C++ in Quantitative Finance part #1 Individual home project

Submitted by**: Nihad Alili (466258)**

Submitted to **: Dr. Pawel Sakowski**

Degree**: Quantitative Finance, Master's degree II year**

Assumptions

- price of the underlying at the moment of option pricing: **S0** = 100,
- strike price **K** = 110,
- annualized volatility rate **σ** = 25%,
- annualized risk-free rate **r** = 5%,
- time to maturity **T** = 0.75,
- barrier level **H** = 125
- number of steps **N** = 1000,
- number of path **M** = 100.000.

1. Methodology

This project aims to price barrier options using Monte Carlo Simulation on C++. First of all, details of barrier options and mathematical background of Monte-Carlo Simulation for this process and secondly, the codes are explained.

1.1 Barrier option and Monte Carlo Simulation

1. **Barrier Options Definition**:
   Barrier options are a type of exotic option whose payoff depends not only on the strike price and the underlying asset's final price but also on whether the asset price crosses a predetermined barrier level during the option's lifetime. There are eight main types, including knock-in (up/down) and knock-out (up/down) call/put options.

2. **Knock-Up-and-Out Call Option**:
   A knock-up-and-out call option behaves like a European call option, but with an additional condition:

   - If the underlying asset price ever reaches or exceeds the barrier level before maturity, the option immediately becomes worthless (knocked out).
   - If the barrier is never breached, then the option behaves like a standard European call option, with payoff at expiration of **MAX** $(\boldsymbol{ST} - \boldsymbol{K}, \mathbf{0})$

3. **Monte Carlo Pricing & Geometric Brownian Motion (GBM)**:

   - The stock price follows a GBM process given by:

$$\boldsymbol{S_{t+dt}} = \boldsymbol{S_t} \cdot \mathbf{exp}\left((\boldsymbol{r} - \mathbf{0.5} \cdot \boldsymbol{\sigma}^{\mathbf{2}}) \cdot \boldsymbol{dt} + \boldsymbol{\sigma} \cdot \sqrt{\boldsymbol{dt}} \cdot \boldsymbol{Z}\right)$$

4. **Simulation & Payoff Calculation**:

   - The stock price is simulated over time using discrete steps.
   - Barrier Condition Check: If at any time step the stock price crosses the barrier, the payoff is set to zero immediately.
   - At maturity, if the barrier was never hit, the payoff is **MAX (0,** $\boldsymbol{ST} - \boldsymbol{K}$).
   - All nonzero payoffs are averaged over all simulations and discounted back to present value using the risk-free rate.

1.2 Code Implementation on C++

1. The **BarrierOption** class is declared in the header file. Inside the class, parameters such as the initial stock price, strike price, volatility, risk-free rate, time to maturity, barrier level, number of time steps, and number of simulations are initialized. The class also includes methods to simulate a single price path and calculate the corresponding payoff, as well as a method to price the option using Monte Carlo simulation.

2. In BarrierOption.cpp source file, the methods declared in header file are defined.
   SimulatePrice() simulates a single price path of the underlying asset using Geometric Brownian Motion (GBM) and checks if the barrier level is breached at any point during the simulation. If the stock price S crosses the barrier level H at any point during the simulation, the option is knocked out, and the method immediately returns a payoff of 0. If the option is not knocked out, the payoff is the maximum of 0 or the difference between the final stock price and the strike price K. The method returns these payoffs for a single simulated price path.
   PriceOption() prices the barrier option using Monte Carlo simulation by running multiple simulations of the stock price path and averaging the payoffs to estimate the option price. The method runs M simulations, calling the simulatePrice method for each simulation, and sums the resulting payoffs. After calculating the average payoff, the method applies the discount factor exp(-r * T) to account for the time value of money. The discounted average payoff is returned as the option price.

3. Overall, Main.cpp gives the user input-output stream and make the file ready for last outcome.

2. Result:

For time steps and number of paths, numbers as a thousand and a hundred thousand were added to the model, respectively. taking into account of robustness of C++. These numbers are chosen based on the trade-off between accuracy and computational time. A large number of iterations would provide a more accurate result, but it would also take longer to run the simulation. The number of iterations can be adjusted depending on the desired level of accuracy and the available computational resources. The results of this simulation should be considered an approximation of the theoretical price of the option and may not be exact.

In the beginning, H = 120 was selected as a barrier level and the result was 0,13 which is much lower than expected before experiment. After that, 130 was selected as a barrier level and the price of the option was 0,63 but It is not believable that price would increase to 130 so often in that time to maturity. As an average, 125 was chosen as barrier level, so It is believed that It can have impact on the price of the option. Result is **0.40787** for the price of up-and-out barrier using call option with parameters S0=100, K=110, σ =25%, r=5%, T=0.75, H=125 after one hundred thousand simulation.

In order to be sure about the correctness of the model, barrier level was increased to 10000 which makes price even impossible to be breached and the result was compared to price of usual European call option which is calculated in Black-Scholes model with the same parameters. The results were so close.

In accordance with the Honor Code, I certify that my answers here are my own work, and I did not make my solutions available to anyone else.