ISYE 6767 INTERIM PROJECT

IMPLEMENTATION OF DELTA HEDGING STRATEGY USING C++

Submitted by
Haripriya Nair
GT ID: 903452244

NOTE: I would like to exercise my extension option for this project.

1. Problem Addressed by the Project:

The project aims to implement a Delta-Hedging Strategy which hedges the risk imposed by the inherent volatility of a call option and the price movements of the underlying asset. The principle of the strategy is that a change in the option price is caused by the change in underlying asset price. The solution to this problem would be to offset the risk of the option into long and short positions. For example, a long call option would be delta hedged by shorting the underlying stock and saving money in a bank account. This is achieved by replicating the value of the call option trading on the stock into a portfolio of stocks and bank savings.

The fundamental objective of the project is to calculate the delta (number of stocks to long/short) and the hedging error associated with this position which should be approximately zero. Therefore, the project is split into two parts. The first part aims to simulate an underlying asset and associated call option price series to test out the strategy. Deltas and Hedging errors for these simulated assets are evaluated and reported. The second part of the project takes in real market data to come up with delta-hedge positions and test out scenarios.

2. Model used in the Project:

The delta hedging strategy is implemented using replicating portfolio method. For the first part of the project, stock price series is simulated using the geometric Brownian motion given by the following equation:

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

where are independent random variables following standard normal distribution, which are computed using the C++ boost random number generator and normal distribution functionalities. The call option price series on this underlying stock is calculated using the Black-Scholes pricing model mentioned below.

$$C = SN(d_1) - N(d_2)Ke^{-rt}$$

$$d_1 = \frac{\ln\left(S/K\right) + \left(r + s^2/2\right)t}{s \cdot \sqrt{t}}$$

$$d_2 = d_1 - s \cdot \sqrt{t}$$

C = Call premium
S = Current stock price
t = Time until option exercise
K = Option striking price
r = Risk-free interest rate
N = Cumulative standard normal distribution
e = Exponential term

s = St. Deviation
ln = Natural Log

Using the option prices, the corresponding deltas are calculated using the equation N(d1) where N and d1 are from the Black Scholes computation. Following this step, we compute the residual cash (B) as below:

$$B_0 = C_0 - \Delta_0 S_0$$

$$B_i = \Delta_{i-1} S_i + B_{i-1} e^{r_{i-1} dt} - \Delta_i S_i$$

where $i \geq 1$, Δ_I is delta, S_i is the stock price and C_0 is the initial call option price.

After creating the replicating portfolio of deltas and residual cash, the hedging error for this position is calculated as below:

$$HE_i = \Delta_{i-1} S_i + B_{i-1} e^{r_{i-1} dt} - V_i$$

where r_i is the rate for that day and dt is time interval of that period.

The hedging errors are then reported and verified to be centred around zero.

The second part is then implemented taking in real market data. In this case, we have taken Google stock prices as our underlying asset for the year 2011, the corresponding risk free rates and the option prices data on the stock. To apply the delta-hedging strategy on this data, we need to compute implied volatility on the option. I have used the Newton Raphson method to arrive at that using sigma as 0.24 to be my initial guess. The idea here is to use the analytic derivative i.e. the vega to make a linear estimate of where the solution i.e. the implied volatility should occur, which is more accurate than mid-point approach taken by interval bisection. Once the implied volatilities are computed, the approach in first part is taken to arrive at the delta hedge positions. For each position, the total wealth is computed if we were to sell the call without any hedge. Additionally, an output file is generated comprising of the stock price, option price, implied volatility, delta, hedging error, PNL and PNL with hedge.

3. Structure of Model Implementation:

The Model has three components and a main program implementing the components discussed in detail below:

a. **Underlying Class:** This class represents the price series for the underlying asset i.e. the stock in our case. It has static constant variables which are private to the class and has values of uptick(u), volatility(sigma) and time interval duration(delta_t). These private variables are used to create a stock price series object with 100 prices as a default case, using constructor.
It has three member functions:
- Underlying(): The default constructor. This method is used to create a default Stock price series object with 100 stock prices computed using the boost random number generator and standard normal distribution function.
- Underlying(std::vector<double> P): The parameterised constructor. This method creates a stock price series object with stock prices initialised to the input stock prices.
- double operator()(int i) const: This is an operator overloading function over (). Therefore, an object of Underlying class, say S, would return individual stock prices for S(i) where i is an integer.

b. **Option Class:** This class represents the price series for option price on the Underlying object. The private static variables are used to compute the Black Scholes options price, corresponding deltas and in case of given option prices, the vegas, deltas and implied volatility.
Member functions:
- Option(Underlying S): The default constructor computes call option values over given Underlying object using the Black Scholes equation. Additionally, it also computes deltas for the replicating portfolio which is N(d1) and stores it in member variable delta.
- Option(std::vector<double> F, Underlying S): The parameterized constructor which sets the Option prices to input vector F and computes corresponding deltas similar to above.
- double operator()(int i) const: This is an operator overloading function over (). Therefore, an object of Option class, say C, would return individual call option price for C(i) corresponding to S(i) where i is an integer.
- void calculate_PNL(int n): Calculates the PNL value for the Call option without hedge.
- void calculate_Wealth(int n, std::vector<double> rate): Calculates the total wealth if we sell the call option without hedge.
- void calculate_delta(Underlying S, int n, double K, std::vector<double> rate): Calculates deltas for the Underlying object to replicate the call option.

There are three private methods which are used to compute intermediate values used for above calculations:
- double calculate_vega(double s, double r, double sig, double tau, double K): Calculates vega to arrive at the implied volatility using the Newton-Raphson method of approximation.
- double calculate_implied_volatility(double guess, double C_m, double s, double N, double r, int i, double tau, double K): Computes implied volatility

using Newton-Raphson methodology where the function is Black Scoles pricing equation and derived function is the vega = S*Delta*sqrt(tau) where tau = time to maturity and delta = N(d1).

- double calculate_option_price(double s, int i), double calculate_option_price(double S, int i, double sig, double N, double d_t) : These are overloaded functions, with former computing simulated option prices and later being used for intermediate calculation of option price for computing implied volatility.
- double calculate_d1(double s, double K, double r, double sig, double tau): This is an operator overloading function over (). Therefore, an object of Option class, say C, would return individual call option price for C(i) where i is an integer.

Public variables of Option class available for user access: PNL, delta, wealth, implied_volatility. All of these are vectors of type double.

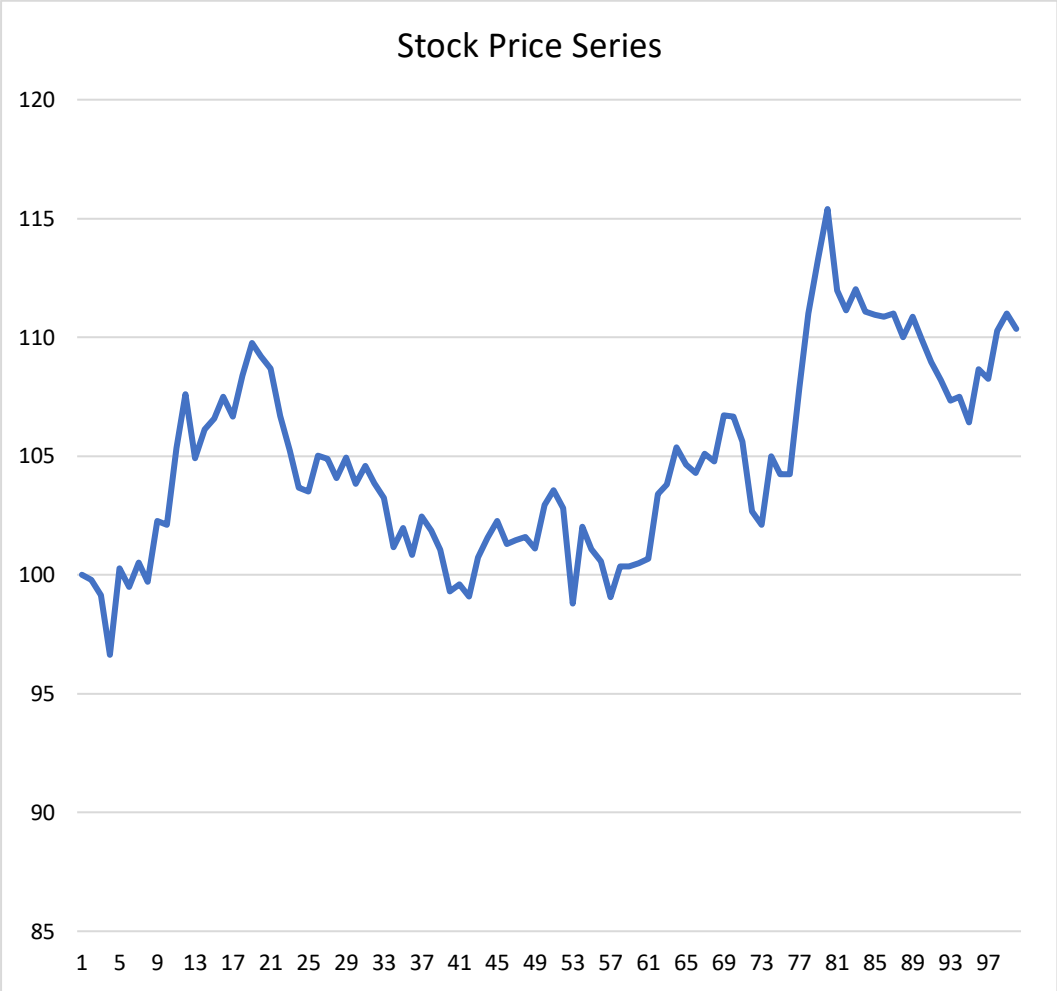
c. **Hedging Class:** This class represents the hedging error series for delta hedging the Options object. It comprises of a private risk free rate, r and delta time duration for the simulation case.
Member functions:
   - Hedging() : Not implemented.
   - void calculate_hedging_error(Underlying S, Option C), void calculate_hedging_error(Underlying S, Option C, std::vector<double> r, double d_t): Calculates hedging errors and residual cash for simulation (former) and real market data(latter).
   - void PNL_with_hedge(std::vector<double> r): Computes the Profit and Loss if the user sells the call option with hedging.
   - void calculate_residual_cash(): Computes the residual cash user should borrow or deposit from/in a bank for delta hedging the option.
   - double operator()(int i) const: This is an operator overloading function over (). Therefore, an object of Hedging class, say H, would return individual hedging error for H(i) where i is an integer.

Public variables of Hedging class available for user access: PNL_WITH_H and Cash, both of which are vectors of type double.

4. Unit Test Cases

Underlying:

The stock price series follow Brownian motion.



Option:

Call option:
Scenario: Out of money where K = 1000, r=0.025, sigma = 0.24, T = 10 days
Expected output: C = 0
Actual output: C = 0

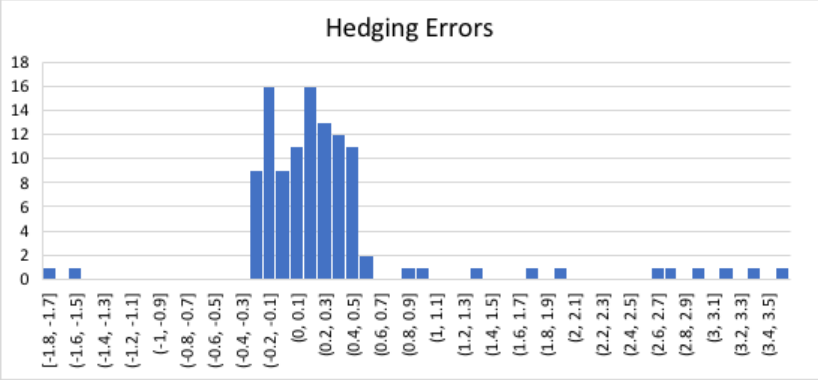
Delta:
Scenario: Out of money where K = 1000, r=0.025, sigma = 0.24, T = 10 days
Expected output: C.delta = 0
Actual output: C.delta = 0

Implied Volatility:
Scenario: S=100, K=105, r=0.025, C=, T=0.4
Expected output: C.implied_volatility[0] = 0.24

Actual output: C.implied_volatility[0] = 0.24

Hedging:

Plot of Hedging errors: Centered around zero.



5. Outcome of the Implementation:

The implementation generated stock price series, call option price series and corresponding delta hedging portfolios for part one of the project. The hedging error was mostly centered around zero as seen in the unit test cases. The part 2 was implemented with real market data of Google stock, its option prices and risk free rates for the corresponding period. The delta hedge portfolios for this data has close to zero error taking into account system noise and technical constraints. The returns over the delta hedged portfolios are definitely higher as compared to without hedging option returns. This can be seen through the PNL values output file generated in part two of the implementation.

The drawback to the strategy is the necessity to constantly watch over price fluctuations and adjusting positions in real time. As seen, the positions keep altering on a daily basis which implies that the user needs frequently buy or sell securities to avoid being under or overhedged. The ideal case to apply this strategy for trading is when the trader expects strong price movements in the underlying asset.

The scope of the project can be extended to user interaction for future. The model can be extended to include options for user to input real market data files or individual stock and option prices for the day and come up with delta hedging portfolio.

6. References:

1. https://www.quantstart.com/articles/Implied-Volatility-in-C-using-Template-Functions-and-Newton-Raphson
2. https://www.codesynthesis.com/pipermail/odb-users/2012-May/000561.html
3. https://quant.stackexchange.com/questions/7761/how-can-the-implied-volatility-be-calculated
4. https://www.investopedia.com/terms/d/deltahedging.asp
5. https://quant.stackexchange.com/questions/39010/interpertation-of-delta-hedge-error-in-black-scholes
6. https://www.investopedia.com/university/options-pricing/black-scholes-model.asp