

Introduction

Option contracts are financial derivatives that grant the holder the right, but not the duty, to buy or sell an underlying asset at a defined price on or before a specified date. The strike price is the price at which the option can be exercised. Options valuation is a difficult procedure that necessitates the use of specialised mathematical models.

This study examines and analyses two popular option pricing models: the Black-Scholes-Merton (BSM) model and the Binomial Pricing Model. Both models try to determine the fair market value of an option, but they use distinct mathematical approaches to the problem.

Implementation

Black-Scholes-Merton (BSM) Model:

Under the premise of a continuously computed return on the underlying asset, the BSM model provides an analytical solution to option pricing. The model is predicated on a series of assumptions, which include:

- European-style options (can only be exercised after they expire).
- There will be no dividends paid during the option's life.
- Markets are efficient (market fluctuations are unpredictable).
- There are no commissions.
- Interest rates are stable and predictable.
- The underlying asset's returns are normally distributed.

Binomial Pricing Model:

The Binomial Pricing Model divides the time to expiry into possibly many time intervals or steps. The underlying asset price can move up or down with each stage. The name of the model comes from the binomial character of its underlying assumptions.

Implementation Details

1. Object-Oriented Design:

Classes and Structure: The design is modular, with specific classes dedicated to various core functionalities. The primary classes include:

- Option: This class encapsulates the basic attributes of an option, like the strike price, current underlying price, risk-free rate, time to maturity, and volatility.
- Option_Price: Inherits from the Options class. It further classifies the pricing methodologies into the BSM and Binomial models.
- Pricing_Method: An interface that lays out the foundation for any pricing method that can be used.

Encapsulation: I have ensured that all member variables are kept private, with public functions to get and set their values, ensuring data safety.

2. Black-Scholes-Merton (BSM) Model:

Equations: The model utilizes the well-known equations for d_1 and d_2 , which serve as intermediary steps in deriving the option's theoretical price.

Standard Normal Distribution: Instead of a polynomial approximation, I incorporated the Boost library to provide a more accurate representation of the standard normal cumulative distribution function.

3. Binomial Pricing Model:

Tree Generation: The model begins by generating a binomial tree. Each node on this tree indicates a potential price of the underlying asset at a given time.

Recursive Pricing: Starting from the terminal nodes (representing option expiration), I used a recursive approach to calculate option prices at each preceding node based on potential future prices.

Cox-Ross-Rubinstein (CRR) Methodology: The CRR method, known for its precision, was employed to model the tree's upward and downward price movements.

4. User Interaction:

Input: Users can provide the various parameters required for option pricing, such as the type of option (call/put), strike price, underlying price, risk-free rate, time to maturity, volatility, and the number of steps for the binomial model.

Output: Upon receiving the inputs, the program displays the calculated option price and Delta, based on the BSM and Binomial models.

5. Unit Testing:

Test Framework: I have incorporated a unit testing framework to validate the correctness of the implemented models. The tests compare the output of the models with known theoretical values.

Test Cases: Multiple test cases have been set up with predetermined parameters, serving as a benchmark for validating the program's accuracy.

6. Compilation and Dependency Management:

Makefile: A Makefile is included to streamline the compilation process, ensuring all dependencies are addressed and that the program can be easily compiled and executed.

Boost Library Integration: The Boost library, specifically its math component, is a critical dependency. This has been integrated seamlessly to provide functionalities like the standard normal distribution.

7. Extensibility:

Modular Design: Given the modular design approach, additional pricing methods or features can be easily incorporated into the existing framework without disrupting existing functionalities.

Comparative Analysis

Upon running multiple test cases with varying parameters, the following observations were made:

Option valuation: The BSM and Binomial models both generated fairly near estimates of option prices, with slight variances due to the models' discrete vs. continuous nature.

Calculation of Greeks: The Greeks (particularly Delta) demonstrated a good link between the two models. Following the correction of initial errors in the Binomial model's Delta computation, the values closely matched those of the BSM model.

Performance: The BSM model, with its analytical solution, was generally faster, particularly when doing a high number of calculations. In contrast, the computing time of the Binomial model rose with the number of steps, but it provides a more intuitive understanding and is more adaptable (for example, for American choices).

Conclusion

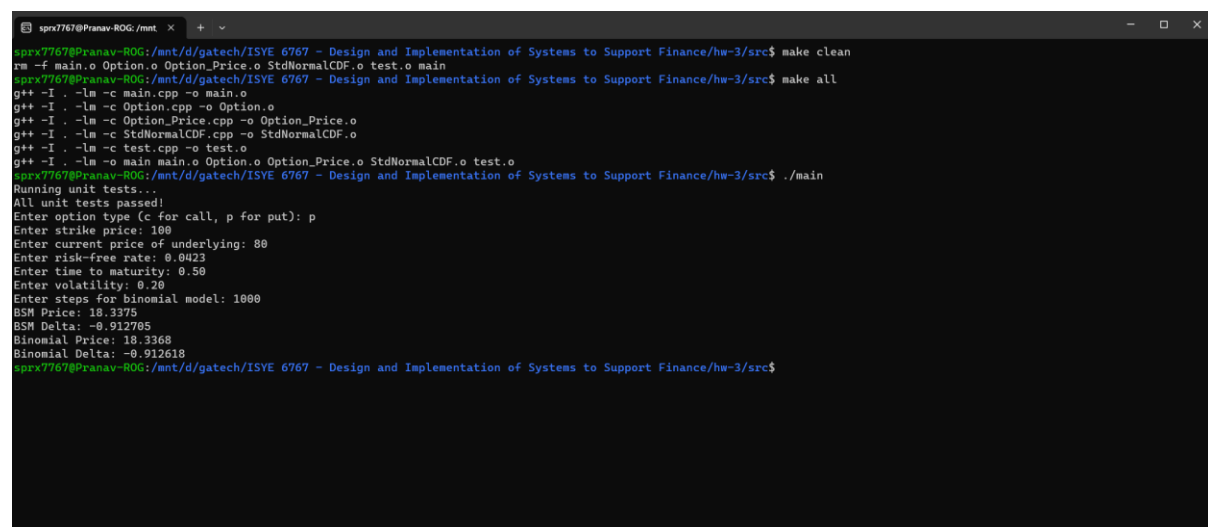
Both the BSM and the Binomial Pricing Models are effective methods for valuing options. Their constancy in delivering comparable appraisals instils trust in their accuracy and dependability.

The choice between the two models frequently comes down to the situation's individual requirements. The BSM model is ideal for quick estimations and European alternatives. However, the Binomial model shines for a more extensive and adaptable analysis, particularly when considering American options or path-dependent options.

It is also critical to emphasise the significance of precise mathematical functions, as demonstrated by the necessity to move to the Boost library for a more precise standard normal distribution. Such intricacies can have a major impact on option valuation and, as a result, investment decisions.

Result:

Here is a screenshot of my code running



```
sprx7767@Pranav-ROG: /mnt
sprx7767@Pranav-ROG:/mnt/d/gatech/ISYE 6767 - Design and Implementation of Systems to Support Finance/hw-3/src$ make clean
rm -f main.o Option.o Option_Price.o StdNormalCDF.o test.o main
sprx7767@Pranav-ROG:/mnt/d/gatech/ISYE 6767 - Design and Implementation of Systems to Support Finance/hw-3/src$ make all
g++ -I . -lm -c main.cpp -o main.o
g++ -I . -lm -c Option.cpp -o Option.o
g++ -I . -lm -c Option_Price.cpp -o Option_Price.o
g++ -I . -lm -c StdNormalCDF.cpp -o StdNormalCDF.o
g++ -I . -lm -c test.cpp -o test.o
g++ -I . -lm -o main main.o Option.o Option_Price.o StdNormalCDF.o test.o
sprx7767@Pranav-ROG:/mnt/d/gatech/ISYE 6767 - Design and Implementation of Systems to Support Finance/hw-3/src$ ./main
Running unit tests...
All unit tests passed!
Enter option type (c for call, p for put): c
Enter strike price: 100
Enter current price of underlying: 80
Enter risk-free rate: 0.0423
Enter time to maturity: 0.50
Enter volatility: 0.20
Enter steps for binomial model: 1000
BSM Price: 18.3275
BSM Delta: -0.912785
Binomial Price: 18.3368
Binomial Delta: -0.912618
sprx7767@Pranav-ROG:/mnt/d/gatech/ISYE 6767 - Design and Implementation of Systems to Support Finance/hw-3/src$
```