Project Report

Implementation of a Dynamic Delta Hedging Strategy

Submitted in partial fulfilment of the requirements of the class of

ISyE 6767 Sys-Computation Finance

Submitted By
Abhinav Agrawal
GT Id: 903618080

Objective:

The objective of the project is to test Delta Hedging deployed using Black-Scholes-Merton model for an option (Call/Put).

The project has 2 parts:

Part 1 of the project deals with simulating paths of the stock over given number of paths (n_paths = 1000) and timesteps (n = 100). This part keeps the interest rate and the volatility constant

Part 2 of the project deals with calculation of implied volatility using marked observed prices of the option and calculation of pnl and hedging error.

The implementation of the project is done in C++ keeping in mind the standard implementation rules used in the industry.

Delta Hedging Strategy and Black Scholes Model:

Delta-hedging is a hedging strategy that aims to replicate the value of a financial derivative, such as a Call option, written on a traded asset through dynamically buying (or selling) a proper number of shares of the underlying asset and borrowing from (or lending to) a bank.
Delta is defined as change in value of the option to the change in the underlying.

$$\Delta = \frac{\partial V}{\partial S}$$

In case of Black Scholes, delta of a call option is N(d1) whereas for a put option it is 1-N(d1).

Black Scholes is a model used for pricing derivatives.
In case of an option on non-dividend paying stock, the price of the call and put be calculated as:

$$\begin{aligned} P(S_t, t) &= Ke^{-r(T-t)} - S_t + C(S_t, t) \\ &= N(-d_2)Ke^{-r(T-t)} - N(-d_1)S_t \end{aligned}$$

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-z^2/2}\, dz.$$

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

$$\text{where } d_1 = \frac{\ln \frac{S_t}{K} + (r + \frac{\sigma^2}{2})t}{\sigma\sqrt{t}}$$

$$\text{and } d_2 = d_1 - \sigma\sqrt{t}$$

C: Call option Price
N: CDF of Normal Distribution
S(t): Spot Price of an Asset
K: Strike Price
r : risk free interest rate
t: time to maturity
σ = Volatility

Hedging Error:

The concept of hedging error arises from the rebalancing frequency. The delta hedging assumes a continuous rebalancing of the portfolio to be delta neutral. Since it is not practically possible to rebalance continuously, we get hedging error which keeps on reducing as we increase our rebalancing frequency.

Another reason of hedging error could be the rate of change of delta. For a higher rate, we will have a higher hedging error as rebalancing would be difficult.

Hedging error can be calculated as:

$$HE(i) = \delta(i-1) * S(i) + B(i-1) * e^{r(i-1)\Delta t} * V(i)$$

$$B(i) = \delta(i-1) * S(i) + B(i-1) * e^{r(i-1)\Delta t} - \delta(i) * S(i)$$

$$B_0 = V_0 - \delta_0 S_0.$$

S(i): Stock Price
V(i): Option Price
r(i): risk-free rate

Implementation:

Part 1:

Method:

1) Simulation of Stock Price path as a geometric Brownian motion using the following formula:

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

$$\Delta t = T/N$$

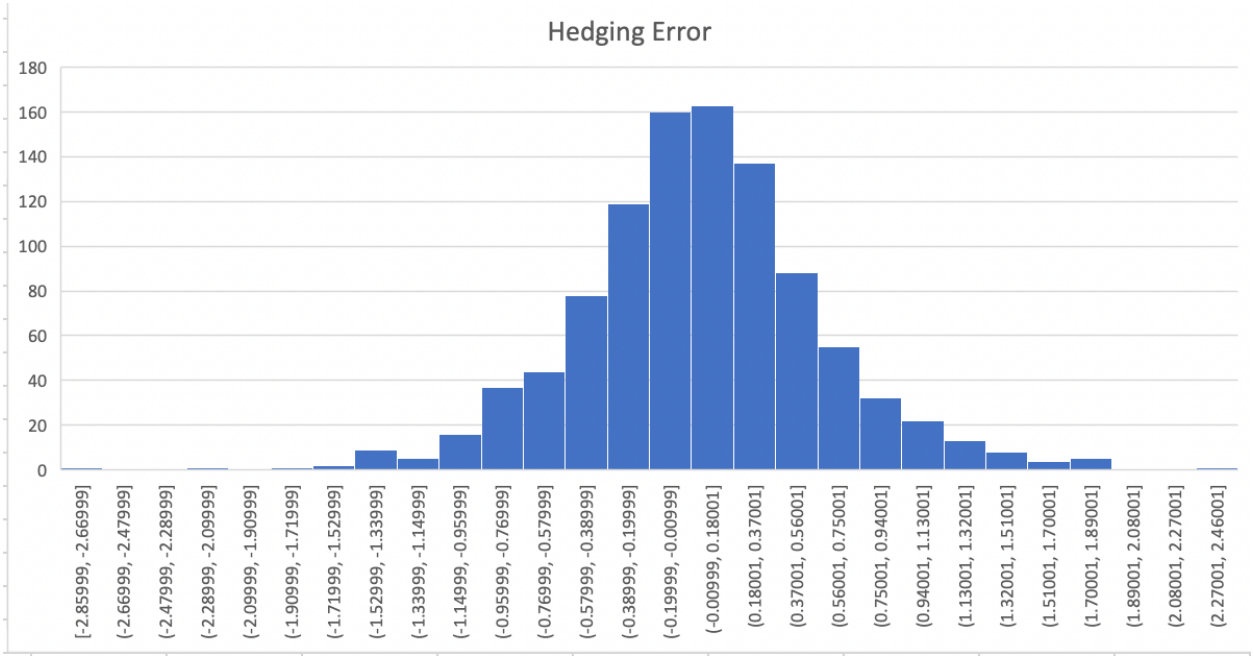
2) Parameters used for the above simulation are:

$$S0 = 100,\ T = 0.4,\ \mu = 0.05,\ \sigma = 0.24,\ r = 0.025,\ N = 100$$

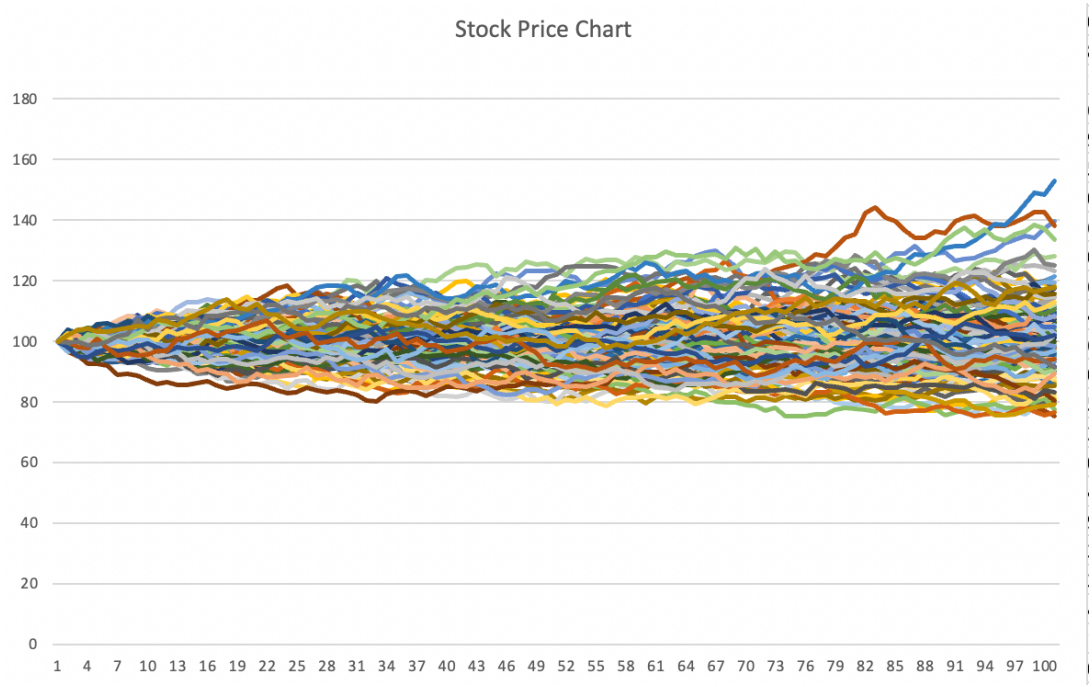
3) The price and delta of European Call option with strike (K = 105) and T = 0.4 is calculated using black Scholes formula. Delta_t is calculated as mentioned in the formula above.
4) In addition to this, calculate the cumulative hedging error for each path and plot it
5) The code is run for 1000 paths to accommodate for randomness of the simulation.
6) Plot the cumulative hedging error at expiration for all paths.

Results:

Hedging Error at N:



Stock Price Path (100 paths):



Option Price Path (100 Paths):

NOTE: The output file is attached for part 1 in the submission.

a) HedgingError.csv
b) stock_path.csv
c) option_price.csv

Inference:

1) Cumulative hedging error accumulates with time. It follows a normal distribution with mean 0
2) Payoff of the option is 0 for some cases when k>s(t). The price of option at t=0 is 4.3916
3) Stocks follow as random path which follow a geometric Brownian motion

Code Walkthrough (Part 1):

The implementation consists of following parts:

Option.h: Header file for option related date:

- Strike_price (double)
- isCall (bool)
- volatility (double)

It also consists of getters for the private member variables

Option.cpp: Consists of the constructor and destructor of the option class. Also consists of a parameterized constructor for initializing values to the variables

Stock.h: Header file for stock related date:

- spot_price (double)
- mu (double)
- interest_rate (double)

It also consists of getters and setters for the private member variables

Stock.cpp: Consists of the constructor and destructor of the option class. Also consists of a parameterized constructor for initializing values to the variables

Simulated_Price.h: This file is the header file for the simulation of the price. It consists of a call to price simulate function that is performing the simulation for task 1 and creating the stock_path, option_path and hedging_error.

Simulated_Price.cpp: This consists of default constructor and destructor to the Simulated_Price class. In addition to this, it has a function name: price_simulate:

- **price_simulate ():** This uses the functionality of random number generator which generates normally distributed random numbers (mean 0 and variance 1).

The function computes 1000 simulated stock paths for 100 timesteps in each path. For each of the path, option price and hedging error is calculated.

Cumulative hedging error is the hedging error at N.

The function writes the output to 3 files:

d) Hedging_Error.csv
e) Simulated_Stock_Path.csv
f) Calculated_Option_Price.csv

- The price_simulate function uses BSM_pricer function which has the implementation of Black Scholes Pricing like assignment 3.
  - It returns the price and delta of the option which is being used by the price_simulate function to price the option for each simulated path.

Option_Price.h: It contains the implementation of black Scholes pricing model which returns the price and delta of an option.

Option_Price.cpp: This file contains the implementation of Black Scholes Pricing model which returns the price as well as delta of the option.

Part 2:

The problem asks to compute the implied volatility, hedging error, PnL with hedge, delta and pnl.
In most practical scenarios, instead of using the historical volatility, we used black Scholes model to compute the implied volatility of the option using the market observed price of the option. This implied volatility is then used to computing the Greeks of the option.

There are numerous methods that can help in computing the implied volatility (root finding problem). I have used Bisection method (Binary Search method) for solving the root finding problem and computing the implied volatility.

Bisection Method:

Also known as Binary Search Method, this method is used to find roots of a polynomial equation. If a function is continuous in [a,b] and f(a).f(b) <0, then there exists a solution such that f(x) = 0.

In the implementation, I have used initial values of a and b 0.00001 and 1. The function takes the difference between the observed price of the option and calculated price from BS model and till this price difference is greater than a tolerance, it will go on search for a root.
The value of a and b is updated for condition depending on price being greater than observed price or lesser than that and mid_value is updated as (a+b)/2. The function converges when it finds a value of x for which it can return a price that is within the acceptable tolerance limit.

Method:

1) We are given 3 files with interest_rate, security_price and option_price data. This file is read and converted to desired format and then volatility is calculated for each interest_rate, time_to_maturity, option_price and strike for the date ranges given.
2) Market observed price is used to calculate the implied volatility which is used in the Black Scholes formula for calculating the delta of the option. I have used Bisection Method (Binary Search method) to get the implied volatility.

3) The last part of the code calculates the Profit & Loss for two scenarios:

    a. Portfolio of only call option
    b. PNL with cumulative hedging error; rebalancing frequency (Δt) = 1/252

4) **The total wealth is equal to the pnl of the option** (as seen in the PNL) column.

Results:

Date	Stock Price	Option Price	Implied Volatility	Delta	Hedging Error	PNL	PNL with hedge
2011-07-05	532.44	44.2	0.264557	0.722576	0	0	0
2011-07-06	535.36	46.9	0.273632	0.733253	-0.590078	-2.7	-0.590078
2011-07-07	546.6	55.3	0.273395	0.787512	-0.158233	-11.1	-0.748311
2011-07-08	531.99	43.95	0.271265	0.719361	-0.155556	0.25	-0.903867
2011-07-11	527.28	41	0.281972	0.691443	-0.438191	3.2	-1.34206
2011-07-12	534.01	46.4	0.292194	0.722705	-0.74659	-2.2	-2.08865
2011-07-13	538.26	49.3	0.29166	0.745007	0.171495	-5.1	-1.91715
2011-07-14	528.94	41.15	0.275592	0.706868	1.20653	3.05	-0.710619
2011-07-15	597.62	99.65	0.284149	0.940693	-9.95232	-55.45	-10.6629
2011-07-18	594.94	97.65	0.307212	0.926341	-0.521057	-53.45	-11.184
2011-07-19	602.55	103.8	0.271484	0.960156	0.899456	-59.6	-10.2845
2011-07-20	595.35	97.8	0.304466	0.931833	-0.913125	-53.6	-11.1977
2011-07-21	606.99	108.15	0.279129	0.964161	0.496542	-63.95	-10.7011
2011-07-22	618.23	118.7	0.25074	0.985911	0.287173	-74.5	-10.4139
2011-07-25	618.98	119.95	0.301368	0.971417	-0.510567	-75.75	-10.9245
2011-07-26	622.52	123.25	0.292915	0.978417	0.138817	-79.05	-10.7857
2011-07-27	607.22	108.65	0.309638	0.957588	-0.369779	-64.45	-11.1555
2011-07-28	610.94	112.1	0.306205	0.964895	0.112227	-67.9	-11.0433
2011-07-29	603.69	106.8	0.37266	0.924679	-1.69549	-62.6	-12.7387

Inference:

1) Delta of the option increases as we go near to the time to maturity. This is because the delta for ITM option is 1 at maturity and hovers around 1 as we get close to maturity.

2) The delta hedged portfolio has a lower drawdown as compared to naked position in the option. This could limit the upside potential as well.

3) As we see from the results, the implied volatility of the option increases as the price of the stock increases. This is in line with what we have studied for in the money option

Conclusion:

Talking about the strategy in general, achieving a completely delta neutral portfolio is not practical due to continuous price movement and change in the delta of the option in continuous time. Hence it can keep on accumulating some hedging error. Also, the change in delta could

also lead to a difficulty in getting a completely delta neutral portfolio. However, if not exactly but almost delta neutral is maintained, we can limit the downside risk.
Although practically a complete delta neutrality is not possible because of the behavior of delta.

Unit Test (For the project):

Unit test for the code check the price, delta and volatility of a given set of input and the price from the code is compared with the price calculated using BS price (used online calculator) and is checked with a given level of tolerance.

The level of tolerance for the delta and price is 0.2 and for implied volatility is 0.05.

Test Results:

```
/Users/global_minima/Desktop/GeorgiaTech/Fall22/Subjects/SysComp/Mid_Term/cmake-build-debug/Mid_Term
Results of Unit Test Case
Black Scholes Option Price function returns correct value
Black Scholes Option Delta function returns correct value
Implied Volatility function returns correct value
```

Code (Unit Test):

Unit Test (Price):

```
void BS_Price_Test() {

    Option_Price op;
    double underlying_price = 630;
    double strike_price = 600;
    double interest_rate = 0.002;
    double time_to_maturity = 0.0822;
    double sigma = 0.3;

    double actual_bs_price = 39.45;

    double calculated_bs_price = op.BSM_Price_iv( spot: underlying_price,  strike: strike_price,  isCall: true,interest_rate, volatilty: sig

    if (abs( lcpp_x: calculated_bs_price - actual_bs_price) < 0.2) {
        cout << "Black Scholes Option Price function returns correct value" << endl;
    } else {
        cout << "Black Scholes Option Price function does not return correct value" << " " << calculated_bs_price << endl;
    }

}
```

Unit Test (Delta):

```cpp
void BS_Delta_Test() {

    Option_Price op;
    double underlying_price = 630;
    double strike_price = 600;
    double interest_rate = 0.002;
    double time_to_maturity = 0.0822;
    double sigma = 0.3;
    double actual_bs_delta = 0.729;
    double calculated_bs_delta = op.BSM_Price_iv( spot: underlying_price,  strike: strike_price,  isCall: true,interest_rate, volatilty: sigma

    if (abs( lcpp_x: calculated_bs_delta - actual_bs_delta) < 0.2) {
        cout << "Black Scholes Option Delta function returns correct value" << endl;
    } else {
        cout << "Black Scholes Option Delta function does not return correct value" << " " << calculated_bs_delta << endl;
    }
}
```

Unit Test (Volatility):

```cpp
void IV_Test() {
    BSM_Part2 pl;
    double option_target_price = 40;
    double underlying_price = 630;
    double strike_price = 600;
    double bank_rate = 0.002;
    double time_to_maturity = 0.0822;

    double vol_real_ans = 0.3089;
    double our_ans = pl.implied_volatility( price_from_file: option_target_price,  interest_rate: bank_rate,  spot: underlying_price, strike: st

    if (abs( lcpp_x: our_ans - vol_real_ans) < 0.05) {
        cout << "Implied Volatility function returns correct value" << endl;
    } else {
        cout << "Implied Volatility function does not return correct value" << endl;
    }

}
```

Code Walkthrough (Part 2 and Unit Test):

The code for part 2 is split across:

a) BSM_Part2.cpp
b) Option_Price.cpp
c) main.cpp

BSM_Part2:

- **BSM_part2.h:** The header file has default constructor and destructor alongwith definition of 2 functions namely implied_volatility and portfolioPnL.
- **BSM_part2.**cpp:
    - Contains the constructor and destructor and implementation of the mentioned functions:
    - Implied Volatility return the volatility which is being used in the blackscholes pricer to calculate the delta of the option and pnl and hedging error etc going forward
    - portfolioPnL : This is the function with void return type and is responsible for printing the result to the result.csv file. It computes the implied volatility for all the option_prices, interest_rate, time_to_maturity, stock_price and given strike price. The formula used is same as what we have in part 1.
    - The function takes the following input parameters:
        - Option_matrix (vector<double>) – Contains the computed price of option for the given date range
        - Interest_rate (vector<double>) – Contains the interest_rate as computed from the interest_rate file. The values are divided by 100
        - Stock_price (vector <double>) – Contains the stock price for date range from the file
        - Strike_price (double): Strike price as mentioned in question (K =500)
        - ir (map<string, double>) – gives the date and then all is written in the result.csv file

- **<u>Option_price.h and Option_price.cpp</u>**
    - The implementation has implementation of black Scholes pricer which returns the price and delta of the option.
    - The function depends on normalCDF function which returns the CDF of a function.

- **main.cpp**
    - This is the driving file of the program. The file has following functions:
        - **timeToMaturity**: vector<double> - Returns a vector of the difference of the expiration date and the input date in years
        - **map_to_vector:** vector<double> - Converts a map type to vector
        - **dateDifference :** double – returns the date difference between 2 dates in years
        - **file_read_option :** map<string, double> - returns the mapping of date alongwith the price from the options file using input date range
        - **ratesFromDate :** map<string, double> - returns the mapping of rates and dates from the file using input date range

        - **priceFromDate:** map<string, double> - returns the mapping of price and dates from the file using input date range
        - The map<string, double> above is converted to vector<double> using the map_to_vector function and is passed through the portfolioPnL function of the BSM_Part2 class using its object bsmPart2.

- **Unit test Implementation:**
    - The unit test has 2 files:
        - Unit_Test.h and Unit_Test.cpp
            - The Unit_Test.h has the definition of runtest() function that is created in the Unit_test.cpp
            - Unit_Test.cpp:
                - BS_Price_Test: Test for the price of the option using the calculated price from the program and the price calculated using online calculator
                - BS_Delta_Test: Test for the delta of the option (at t=0) using the calculated delta from the program and the delta calculated using online calculator
                - IV_Test: Test for the volatility of the option using the calculated volatility from the program and the volatility calculated using online calculator

The result of all the test function is Acceptance or Not Acceptance on the basic of tolerance as defined above in the Unit_Test section.

Test Output:

```
/Users/global_minima/Desktop/GeorgiaTech/Fall22/Subjects/SysComp/Mid_Term/cmake-build-debug/Mid_Term
Results of Unit Test Case
Black Scholes Option Price function returns correct value
Black Scholes Option Delta function returns correct value
Implied Volatility function returns correct value
```

Final Output:

```
/Users/global_minima/Desktop/GeorgiaTech/Fall22/Subjects/SysComp/Mid_Term/cmake-build-debug/Mid_Term
Results of Unit Test Case
Black Scholes Option Price function returns correct value
Black Scholes Option Delta function returns correct value
Implied Volatility function returns correct value
Simulation of Part 1 started!
Simulation of Part 1 ended!
Files have been created!
Starting Part 2
Part 2 finished
Result.csv file created

Process finished with exit code 0
```