DoOutput()
33  {
34     ut::OutputLine("Explicit Solution = ", expl_);
35
36     ut::OutputLine("");
37     ut::OutputLine("Plain Method Output");
38      ut::OutputLine("Option value  ", is_val_);
39      ut::OutputLine("se            ", is_se_);
40      ut::OutputLine("Time taken    ", is_t_);
41     ut::OutputLine("");
42
43     ut::OutputLine("IS Method Output");
44      ut::OutputLine("Option value  ", pl_val_);
45      ut::OutputLine("se            ", pl_se_);
46      ut::OutputLine("Time taken    ", pl_t_);
47      ut::OutputLine("");
48
49      ut::OutputLine("speed up is   ", speed_up_);
50  }
51
52  void Output::OutputBanner(std::string strg)
53  {
54      ut::OutputLine(strg);
55  }
56
57  void Output::OutputCounter(long j, long M, long interval)
58  {
59      ut::OutputCounter(j, M, interval);
60  }
61
62  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
63  // end
64  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 11: Output.cpp

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  //   Valuation.h
3  //   By : 1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5  #ifndef ValuationH
6  #define ValuationH
7  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
8
9  class OptionBase;
10 class Method_Base;
11 class IS_process;
12
13 class IS_MC_method;
14 class Input;
15 class Output;
16
17 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
18 //   class Valuation
19 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
20
21 class Valuation
22 {
23     public:
24         Valuation(Input & inp, Output & out);
25         ~Valuation();
26
27         double Get_expl() const;
```

```
28
29        double Get_IS_SE() const;
30        double Get_IS_Value()const;
31        double Get_IS_Time() const;
32
33        double Get_PL_SE() const;
34        double Get_PL_Value() const;
35        double Get_PL_Time() const;
36
37        double Get_Speed_up() const;
38
39          void run();
40
41      private:
42         OptionBase * opt_;
43
44         Method_Base * is_mth_;
45         Method_Base * pl_mth_;
46
47         OptionBase * CreateOption(const Input & inp);
48
49         double BS_Valuation(const Input & inp);
50         double expl_;
51
52         void Banner(Input & inp) const;
53 };
54
55 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
56 #endif
57 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
58 //   End
59 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 12: Valuation.h

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  //  Valuation.cpp
3  //   By : 1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6  #include "Valuation.h"
7
8  #include "Input.h"
9  #include "Output.h"
10
11 #include "OptionBase.h"
12 #include "EuroCall.h"
13 #include "EuroPut.h"
14
15 #include "IS_process.h"
16
17 #include "Method_Base.h"
18 #include "IS_method.h"
19 #include "MC_method.h"
20
21 #include "AnalyticalFormulae.h"
22 #include "utility.h"
23
24 #include <stdexcept>
25
26
27 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
28 // constructor
29 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
30
31 Valuation::Valuation(Input & inp, Output & out)
32 {
```

```
33      opt_ = CreateOption(inp);
34
35     expl_ = BS_Valuation(inp);
36
37      is_mth_ = new IS_method(inp, out);
38      pl_mth_ = new MC_method(inp, out);
39
40      Banner(inp);
41  }
42
43  Valuation::~Valuation()
44  {
45      delete opt_;
46      delete is_mth_;
47      delete pl_mth_;
48  }
49
50  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
51  // Banner Generator
52  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
53
54  void Valuation::Banner(Input & inp) const
55  {
56     ut::OutputLine(">>>>>>>>>>>>>>>> Option Evaluator running <<<<<<<<<<<<<<<<<<<");
57      ut::OutputLine( opt_->Identity() );
58      ut::OutputLine( "Exercise Price: ", inp.GetX());
59      ut::OutputLine("");
60  }
61
62  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
63  // getters
64  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
65
66  double Valuation::Get_expl() const {return expl_;}
67
68  double Valuation::Get_IS_SE() const {return is_mth_->GetOptionSE();}
69  double Valuation::Get_IS_Value()const {return is_mth_->GetOptionValue();}
70  double Valuation::Get_IS_Time() const {return is_mth_->GetTime();}
71
72  double Valuation::Get_PL_SE() const {return pl_mth_->GetOptionSE();}
73  double Valuation::Get_PL_Value() const {return pl_mth_->GetOptionValue();}
74  double Valuation::Get_PL_Time() const {return pl_mth_->GetTime();}
75
76  double Valuation::Get_Speed_up() const
77  {
78     double Pl_se = Get_PL_SE();
79     double Pl_time = Get_PL_Time();
80     double IS_se = Get_IS_SE();
81     double IS_time = Get_IS_Time();
82     return(Pl_se*Pl_se*Pl_time)/(IS_se*IS_se*IS_time);
83  }
84
85  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
86  // run
87  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
88
89  void Valuation::run()
90  {
91      is_mth_->run(*opt_);
92      pl_mth_->run(*opt_);
93  }
94
95  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
96  // CreateOption()
97  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
98
99  OptionBase * Valuation::CreateOption(const Input & inp)
100 {
```

```
101     char o_type = inp.GetOptionType();
102 
103     switch(o_type)
104     {
105         case 'c': return new EuroCall(inp);  break;
106         case 'p': return new EuroPut(inp);   break;
107         default:  throw std::runtime_error("Valuation: CreateOption:  bad option");
108     }
109 }
110 
111 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
112 // CreateOption()
113 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
114 
115 double Valuation::BS_Valuation(const Input & inp)
116 {
117     char o_type = inp.GetOptionType();
118     double S = inp.GetS0();
119    double r = inp.Getr();
120    double sig = inp.Getsig();
121    double T = inp.GetT();
122    double X = inp.GetX();
123 
124     switch(o_type)
125     {
126         case 'c': return AnalyticalFormulae::European_call_GBM( S, r, sig, T, X);  break;
127         case 'p': return AnalyticalFormulae::European_put_GBM( S, r, sig, T, X);   break;
128         default:  throw std::runtime_error("Valuation: BS_Valuation:  bad option");
129     }
130 }
131 
132 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
133 // end
134 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 13: Valuation.h

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  //   OptionBase.h
3  //   By : 1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5  #ifndef OptionBaseH
6  #define OptionBaseH
7  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
8  
9  #include <string>
10 
11 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
12 //   class OptionBase
13 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
14 
15 class OptionBase
16 {
17     public:
18         virtual ~OptionBase(){}
19 
20         virtual std::string Identity() const = 0;
21         virtual double IdentityN() const = 0;
22       virtual double ComputePO(double S) const = 0;
23 };
24 
25 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
26 #endif
27 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
28 //   End
29 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 14: OptionBase.h

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// OptionBase.cpp
//  By : 1222781
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "OptionBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// dummy

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// end
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 15: OptionBase.cpp

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  EuroCall.h
//  By : 1222781
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef EuroCallH
#define EuroCallH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "OptionBase.h"

#include <string>

class Input;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class EuroCall
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class EuroCall : public OptionBase
{
    public:
        explicit EuroCall(const Input & inp);

        std::string Identity() const;
        double IdentityN() const;
        double ComputePO(double S) const;

    private:
        double X_;
        double T_;
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 16: EuroCall.h

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  // EuroCall.cpp
3  //  By : 1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6  #include "EuroCall.h"
7
8  #include "Input.h"
9
10 #include <string>
11
12 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
13 // constructor
14 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
15
16 EuroCall::EuroCall(const Input & inp)
17 :   X_(inp.GetX())
18 ,   T_(inp.GetT())
19 {}
20
21 std::string EuroCall::Identity() const
22 {
23    return "Call Option";
24 }
25
26 double EuroCall::IdentityN() const
27 {
28    return 1;
29 }
30
31 double EuroCall::ComputePO(double S) const
32 {
33     return std::max(0.0, S - X_);
34 }
35
36 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
37 // end
38 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 17: EuroCall.cpp

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  //  EuroPut.h
3  //  By : 1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5  #ifndef EuroPutH
6  #define EuroPutH
7  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
8
9  #include "OptionBase.h"
10
11 #include <string>
12
13 class Input;
14
15 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
16 //  class EuroPut
17 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
18
19 class EuroPut : public OptionBase
20 {
21     public:
22         explicit EuroPut(const Input & inp);
23
24       std::string Identity() const;
25       double IdentityN() const;
26         double ComputePO(double S) const;
```

```
27
28      private:
29          double X_;
30          double T_;
31  };
32
33  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
34  #endif
35  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
36  //   End
37  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 18: EuroPut.h

```
1   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2   // EuroPut.cpp
3   //   By : 1222781
4   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6   #include "EuroPut.h"
7
8   #include "Input.h"
9
10  #include <string>
11
12  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
13  // constructor
14  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
15
16  EuroPut::EuroPut(const Input & inp)
17  :   X_(inp.GetX())
18  ,   T_(inp.GetT())
19  {}
20
21  std::string EuroPut::Identity() const
22  {
23     return "Put Option";
24  }
25
26  double EuroPut::IdentityN() const
27  {
28     return (-1);
29  }
30
31  double EuroPut::ComputePO(double S) const
32  {
33      return std::max(0.0, X_ - S);
34  }
35
36
37  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
38  // end
39  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 19: main.cpp.

```
1   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2   //   Method_Base.h
3   //   By : 1222781
4   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5   #ifndef Method_BaseH
6   #define Method_BaseH
7   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
8
9   class OptionBase;
```

```
10
11 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
12 //   class Method_Base
13 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
14
15 class Method_Base
16 {
17         public:
18           virtual ~Method_Base(){}
19
20          virtual void run(OptionBase & opt) const = 0;
21          virtual double GetOptionValue() const = 0;
22          virtual double GetOptionSE() const = 0;
23          virtual double GetTime() const = 0;
24 };
25
26 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
27 #endif
28 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
29 //   End
30 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 20: Method_Base.h

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  // Method_Base.cpp
3  //   By : 1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6  #include "Method_Base.h"
7
8  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
9  // constructor
10 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
11
12 // dummy
13
14 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
15 // end
16 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 21: Method_Base.cpp

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  //   IS_method.h
3  //   By : 1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5  #ifndef IS_methodH
6  #define IS_methodH
7  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
8
9  #include "Method_Base.h"
10
11 class Input;
12 class Output;
13 class OptionBase;
14 class Process_Base;
15 class Accumulator;
16
17 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
18 //   class MC_method
19 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
20
21 class IS_method : public Method_Base
22 {
```

```
23      public:
24          explicit IS_method(Input & inp, Output & out);
25
26          void run(OptionBase & opt) const;
27
28          double GetOptionValue() const;
29          double GetOptionSE() const;
30          double GetTime() const;
31
32      private:
33          long M_;
34          mutable double t_; //storing the running time.
35
36          Accumulator * acc_;
37          Output * out_;
38          Input * inp_;
39
40          mutable bool run_status_;
41  };
42
43  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
44  #endif
45  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
46  //  End
47  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 22: IS_method.h

```
1   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2   // IS_method.cpp
3   //  By : 1222781
4   //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6   #include "IS_method.h"
7
8   #include "Input.h"
9   #include "Output.h"
10
11  #include "Accumulator.h"
12
13  #include "IS_process.h"
14  #include "StopWatch.h"
15
16  #include <vector>
17  #include <algorithm>
18  #include <cmath>
19  #include <stdexcept>
20
21  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
22  // constructor
23  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
24
25  IS_method::IS_method(Input & inp, Output & out)
26  :   M_(inp.GetM())
27  {
28      acc_ = new Accumulator(inp);
29
30      out_ = &out;
31
32      inp_ = &inp;
33
34      run_status_ = false;
35
36      t_ = 0;
37  }
38
39
```

```
40 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
41 // getters
42 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
43
44 double IS_method::GetOptionValue() const
45 {
46    if (run_status_ == false) throw std::runtime_error("IS_method:  Option Value requested before ↩
       running");
47     return acc_->GetOptionValue();
48 }
49
50 double IS_method::GetOptionSE() const
51 {
52    if (run_status_ == false) throw std::runtime_error("IS_method:  SE requested before running");
53     return acc_->GetSE();
54 }
55
56 double IS_method::GetTime() const
57 {
58    if (run_status_ == false) throw std::runtime_error("IS_method:  Time requested before running")↩
       ;
59     return t_;
60 }
61
62 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
63 // run()
64 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
65
66 void IS_method::run(OptionBase & opt) const
67 {
68     if (run_status_ == true) throw std::runtime_error("IS_method:  invalid run request, method has↩
         been ran and is matured");
69
70    StopWatch stw;
71     stw.StartStopWatch();
72
73    IS_process prc(* inp_, opt);
74
75     std::default_random_engine uni_gen;
76
77    double alpha = prc.Get_g_alpha();
78    double beta = prc.Get_g_beta();
79
80     std::gamma_distribution<double> gamma(alpha, 1.0/beta);
81
82
83
84     for(long j = 1; j <= M_; ++j)
85     {
86
87       double random_gamma = gamma(uni_gen);
88
89          double payoff = prc.Next_P(random_gamma);
90
91          acc_->AddValue(payoff);
92     }
93
94     t_ = stw.GetTime();
95     run_status_ = true;
96 }
97
98 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
99 // end
100 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 23: IS_method.cpp

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//   MC_method.h
//   By : 1222781
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef MC_methodH
#define MC_methodH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include"Method_Base.h"

class Input;
class Output;
class OptionBase;
class Process_Base;
class Accumulator;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//   class MC_method
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class MC_method : public Method_Base
{
    public:
        explicit MC_method(Input & inp, Output & out);

        void run(OptionBase & opt) const;

        double GetOptionValue() const;
        double GetOptionSE() const;
        double GetTime() const;

    private:
        long M_;
        mutable double t_;

        Accumulator * acc_;
        Output * out_;
        Input * inp_;

        mutable bool run_status_ ;
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//   End
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 24: MC_method.h

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// MC_method.cpp
//   By : 1222781
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "MC_method.h"

#include "Input.h"
#include "Output.h"
#include "Accumulator.h"
#include "OptionBase.h"
#include "GBM_process.h"
#include "StopWatch.h"

#include <vector>
#include <stdexcept>
```

```
17
18 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
19 // constructor
20 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
21
22 MC_method::MC_method(Input & inp, Output & out)
23 :   M_(inp.GetM())
24 {
25     acc_ = new Accumulator(inp);
26
27     out_ = &out;
28
29     inp_ = &inp;
30
31     run_status_ = false;
32 }
33
34 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
35 // getters
36 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
37
38 double MC_method::GetOptionValue() const
39 {
40    if (run_status_ == false) throw std::runtime_error("MC_method:  value requested before running"↩
         );
41     return acc_->GetOptionValue();
42 }
43
44 double MC_method::GetOptionSE() const
45 {
46    if (run_status_ == false) throw std::runtime_error("MC_method:  SE requested before running");
47     return acc_->GetSE();
48 }
49
50 double MC_method::GetTime() const
51 {
52    if (run_status_ == false) throw std::runtime_error("MC_method:  Time requested before running")↩
         ;
53     return t_;
54 }
55
56 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
57 // run()
58 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
59
60 void MC_method::run(OptionBase & opt) const
61 {
62     if (run_status_ == true) throw std::runtime_error("MC_method:  invalid run request, method has↩
          been ran and is matured");
63
64     StopWatch stw;
65     stw.StartStopWatch();
66
67     GBM_process prc(* inp_);
68
69     for(long j = 1; j <= M_; ++j)
70     {
71
72         double Next_S = prc.Next_S();
73
74         double payoff = opt.ComputePO(Next_S);
75
76         acc_->AddValue(payoff);
77     }
78
79     t_ = stw.GetTime();
80
81     run_status_ = true;
```

```
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// end
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 25: MC_method.cpp

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//   IS_process.h
//   By : 1222781
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef IS_processH
#define IS_processH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class Input;
class OptionBase;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//   class GBM_process
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class IS_process
{

    public:
        explicit IS_process(Input & inp, OptionBase & opt);

        double Next_P(double ga_rand) ;

        double Get_mu_T() const {return g_alpha_;}
        double Get_sig_T() const {return g_beta_;}
        double Get_g_alpha() const {return g_alpha_;}
        double Get_g_beta() const {return g_beta_;}



    private:
        double mu_T_;
        double sig_T_;

      double S_0_;
        double sig_;
        double r_;
        double T_;
        double X_;

        OptionBase * opt_;

        double g_alpha_;
        double g_beta_;

        double f_S(double S) const;
        double Find_mode();
        double mode_target(double S);
        double hf(double S) const;
      double g_S(double S) const;

};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//   End
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 26: IS_process.h

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  // IS_process.cpp
3  //  By : 1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6  #include "IS_process.h"
7
8  #include "OptionBase.h"
9  #include "Input.h"
10 #include "rv_library.h"
11
12 #include <algorithm>
13 #include <cmath>
14 #include <random>
15 #include <stdexcept>
16
17 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
18 // constructor
19 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
20
21 IS_process::IS_process(Input & inp, OptionBase & opt)
22 :   S_0_(inp.GetS0())
23 ,   sig_(inp.Getsig())
24 ,   r_(inp.Getr())
25 ,  T_(inp.GetT())
26 ,  X_(inp.GetX())
27 {
28    mu_T_ = std::log(S_0_) + (r_ - 0.5*sig_*sig_)*T_;
29     sig_T_ = sig_*std::sqrt(T_);
30     opt_ = &opt;
31     g_alpha_ = 2;
32
33     double In_mode = Find_mode();
34
35    double Mode_difference = (In_mode - X_) * opt.IdentityN();
36     g_beta_ = (g_alpha_ - 1.0)/Mode_difference;
37     //g_beta_ = 0.02;
38     //g_beta_ = 0.2;
39 }
40
41
42 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
43 // Next_P()
44 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
45
46 double IS_process::Next_P(double random)
47 {
48    double S = X_ + random * opt_->IdentityN();  // generates random gamma variate
49     return hf(S)/g_S(S);
50 }
51
52 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
53 // mode_target()
54 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
55 double IS_process::mode_target(double S)
56 {
57     return (S - X_)*(std::log(S) - mu_T_) - X_*sig_T_*sig_T_;
58 }
59
60 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
61 // Find_mode()
62 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
63 double IS_process::Find_mode()
```

```
64 {
65    double O_IdentityN = opt_->IdentityN();
66
67    static const long MAX_ITERATIONS = 100;
68    static const double ROOT_TOLERANCE = 0.000000001;
69
70     double first_S = X_;
71
72     double target = 0.0;
73
74     if(mode_target(first_S) >= 0) throw std::runtime_error("Bad root finding");
75
76     double second_S = X_;
77     do
78     {
79         second_S += X_ * O_IdentityN;
80     }
81     while(mode_target(second_S) <= 0);
82
83     long i = 1;
84
85     while (( (second_S - first_S) * O_IdentityN > ROOT_TOLERANCE) && (i < MAX_ITERATIONS))
86     {
87         double root = 0.5*(second_S + first_S);
88
89         double this_value = mode_target(root) ;
90
91         if (this_value == target) return root;
92
93         if (this_value < target) first_S = root;
94         if (this_value > target) second_S = root;
95
96         ++i;
97    }
98
99     return 0.5*(second_S + first_S);
100 }
101
102 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
103 // f_S() - Probability density function
104 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
105
106 double IS_process::f_S(double S) const
107 {
108     double nS = (std::log(S) - mu_T_)/sig_T_;
109     double w = S*sig_T_*rv::ROOT_TWO_PI;
110
111     return std::exp(-0.5*nS*nS)/w;
112 }
113
114 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
115 // hf() -
116 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
117 double IS_process::hf(double S) const
118 {
119    double p = f_S(S);                  //probability of S
120    double payoff = opt_ -> ComputePO(S);
121     return p*payoff;
122 }
123 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
124 // g_s() - option based
125 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
126 double IS_process::g_S(double S) const
127 {
128    double s = (S - X_) * opt_ -> IdentityN();
129     return g_beta_*std::pow(g_beta_*s, g_alpha_ - 1)*std::exp(-g_beta_*s)/std::exp(rv::gammln(↩
        g_alpha_));
130 }
```

```
131
132 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
133 // end
134 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 27: IS_process.cpp

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  //   GBM_process.h
3  //   By : 1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5  #ifndef GBM_processH
6  #define GBM_processH
7  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
8
9  #include <vector>
10
11 class Input;
12
13 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
14 //   class GBM_process
15 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
16
17 class GBM_process
18 {
19     public:
20         explicit GBM_process(const Input & inp);
21         double Next_S() const;
22
23         double GetS0() const {return S_0_;}
24
25     private:
26         double S_0_;
27         double sig_;
28         double r_;
29
30       double mu_T_;
31       double sig_T_;
32 };
33
34 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
35 #endif
36 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
37 //   End
38 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 28: GBM_process.h

```
1  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  // GBM_process.cpp
3  //   By : 1222781
4  //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5
6  #include "GBM_process.h"
7
8  #include "Input.h"
9  #include "rv_library.h"
10
11 #include <cmath>
12
13 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
14 // constructor
15 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
16
17 GBM_process::GBM_process(const Input & inp)
```

```
18 :   S_0_(inp.GetS0())
19 ,   sig_(inp.Getsig())
20 ,   r_(inp.Getr())
21 {
22
23     double T = inp.GetT();
24
25     mu_T_ = std::log(S_0_) + (r_ - 0.5*sig_*sig_)*T;
26     sig_T_ = sig_*std::sqrt(T);
27 }
28
29
30 double GBM_process::Next_S() const
31 {
32     double z = rv::GetNormalVariate();
33     return std::exp(mu_T_ + sig_T_*z);
34 }
35
36
37
38 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
39 // end
40 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Source Code 29: GBM_process.cpp