FE545 Homework Assignment #1

Due Date: March 5th (Tuesday).

Data: An Asian option is a type of exotic option. Unlike a vanilla European option where the price of the option is dependent upon the price of the underlying asset at expiry, an Asian option pay-off is a function of multiple points up to and including the price at expiry. Thus it is "path-dependent" as the price relies on knowing how the underlying behaved at certain points before expiry. Asian options in particular base their price off the mean average price of these sampled points. To simplify the problem, we will consider equally distributed sample points beginning at time and ending at maturity,

In this problem, we will consider Geometirc mean A of the spot prices we use the following formula:

$$A(0,T) = \exp\left(\frac{1}{N}\sum_{i=1}^{N}\log(S(t_i))\right) \tag{1}$$

Unlike in the vanilla European option Monte Carlo case we have learned in class, where we only needed to generate multiple spot values at expiry, we now need to generate multiple spot paths, each sampled at the correct points. Thus instead of providing a double value representing spot to our option, we now need to provide a $std :: vector < double >$ (i.e. a vector of double values), each element of which represents a sample of the spot price on a particular path. We will still be modeling our asset price path via a Geometric Brownian Motion (GBM), and we will create each path by adding the correct drift and variance at each step in order to maintain the properties of GBM.

Implement PayOff classes according to the following template:

```
#ifndef __PAY_OFF__
#define __PAY_OFF__

#include <algorithm> // This is needed for the std::max
// comparison function, used in the pay-off calculations

class PayOff {
 public:
  PayOff(); // Default (no parameter) constructor
  virtual ~PayOff() {}; // Virtual destructor

  // Overloaded () operator, turns the PayOff into an abstract
  // function object
  virtual double operator(const double& S) const = 0;
};

class PayOffCall : public PayOff {
 private:
  double K; // Strike price

 public:
  PayOffCall(const double& K_);
  virtual ~PayOffCall() {};

  // Virtual function is now over-ridden
  // (not pure-virtual anymore)
  virtual double operator(const double& S) const;
};

class PayOffPut : public PayOff {
 private:
  double K; // Strike

 public:
  PayOffPut(const double& K_);
  virtual ~PayOffPut() {};
  virtual double operator(const double& S) const;
};

#endif
```

For this assignment, you need to define a base pure abstract class called *AsianOption* and a derived class called *AsianOptionGeometric* which implements the PayOff operator according to formula (1).

```
#ifndef __Asian_Option__
#define __Asian_Option__


#include <vector>
#include "payoff.h"

class AsianOption {
 protected:
  PayOff* pay_off;  // Pay-off class (in this instance call or put)

 public:
  AsianOption(PayOff* _pay_off);
  virtual ~AsianOption() {};

  // Pure virtual pay-off operator (this will determine arithmetic o
  virtual double OptionPayOff(const std::vector<double>& spot_prices
};

class AsianOptionGeometric : public AsianOption {
 public:
  AsianOptionGeometric(PayOff* _pay_off);
  virtual ~AsianOptionGeometric() {};

  // Overide the pure virtual function to produce geometric Asian Op
  virtual double OptionPayOff(const std::vector<double>& spot_prices
};

#endif
```

Please add a function in the *Random.h* and *Random.cpp* files which generates a Geometric Brownian Motion path according to the following formula:

$$S(t_i) = S(t_{t-1})\text{exp}[(r - \frac{1}{2}\sigma^2\Delta t) + \sigma\sqrt{\Delta t}\epsilon] \tag{2}$$

The price path can be generated recursively from S_0 from above equation (2). The function prototype should follow the following structure with an input argument $std :: vector < double > \&$.

```
#ifndef __Option_Class__Random__
#define __Option_Class__Random__


double GetOneGaussianByBoxMuller();
void GetGBMSpotPricePath(
                           std::vector<double>& spotPrices,
                           // Vector of spot prices to be filled in
                           const double& r,
                           // Risk free interest rate (constant)
                           const double& v,
                           // Volatility of underlying (constant)
                           const double& T
                           // Expiry
                           );
#endif /* defined(__Option_Class__Random__) */
```

You will also need to implement a *SimpleMonteCarlo* class with a function to implement the simulation procedure:

```
#ifndef __Option_Class__SimpleMonteCarlo__
#define __Option_Class__SimpleMonteCarlo__

#include <iostream>
#include "Vanilla1.h"

double SimpleMonteCarlo3(const AssianOption& TheOption,
                         double Spot,
                         double Vol,
                         double r,
                         unsigned long NumberOfPaths);



#endif /* defined(__Option_Class__SimpleMonteCarlo__) */
```

Overall, your Asian option pricer should include the following files:

- Random.h
- Random.cpp
- PayOff.h
- PayOff.cpp
- AsianOption.h
- AsianOption.cpp
- SimpleMC.h
- SimpleMC.cpp
- AsianOptionMain.cpp

Please use the following parameter and generate Geometric Asian Put and Call option prices:

$$\begin{aligned} T &= 1 \\ S(0) &= 50 \\ K &= 50 \\ \sigma &= 0.30 \\ r &= 0.05 \end{aligned}$$

For each path you will use 250 intervals, and please run the simulation for 1000 times to calculate expected option prices.

Homework Honor Policy: You are allowed to discuss the problems between yourselves, but once you begin writing up your solution, you must do so independently, and cannot show one another any parts of your written solutions.