FE545 Homework Assignment #4

Due Date: April 24th (Wednesday).

Problem 1: Pricing a derivative security entails calculating the expected discounted value of its payoff. This reduces, in principle, to a problem of numerical integration; but in practice this calculation is often difficult for high-dimensional pricing problems. Broadie and Glasserman (1997) proposed the method of the simulated tree to price American options, which can derive the upper and lower bounds for American options. This combination makes it possible to measure and control errors as the computational effort increases. The main drawback of the random tree method is that its computational requirements grow exponentially in the number of exercise dates m, so the method is applicable only when m is small. Nevertheless, for problems with small m it is very effective, and it also serves to illustrate a theme of managing soures of high and low bias.

Let $\tilde{h}_i$ denote the payoff function for exercise at t_i, which we now allow to depend on i. Let $\tilde{V}_i(x)$ denote the value of the option at t_i given $X_i = x_i$ (the option has not exercised). We are ultimately interested in $\tilde{V}_0(X_0)$. This value is determined

recursively as follows:

$$\tilde{V}_m(x) = \tilde{h}_m(x) \quad (1)$$

$$\tilde{V}_{i-1}(x) = max\{\tilde{h}_{i-1}(x), E[D_{i-1,i}(X_i)\tilde{V}_i(X_i)|X_{i-1} = x]\}, \quad (2)$$

$i = 1, ..., m$ and we have introduced the notation $D_{i-1,i}(X_i)$ for the discount factor from t_{i-1} to t_i. It states that the option value at expiration is given by the payoff function $\tilde{h}_m$; and at time $(i-1)$th exercise date the option value is the maximum of the immediate exercise value and the expected present value of continuing.

As its name suggests, the random tree method is based on simulating a tree of paths of the underlying Markov chain $X_0, X_1, ..., X_m$. Fix a branching parameter $b \geq 2$. From the initial state X_0, simulate b independent successor states $X_1^1, ..., X_1^b$ all having the law of X_1. From each X_1^i, simulate b independent successors $X_2^{i1}, ..., X_2^{ib}$ from the conditional law of X_2 given $X_1 = X_1^i$. From each $X_2^{i_1 i_2}$, generate b successors $X_3^{i_1 i_2 1}, ..., X_3^{i_1 i_2 b}$, and so on. We denote a generic node in the tree at time step i by $X_i^{j_1 j_2 \cdots j_i}$. The superscript indicates that this node is reached by following the j_1-th branch out of X_0, the j_2th branch out the next node, and so on.

At all terminal nodes, set the estimator equal to the payoff

at that node:

$$\hat{v}_m^{j_1..j_m} = h_m(X_m^{j_1...j_m}). \tag{3}$$

At node $j_1 j_2 ... j_i$ at time step i, and for each $k = 1, ..., b$, set

$$\hat{v}_{ik}^{j_1 j_2 ... j_i} = \begin{cases} h_i(X_i^{j_1 j_2 ... j_i}) & \text{if } \frac{1}{b}\sum_{j=1}^{b} \hat{v}_{i+1}^{j_1 j_2 ... j_i j} \leq h_i(X_i^{j_1 j_2 ... j_i}); \\ \hat{v}_{i+1}^{j_1 j_2 ... j_i k} & \text{otherwise} \end{cases} \tag{4}$$

Then set

$$\hat{v}_i^{j_1..j_i} = \frac{1}{b}\sum_{k=1}^{b} \hat{v}_{i,k}^{j_1...j_i} \tag{5}$$

Working backward, we then set

$$\hat{V}_i^{j_1..j_i} = max\left\{ h_i(X_i^{j_1...j_i}), \frac{1}{b}\sum_{k=1}^{b} \hat{V}_{i,k}^{j_1...j_i} \right\} \tag{6}$$

The high estimator of the option price at the current time and state is $\hat{v}_0$.

The low estimator is defined as follows. At all terminal nodes, set the estimator equal to the payoff at that node:

$$\hat{v}_m^{j_1 j_2 ... j_m} = h_m(X_m^{j_1 j_2 ... j_m})$$

At node $j_1 j_2 ... j_i$ at time step i, and for each $k = 1, ..., b$, set

$$\hat{v}_{ik}^{j_1 j_2 \cdots j_i} = \begin{cases} h_i(X_i^{j_1 j_2 \cdots j_i}) & \text{if} \frac{1}{b-1} \sum_{j=1; j \neq k}^{b} \hat{v}_{i+1}^{j_1 j_2 \cdots j_i j} \leq h_i(X_i^{j_1 j_2 \cdots j_i}); \\ \hat{v}_{i+1}^{j_1 j_2 \cdots j_i k} & \text{otherwise} \end{cases} \tag{7}$$

We then set

$$\hat{v}_i^{j_1 j_2 \cdots j_i} = \frac{1}{b} \sum_{k=1}^{b} \hat{v}_{ik}^{j_1 j_2 \cdots j_i}. \tag{8}$$

The low estimator of the option price at the current time and state is $\hat{v}_0$.

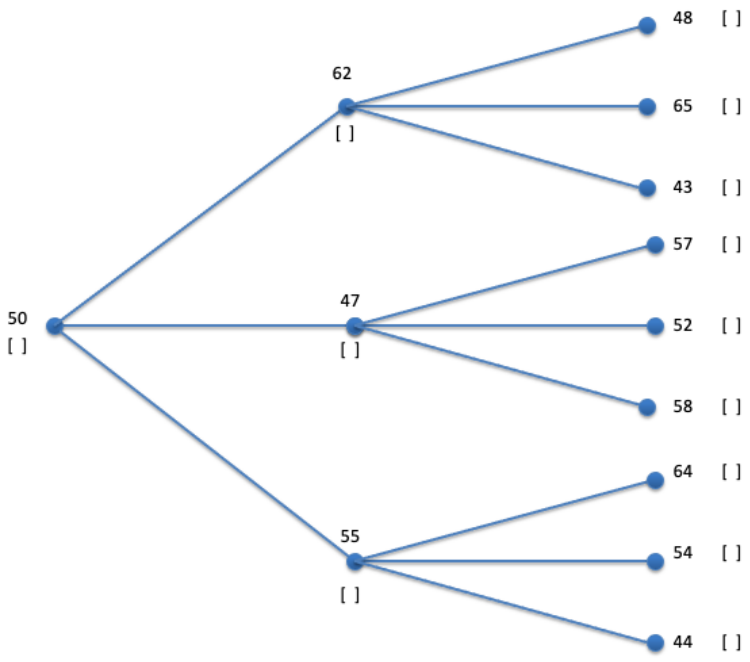

Figure 1: High Estimator

Assume a random tree for pricing American call option is given in Figure (1). Please use the random tree in Figure (1) to calculate the high and low estimate of the American call option price with a strike price of 50. Please show your steps for both the high estimator and low estimator.

Note: You do not need to write C++ program to get the price, but rather manually calculating the estimators following the algorithms introduced in class.

Problem 2: The Black-Scholes formula is sufficiently complicated that there is no analytic inverse function, and therefore this inversion must be carried out numerically. Our objective is to use Newton-Ralphson method and the programming techniques to design a solver in a reusable fashion.

When we have a well-behaved function with an analytic derivative then Newton-Ralphson can be used to find inverse values of the Black-Scholes formula. The idea of Newton-Raphson is that we pretend the function is linear and look for the solution where the linear function predicts it to be. Thus we take a standing point x_0, and approximate f by

$$g_0(x) = f(x_0) + (x - x_0)f'(x_0).\epsilon, \tag{9}$$

We have that $g_0(x)$ equal to zero if and only if

$$x = \frac{y - f(x_0)}{f'(x_0)} + x_0. \tag{10}$$

We therefore take this value as our new guess x_1. We repeat until we find that $f(x_n)$ is within ϵ of y.

In this problem, you will use a **pointer to a member function** which is similar in syntax and idea to a function pointer, but it is restricted to methods of a single class. The difference in syntax is that the class name with a :: must be attached to the $*$ when it is declared. Thus to declare function pointers called $Value$ and $Derivative$ which must point to methods of the class T, we have double (T::*Derivative) (double) double and double (T::*Value) (double) double. The function $Value$ ($Derivative$) is a const member function which takes in a double as argument and outputs a double as return value. If we have an object of class T called $TheObject$ and y is a double, then the function pointed to can be invoked by either $TheObject.*Value(y)$ or $TheObject.*Derivative(y)$. The following is the prototype of such a template class.

```
#include <cmath>

//three template parameters
template<class T, double (T::*Value)(double) const,double (T::*Deriv
    double NewtonRaphson(double Target,
```

```
                    double Start,
                    double Tolerance,
                    const T& TheObject)
{
    double y = (TheObject.*Value)(Start); // pass initial guess to f
    double x=Start;

    while ( fabs(y - Target) > Tolerance )
    {
        double d = (TheObject.*Derivative)(x);
        x+= (Target-y)/d;
        y = (TheObject.*Value)(x); // check the target function valu
    }

    return x;

}
```

Use the classes introduced in Lecture 09 and finish the main.cpp file for European Call Option with the following parameters:

- Expiry: 1
- Spot: 50
- Strike: 50
- Risk free rate: 0.05

- Dividend: 0.08
- Price: 4.23
- Vol initial guess: 0.23
- Tolerance: 0.0001

Use the NewtonRalphson class from the sample as the starting point and find implied volatility for the given option price of 4.23.

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.