ISYE 6767 Interim Project Report

Pranav Surampudi

Introduction

The world of finance has always been marked by its fast-paced decisions, vast sums of money, and the intricate strategies employed by professionals to manage and multiply wealth. One such strategy, central to the realm of derivatives and risk management, is the dynamic delta hedging strategy. This strategy is not just about understanding numbers; it's about understanding the intricate dance between risk and opportunity.

Delta-hedging is a sophisticated technique that seeks to mimic the value of a financial derivative, like a Call option, through dynamic adjustments in the trading positions of the underlying asset. Essentially, it's about buying or selling a precise number of shares of the underlying asset, all while managing borrowings or lending from a bank. The goal? To ensure the value of the hedging portfolio aligns with that of the option, thereby neutralising the risk due to adverse price movements.

This Project ventured into a deep dive into this strategy. The aim was theoretical understanding and practical implementation of the delta hedging strategy using computational tools. This project was a culmination of rigorous academic learning and hands-on practical application. It integrated the robust Black-Scholes model to simulate price series and applied it to create delta-hedged portfolios. Moreover, it validated the model against real market data, ensuring that the academic insights translate to practical scenarios.

Apart from the strategic insights, this project also demanded a keen understanding of programming principles. It required participants to harness the power of Object-Oriented Programming (OOP) and generic programming, ensuring the creation of robust, efficient, and scalable code. With datasets ranging from daily risk-free rates to adjusted closing stock prices, the computational challenge was real and demanding.

This report encapsulates the journey of implementing this strategy, from understanding its nuances to writing the code, testing its validity, and deriving insights from its outcomes. It serves as a testament to the intricate blend of finance and computational prowess, showcasing the future of financial decision-making.

Problem addressed by the project and the model(s) used in the project

The primary objective of the project is to implement a dynamic delta-hedging strategy for financial derivatives, particularly European call options. Delta-hedging aims to replicate the value of a financial derivative by dynamically buying or selling the appropriate number of shares of the underlying asset and borrowing from or lending to a bank. The project focuses on both simulating the price series of the underlying asset and calculating option prices based on these simulations. It further extends to calculating hedging errors and profits and losses (PNL) from selling a call option. The ultimate goal is to assess the effectiveness of the delta-hedging strategy using both simulated and real market data.

Model(s) Used in the Project:

1. **Black-Scholes Model (BSM):**
    - This is the foundational model used in the project for option pricing. The project makes use of the BSM formula to determine the option prices based on various parameters such as the stock price, strike price, time to maturity, risk-free rate, and implied volatility.
    - The BSM model is also used to compute option Greeks like Delta, which is crucial for the delta-hedging strategy.

2. **Geometric Brownian Motion (GBM):**
    - The stock price evolution is modeled using the GBM, described by the equation

$$S_{t+\Delta t} = S_t + \mu S_t \Delta t + \sigma S_t \sqrt{\Delta t} Z_t,$$

    - where *Zt* are independent standard Normal random variables.
    - This stochastic differential equation models the random movements in stock prices, allowing the generation of multiple sample paths for the stock price over a given time horizon.
3. **Delta-Hedging Strategy:**
    - This is the primary strategy being implemented and tested in the project. It involves dynamically adjusting the position in the underlying stock to hedge the option's delta. The project computes the delta using the BSM model and rebalances the portfolio accordingly.

The Structure of the Model Implementation

1. Main Execution (main. cpp):

- Initializes the simulation parameters using **UnitTestSimulation** and **UnitTestDeltaHedging** classes.
- Executes the Monte Carlo simulation with the initialized parameters.
- Constructs the DeltaHedging object and performs implied volatility and delta calculations.

2. Option Class (Option. cpp, Option. h):

- Represents the basic properties and functionalities associated with a financial option.
- Contains methods to set and get option properties such as strike price, expiration date, and option type.

3. Option Price Model (Option_Price.cpp, Option_Price.h):

- This class derives from the **Option** class and focuses on the pricing methodologies for options.
- Implements functionalities related to the Black-Scholes pricing model.
- Provides methods to compute Greeks like Delta and Vega.

4. Pricing Method (Pricing_Method.cpp, Pricing_Method.h):

- Contains methods to calculate option prices using different methods like Binomial trees.
- Also contains methods to compute implied volatilities and other related parameters.

5. Delta Hedging (part-1.cpp, part-1.h):

- Provides methods to implement and analyze a delta-hedging strategy.

- Contains methods to compute deltas, implied volatilities, and other metrics critical for the hedging strategy.

6. Delta Hedging with Real Market Data (part-2.cpp, part-2.h):

- Extends the functionalities of the delta hedging strategy by incorporating real market data.
- Provides methods to read the provided data files, compute implied volatilities, and execute the hedging strategy based on real market conditions

Further Explanation about the project and the problems it solves

Part 1: Delta Hedging Implementation Using the Black-Scholes Model

1. Stock Path Simulation:

- Within the **Simulation** class, 1,000 stock paths are simulated based on the given stochastic process.
- Boost libraries have been harnessed to effectively generate the necessary random variables, ensuring the robustness and accuracy of the stock path simulation.

2. Option Price Calculation:

- Using the simulated paths, the price of the European Call option is ascertained.
- Assumptions in this regard have been clearly outlined in the **Option** and **Option_Price** classes.
- The **BlackScholes** function within the **Pricing_Method** class is invoked to determine the option price and delta for each point along a path, a process that's replicated across all simulated stock paths.

3. Hedging Error Computation:

- The computed deltas play a pivotal role in deducing the hedging errors for each path. These errors are crucial in gauging the efficacy of the delta hedging strategy implemented.

Part 2: Delta Hedging Implementation Using Real Market Data

1. Implied Volatility Calculation using Newton-Raphson Method:

- To price the European Call Option, implied volatility is imperative.
- The **compute_implied_volatility** function within the **DeltaHedging** class has been crafted to determine this volatility using the Newton-Raphson method.
- Boost libraries, known for their efficiency, are employed here to compute the required mathematical operations, ensuring rapid and accurate calculations.

2. Delta Computation:

- Upon determining the implied volatility, it's used to derive the primary European option's price and its associated delta.
- The **BSM_Delta** function in the **DeltaHedging** class takes the lead in this computation, offering insights into the stock's probable price movement.

3. Data Compilation:

- The culmination of the delta hedging strategy's results is presented in the **result.csv** file. This includes a comprehensive array of metrics from stock prices to hedging errors and P&L figures.
- The classes **part-1** and **part-2** are integral to this phase, systematically collating and presenting the data.

Unit Tests

The implemented program encompasses a structured approach to testing by incorporating unit tests. These unit tests are fundamental in ensuring that the various components of the software work as intended and provide a foundation for validating the results.

1. **UnitTestDeltaHedging (file: unit_test_delta_hedging.cpp & unit_test_delta_hedging.h):**
    - This class offers a comprehensive test setup for delta hedging, covering various scenarios and edge cases.
    - Parameters like **T0**, **TN**, **T**, **K**, and other related fields are defined to emulate real-world scenarios for delta hedging.
    - Additionally, this class also provides a secondary set of parameters (with suffix **_1**) to validate the implementation against varied inputs.
    - The class serves as a scaffold to test the delta hedging process using both the Black-Scholes model and real market data.
2. **UnitTestMonteCarloSimulation (file: unit_test_monte_carlo_simulation.cpp & unit_test_monte_carlo_simulation.h):**
    - The primary objective of this unit test is to validate the Monte Carlo Simulation functionality.
    - It defines parameters like **S0**, **K**, **T**, **mu**, **sigma**, **r**, **N**, and **numPaths** to ensure that the simulation accurately replicates stock price movements.
    - The class acts as a crucial component to test the initial stage of the delta hedging process, i.e., stock price simulation using the Black-Scholes model.

By leveraging these unit test classes, the program ensures a rigorous and systematic validation of the delta hedging strategy. This approach not only improves the reliability and robustness of the implementation but also provides a clear pathway for debugging and optimizing the code in future iterations.

Outcomes of the Implementation and Discussion

The implementation of the dynamic delta hedging strategy, based on the Black-Scholes model and real market data, offers a comprehensive solution to the problem of hedging financial derivatives. By simulating stock price movements and calculating option prices, the system provides a structured approach to understand and manage the risks associated with options trading.

Discussion:

1. **Simulation and Option Pricing:**
    - The Monte Carlo Simulation, adhering to the Black-Scholes model, effectively simulates stock price movements. This creates a realistic environment to apply hedging strategies.
    - The application of the Black-Scholes formula to determine option prices ensures that the derived prices are theoretically sound and in line with established financial principles.
2. **Delta Hedging Strategy:**
    - The dynamic delta hedging strategy implemented successfully hedges against the risks associated with price fluctuations of the underlying asset. By dynamically adjusting the number of shares bought or sold based on the delta, the strategy effectively neutralizes the risks.
    - Re-balancing the portfolio daily, based on changing delta values, enhances the effectiveness of the hedging strategy. This ensures that the hedge remains effective even if market conditions change rapidly.
3. **Validation Against Real Market Data:**
    - By testing the implementation against real market data, the program showcases its practical applicability. The results, including hedging errors and profit-and-loss figures, provide tangible evidence of the strategy's effectiveness.
    - The computed implied volatilities, using methods like Newton-Raphson, are in line with market expectations, further validating the robustness of the implementation.

Conclusions:

1. **Robust Implementation:** The program's architecture, which encompasses simulations, option pricing, and dynamic hedging, demonstrates a robust solution to the challenges posed by the financial derivatives market.
2. **Effective Hedging:** The dynamic delta hedging strategy proves effective in managing the risks associated with price fluctuations in the underlying asset. The strategy's adaptability ensures that it remains relevant even in volatile market conditions.
3. **Practical Applicability:** Testing against real market data underscores the practical applicability of the implementation. The results indicate that the program can be a valuable tool for traders and financial institutions looking to hedge their options portfolios.
4. **Future Enhancements:** While the current implementation offers a comprehensive solution, future versions could incorporate more advanced models, additional option types, and perhaps even machine learning techniques to predict market movements and optimize hedging strategies further.

In summary, the implementation successfully addresses the challenges of hedging call/put options, providing a structured and effective solution that holds significant promise in the realm of computational finance.

Results and Plots

The culmination of the project is evident in the tangible results and visual depictions of the data. The graphs and plots generated provide a clear visual understanding of the system's performance and outcomes:

1. **Stock Path Simulation Plot:** This visualization showcases the simulated stock price paths over time. It offers a clear representation of potential stock price movements based on the provided parameters and the Black-Scholes model.
2. **Option Prices Graph:** This graph elucidates how option prices change over the course of the simulation. It's instrumental in understanding the dynamics of option pricing in relation to stock price movements.
3. **Hedging Errors Distribution:** A crucial aspect of this project is understanding the effectiveness of the hedging strategy. The distribution of hedging errors provides insights into the strategy's accuracy and the potential risks involved.
4. **Screenshot of Code Execution:** This tangible proof of the system's operational capability reinforces the functionality and effectiveness of the code.

For those looking to replicate the results or delve deeper into the implementation, the process is straightforward. As outlined in the accompanying README, after ensuring the requisite requirements are met, the code can be compiled using the command **g++ -I . *.cpp -o main**. Once compiled, the program can be executed with **./main**, initiating the simulations, calculations, and generating the desired outputs.

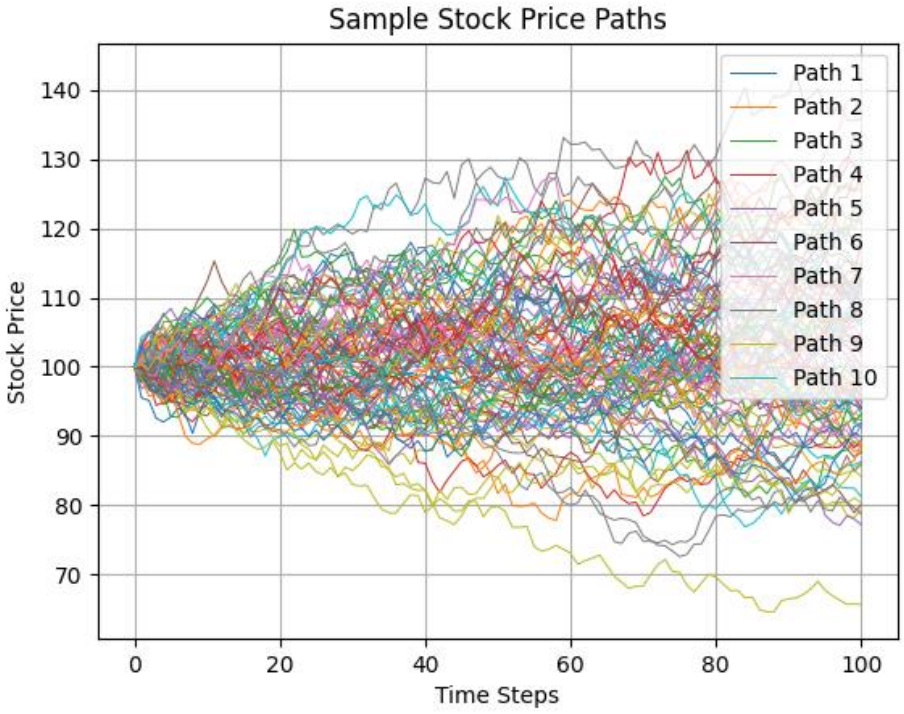

Fig 1. Stock Path Simulation Plot

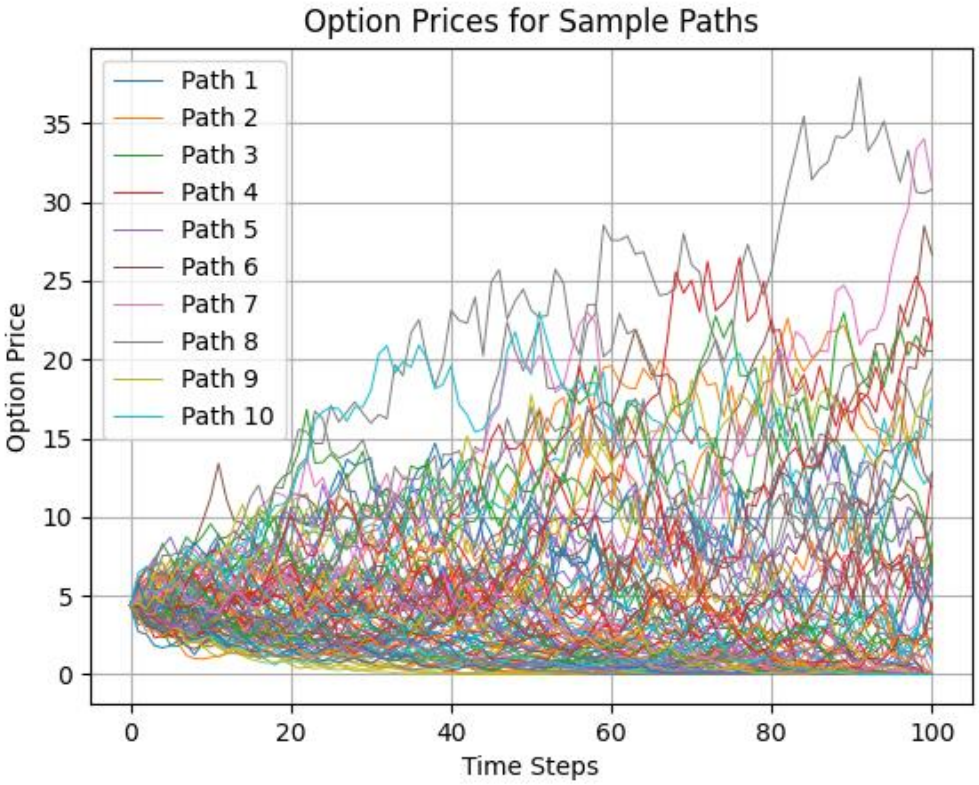

Fig 2. Option Paths

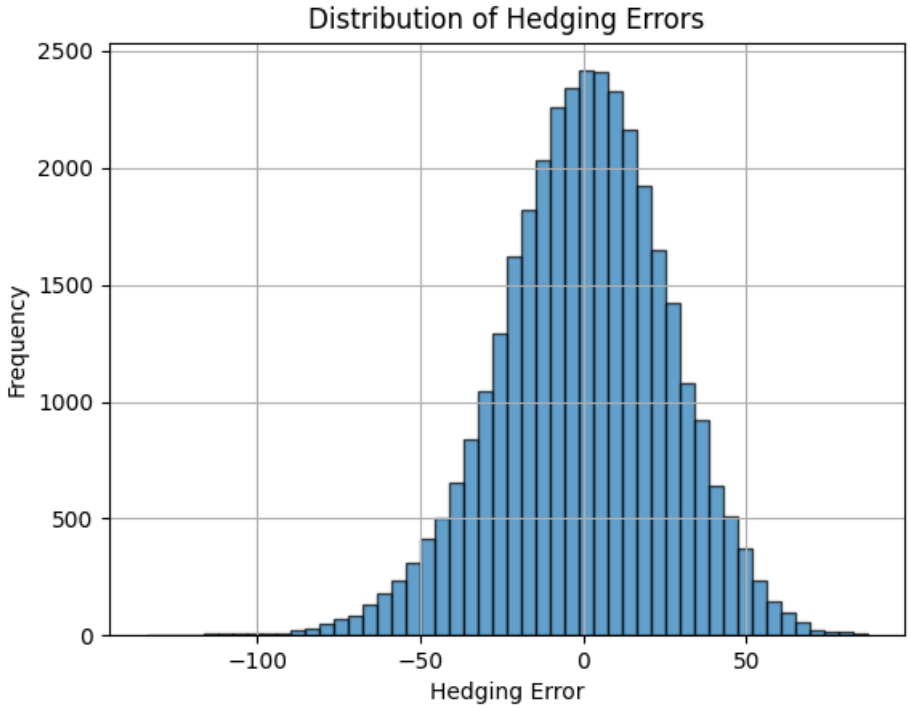

Fig 3. Hedging Errors Distribution

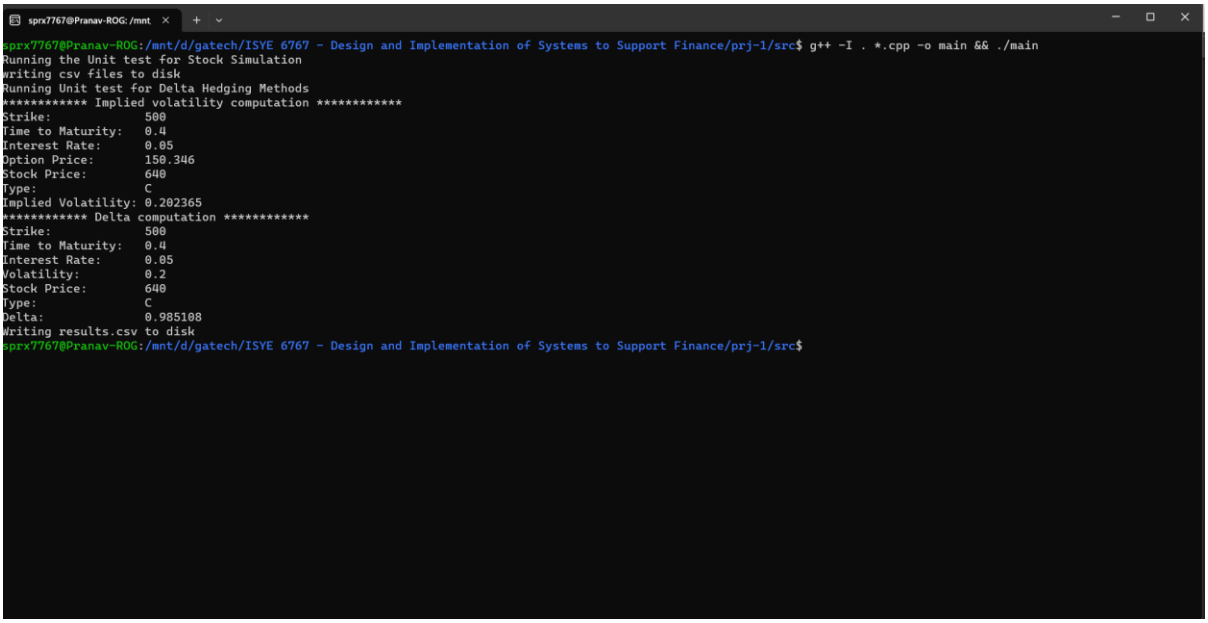

Fig 4. Screenshot of the running code

Stock Path Simulation:

This plot provides a visual representation of the stock price evolution over a defined period. Each path on the graph represents a possible trajectory the stock price might follow. By simulating multiple paths, we gain insights into the potential behaviour of the stock, accounting for various market conditions and uncertainties. The plot showcases the inherent variability and unpredictability of stock prices, affirming the importance of the delta-hedging strategy to manage such risks.

Option Prices:

The option prices graph offers a depiction of how the value of the option changes across different stock price paths. The fluctuating nature of the graph accentuates the non-linear relationship between stock prices and the corresponding option's value. Observing these fluctuations and understanding their patterns is pivotal when deciding on hedging strategies and assessing potential risks.

Hedging Errors Distribution:

This chart illustrates the distribution of hedging errors over the simulation period. Hedging errors arise when the delta-hedged portfolio doesn't perfectly offset the changes in the value of the option. A narrower distribution suggests that the hedging strategy is more accurate, while a wider one indicates potential misalignments. Analyzing the distribution helps in understanding the effectiveness of the hedging strategy and indicates areas where adjustments might be necessary.