

Homework 3

- Author: Ruchit Jerambhai Vithani
- Created on: 10/01/2022
- GTID: 903829303

How to Run

- Download and extract the zip file
- There are four c++ files, namely `main.cpp` , `Option.cpp` , `OptionPrice.cpp` and `UnitTest.cpp` .
- Compile the program using following command

Compiling

```
g++ -o A1 -std=c++11 main.cpp Option.cpp OptionPrice.cpp UnitTest.cpp
```

Running the code

```
./A1
```

Implementation details

BSM model

- The values of d_1 and d_2 are calculated using the formulas given in the `IntroductiontoOptions_Sept17_19.pdf` file in the class.
- Then we compute the value of option using the analytical formula of the Black–Scholes model (again as given in the handout)
- The standard normal function was implemented based on lab 1. It estimates the normal distribution.

Binomial Option Pricing model

- I have set the number of levels in the tree to be a constant number 50. This was the number that
- I first compute entire forward tree. The values of each nodes are stored so that they can be utilised later if required.
 - The tree is stored in a `vector<vector<double>>` data structure.

```
vector<vector<double>> build_tree(double S0, int n, double u, double d){
    vector<vector<double>> result;
    result.push_back(vector<double>{S0});
    for(int i=0;i<n;i++){
        vector<double> previous_layer = result[i];
        vector<double> current_layer;
        for(int j=0;j<previous_layer.size();j++){
            if(j==0){
                current_layer.push_back(previous_layer[j]*u);
                current_layer.push_back(previous_layer[j]*d);
            } else {
                current_layer.push_back(previous_layer[j]*d);
            }
        }
        result.push_back(current_layer);
    }

    return result;
}
```

- Once the forward tree is constructed, we calculate the terminal nodes of the backwards tree using the strike price and the terminal values of the stock price.
- Once the terminal nodes is computed, we compute the value of the risk neutral probability measures p and q to compute the remaining nodes of the tree in backward direction.
- Here also, the entire tree is saved in `vector<vector<double>>` , and thus, saving all the intermediate values in the tree.
- The following function computes the backwards tree

```
vector<vector<double>> compute_price_backwards(const vector<double>& terminal_layer, int n, double discount_factor){
    vector<vector<double>> backwards_tree;
    backwards_tree.push_back(terminal_layer);

    for(int i=0;i<n;i++){
        vector<double> previous_layer = backwards_tree[i];
        vector<double> current_layer;
        for(int j=0;j<previous_layer.size()-1; j++){
            double val = discount_factor * (p * previous_layer[j] + q * previous_layer[j+1]);
            current_layer.push_back(val);
        }
        backwards_tree.push_back(current_layer);
    }
    return backwards_tree;
}
```

Unit Tests

- Online calculator was used to compute the prices of options using black-scholes model.
 - This calculator can be found at <https://goodcalculators.com/black-scholes-calculator/>
 - These test cases were encoded in a function, which is run at the beginning of the main file.
- One test was also run to regenerate the results from Homework 3 in Stochastic Processes in Finance.
 - This test can be found in the results snapshot below.

Results

- Following snapshot contains the results of the experiment.

```
ruchitvithani@ipsec-10-2-194-11:~/Documents/georgia_tech/Assignments/sem-1/System Design for Computational Finance/Homework 3
~/Doc/georgia_tech/A/s/Sy/Homework 3 > g++ -std=c++11 main.cpp Option.cpp OptionPrice.cpp UnitTest.cpp
~/Doc/georgia_tech/A/s/Sy/Homework 3 > ./A1
Running unit tests for BSM pricer model....
S0: 300 K: 250 T: 1 sigma: 0.15 r: 0.03 flag: C Calculated Price: 58.8198 Target Price: 58.82 Result: OK!
S0: 300 K: 250 T: 1 sigma: 0.15 r: 0.03 flag: P Calculated Price: 1.43116 Target Price: 1.43 Result: OK!
S0: 60 K: 58 T: 0.5 sigma: 0.2 r: 0.035 flag: C Calculated Price: 5.0157 Target Price: 5.02 Result: OK
!
S0: 60 K: 58 T: 0.5 sigma: 0.2 r: 0.035 flag: P Calculated Price: 2.00953 Target Price: 2.01 Result: OK
!

Running unit tests for Binomial pricer model
S0: 300 K: 250 T: 1 sigma: 0.15 r: 0.03 flag: C Calculated Price: 58.826 Target Price: 58.82 Result: OK!
S0: 300 K: 250 T: 1 sigma: 0.15 r: 0.03 flag: P Calculated Price: 1.43735 Target Price: 1.43 Result: OK!
S0: 60 K: 58 T: 0.5 sigma: 0.2 r: 0.035 flag: C Calculated Price: 5.01635 Target Price: 5.02 Result: OK
!
S0: 60 K: 58 T: 0.5 sigma: 0.2 r: 0.035 flag: P Calculated Price: 2.01018 Target Price: 2.01 Result: OK
!

Pricing for custom input:
Enter Current Price of Underlying Asset:
36
Enter Strike Price:
40
Enter Risk Free Rate:
0.02
Enter Time to Maturity:
0.416666667
Enter Volatility:
0.3
Enter Option flag:
p

S0: 36 K: 40 T: 0.416667 sigma: 0.3 r: 0.02 flag: p
Binomial Price: 5.10287
BSM Price: 5.1097
Delta value: -0.656975 (N(d1), N(d1) - 1) = (0.343025, -0.656975)
~/Doc/georgia_tech/A/s/Sy/Homework 3 > g++ -std=c++11 main.cpp Option.cpp OptionPrice.cpp UnitTest.cpp
~/Doc/georgia_tech/A/s/Sy/Homework 3 > ./A1
```