INTERIM REPORT – DANIEL FILIPPI

The binomial method for option pricing.

Content:

Introduction – What is an option?

Motivation

An option is a type of derivative, which is a bet on the performance of an asset. Or according to Wikipedia: *"In finance, a derivative is a contract that derives its value from the performance of an underlying entity. This underlying entity can be an asset, index, or interest rate..."[1]*. A call option gives the holder the right, but not the obligation, to purchase from the option writer an asset for a set price (strike price) at a set time in the future.

Let us go through an example with context:

Tesla shares are worth approximately £200 at the moment.

Alice believes these shares will increase in the next 6 months.

Alice goes to an options broker.

The broker sells Alice a contract that says in simple terms:

"By the end of this contract, you will have the right to buy these shares at a specified "strike price" – let's say £220"

This is called a "Call Option." More specifically this type of strategy is called a "Long Call." As she is buying the contract.

By the end of the timeframe Tesla shares are now worth £240. But the broker must respect the contract and sell them to Alice for the strike price! Assuming Alice exercises the contract and proceeds to sell instantly, she is essentially pocketing the difference between the stock value and the strike price here. If the share value goes below £200? No problem! The contract need not be exercised. Alice will not be overpaying for any shares.

It is also worth noting that these contracts involve 100 shares of the underlying stock.

Of course, when modelling the potential wins/losses for both sides here, we see that Alice has a potential for infinite returns and no loss.

While the broker only has the potential for an unlimited loss.

So how does the broker make money in this scenario? They require some sort of compensation, this is the "value" or the "premium" of an option.

This is the problem I will be solving in my project. I will be calculating and visualising the value of options for a desired asset at various contract lengths, using the binomial method.

Aims

My **primary** aim here, to develop an application that provides a graphical interface for the user to evaluate fair put/call option prices. The user should be able to pick any share, and a strike price and

possibly an expiration date. The program will then automatically retrieve the needed data, e.g., Volatility, and proceed to show the user a line graph, the line representing the value of such option at a certain time (X axis).

My **secondary** aim is to create a tool that uses this calculated data. I would like to make a strategy builder where a user can build their position, consisting of various contracts. The builder will then plot out a payout chart of this position. If the strategy is known, it will recognise and name it. It'll also outline the max return/loss, net debit/credit and also margin if necessary.

The UI diagrams will clarify these ideas a lot.

In order to achieve these aims, I will need to do a few things.

Objectives

1) Research heavily on the principles **behind** options.
    a. Arbitrage.
    b. Payout plots.
    c. Volatility.
2) Grasp an understanding of options, how they are (fairly) priced and what variables may influence the price of a contract.
3) Decide on an appropriate framework which can handle UI and the calculation.
4) Decide on an API from which to retrieve the needed data from.
    a. Stock value (Open)
    b. Annual Volatility
    c. Interest (Potentially)
    d. Dividend per share (Potentially)
    e. Time to dividends (Potentially)
5) Design an interface for user data entry and the visualisation of the result.
    a. Lo-Fi designs
    b. Selecting a UI Framework
6) Design code for the calculation of options.
    a. Call options.
    b. Put options.
    c. Short/Long?
    d. American or European, or both?
7) Implement this code and test to see if results are valid.
    a. Compare results to existing and respected calculators.
8) Implement the interface. (For the primary aim)
9) Bind front and backend, making sure data is and visualised properly and accurately.
10) Research on the most known/used option strategies.
11) Build code for secondary aim. Just one contract.
    a. User must be able to build a contract.
    b. Contract must translate onto graph.
    c. Display other valuable information.
12) Implement a slider to dynamically change strike price for each option, also reflected on chart.

13) Improve on just one contract, modify code so that >1 contract can be plotted on the same chart. Allowing users to build more complex strategies.
    a. Recognise user-created strategies.
    b. Dynamically changing payout plot
14) Refinements

Literature Review / General Research

The main book I used to learn about options was written by Desmond J. Higham, called *"FINANCE, An Introduction to Financial Options Valuation"*[2]. It provided me with a great knowledgebase for me to tackle this task. Starting with the key financial principles behind options and slowly leading me to a more complete understanding. It explained how each type of option works and also gave algorithms to calculate their premium. Without this book I would be totally clueless. It has almost completely formed my understanding of options. Moreover, it also taught me how to plot payout charts and how to derive the payout charts for positions consisting of multiple options. Hence the idea for my second aim.

I also spent an extended amount of time on the web looking for Option Pricing calculators, I messed around with the different variables to learn how they affected the final value.

Here is my understanding of options:

There are 2 fundamental types of option. A "call" and a "put". A call is a bet on the **increase** of an asset, while a put is a bet on the **decrease**.

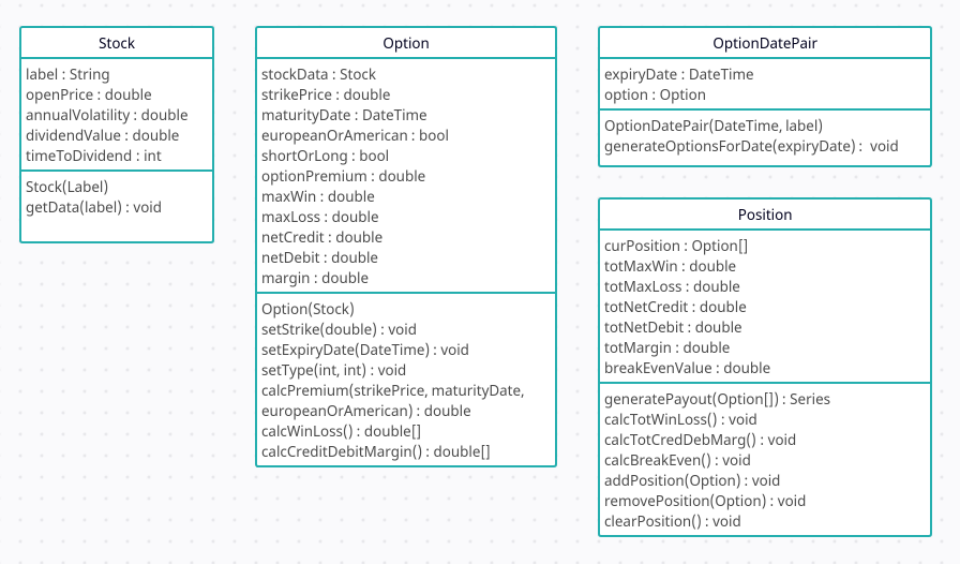
There are also 2 main versions of these types of option. American and European. The main difference is that, European options can only be exercised at the time of expiry. While American options can be exercised at any point in time. We will need to account for this in our code.

The Binomial method is a discreet method, it operates in steps. The root step is the current point in time, the current value (spot price) of the asset is recorded. The child step consists of two values, the value of the first step multiplied by some UP rate, or by some DOWN rate. The volatility of the asset is a big determinant of these rates. At any point in time (a step), the value could go UP or DOWN, hence this creates a binomial tree, each node will have two children. The tree grows over time, up to the expiration date, in an arbitrary amount of steps. Once we have built the tree, we iterate backwards through it to find the option price at the current point in time.

An extremely popular method for pricing options is the "Black-Scholes" method. This is a continuous method, unlike ours. However it is also a probabilistic model, using volatility as a determinant. It assumes that stock prices follow geometric Brownian motion.

System Architecture

This program will be built on .NET Maui for its UI development features and its integration in the .NET ecosystem, allowing me to access an array of useful NuGet packages. For the UI, I will be using Syncfusion, this framework will cover the graphing capabilities and will also be used to make a generally beautiful UI.



This diagram shows every class that will be included in the program, its member data and functions.

An instance of **Stock** is created every time the user enters a new symbol into the main page/strategy builder. The function **getData(...)** will be part of the constructor, it will retrieve the correct data from an API to fill in the other members in the object.

An instance of **Option** is created for every valuation calculated, it includes the user inputs, such as expiry date, strike price etc. But also contains the function **calcPremium(…)**, **calcWinLoss()** etc. However, on the main page a series of **Option** will be displayed on the graph, so a new structure is needed. This is where **OptionDatePair** comes into play, the key being the date of expiration for the value, an instance of **Option**. So if a user chooses to view the “Weekly” graph (after having given all the required data). First the program will figure out the necessary dates, then will create an array of **OptionDatePair**, with one for each of these computed dates. Each option is created with the same data input, except for the expiry date. Each premium is calculated and now we have a nice stream of **OptionDatePair** to display on the graph!

Then we have the **Position** object. This contains a set of **Option,** here is where the data for the overall **Position** is calculated. This will be displayed on the payout plot.

Algorithm

The main algorithm that will be used is the Cox, Ross and Rubinstein method. This appears to be the most commonly implemented way of pricing binomial options. My implementation will use 6 variables to calculate the premium.

- S – Spot price **(Necessary)**
- T – Strike price **(Necessary)**
- rf – Risk-Free rate / Interest **(Necessary)**
- V – Annual volatility (We will then adjust the volatility in the algorithm) **(Necessary)**
- M – Time to expiry **(Necessary)**
- N – Number of steps in binomial tree (Depth of tree) **(Necessary)**
- D – Dividend per share **(Potential)**
- td – Time to dividend payment **(Potential)**

Pseudocode

Step 1 – Build the tree

```
Tree[x, x]
Tree[0, 0] = S
for i in range (N)
    Tree[i, 0] = Tree[i-1, 0] * upvalue
    for j in range(N)
        Tree[i, j] = Tree[i-1, j-1] * downvalue
```

Here the tree is built step by step. It builds all nodes for each step i, then increments i to build (double) the nodes for the next step. It also assigns the node its value. The possible price of the asset at any given time. Up and downvalue are derived from the variables above.

$$upvalue = e^{V \cdot \sqrt{dM}} \qquad downvalue = \frac{1}{upvalue}$$

These are the rates at which the asset would be appreciating/depreciating at. (Side note: Consider a limit tending towards unlimited steps, we are reaching a continuous model, aka the Black-Scholes model)

Step 2 – At each final node, calculate the value of the option.

```
NewTree[x, x]
for i in range(steps)
    if(Call)
        NewTree[steps, i] = max(0, Tree[steps, i]-T)
    else if (Put)
        NewTree[steps, i] = max(0, T-Tree[steps, i])
```

Here we are calculating the expiration value for the option at each final point in the tree. This is the starting point for the next step:

Step 3 – Iterate through the new tree backwards and calculate the option price.

```
for i in range(steps, to 0, i--)
    for j in range(i)
        if(European)
            NewTree[i, j] = decayFunction * (P*NewTree[i+1, j] + (1-P)*NewTree[i+1, j+1])
        else if (American)
            if(Put)
                NewTree[i, j] = max(T-Tree[i, j], decayFunction * (P*NewTree[i+1, j] + (1-P)*NewTree[i+1, j+1]))
            else if(Call)
                NewTree[i, j] = max(Tree[i, j]-T, decayFunction * (P*NewTree[i+1, j] + (1-P)*NewTree[i+1, j+1]))
return NewTree[0, 0]

Binomial(S, T, rf, V, M, N, PutCall, EuroAme)
```

The outer loop iterates backwards starting from the last step, the inner loop iterates over each node at the specified step. The number decreases over time. Depending on the specific type of option, a

calculation will occur. For European options, the value is calculated as the decayed expected value derived from the two previous (future) nodes. P is the probability of an up movement in the assets price at each time step. We must also account for interest in this part.

$$P = \frac{\left(e^{rf\,dM} - downvalue\right)}{upvalue - downvalue}, \quad decayFunction = e^{-rf \cdot dM}$$

For American options, we also need to calculate the intrinsic payoff of the option. As American options could be exercised at any point in time. So we have a max(x, y) containing the two possible scenarios. If we choose to pay off at Tree[i, j] we will take its current payoff, whether it's a put or call. If we don't choose to pay off, we just decay and move on to the next node.

This keeps going until we get to i=0.

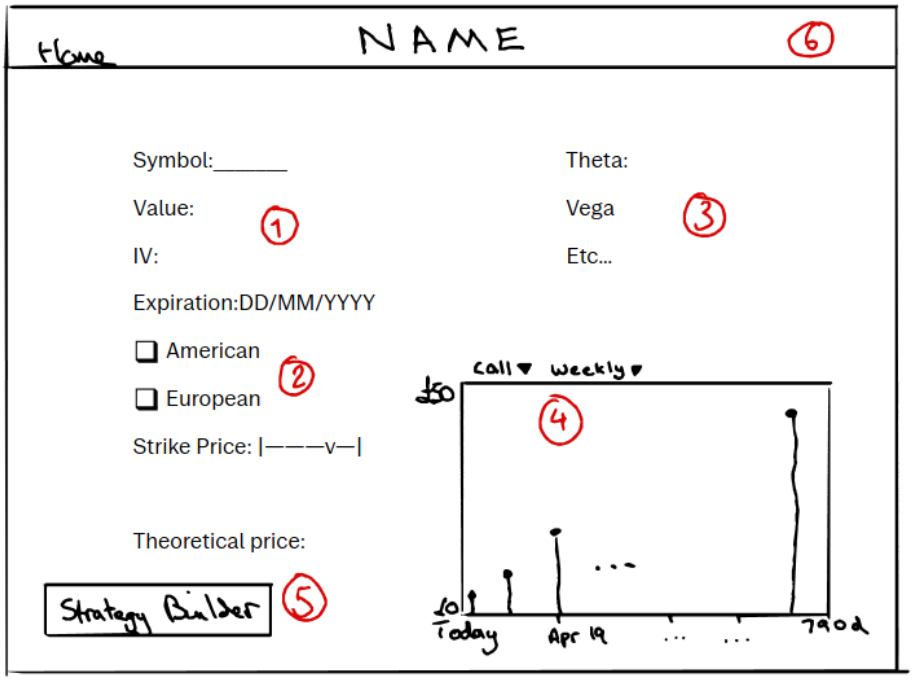
Finally we return the option value at the initial node. This is the present theoretical value of the option!

In addition to this, we could also add the theta decay to each node. Which is rate of decline in the value of an option due to the passing of time. This would be done by calculating the time to expiration for each node in the tree. Then modifying the backwards loop to include the decay. It would just be subtracted from each option value. We must also ensure the values at the final step (calculated in step 2) are also adjusted for this decay.

$$\theta = \frac{\left(NewTree[i,j] - NewTree[i+1,j]\right)}{dM}$$

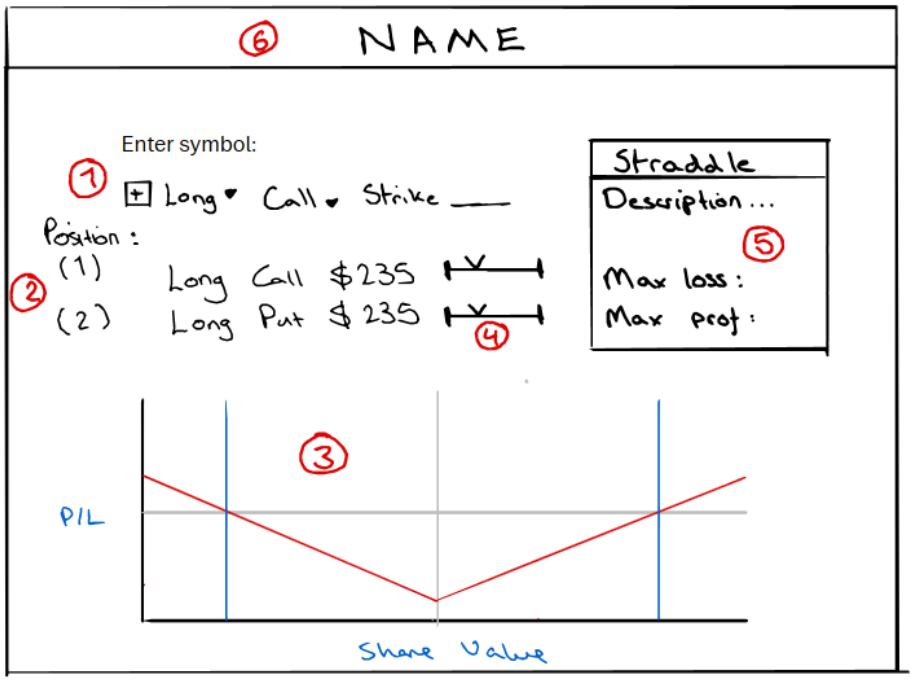
We can also adjust the code to account for dividends, but this adds a fair layer of complexity to the problem, and designing code for this will take longer. We need to ensure the base works first and foremost.

Description of Prototype



1. Upon loading the program, a screen like this should appear. The user is invited to enter a symbol here, once input, the program will fill in the missing data using an API call.
2. The user can decide which type of **Option** they would like to see the value of and also set the expiration date. (Not necessary however, this part of (2) may not exist in final version).

3. Greeks are displayed for further insight, tooltips for people who do not know what the values represent.
4. A graph showing the call/put values for an **Option** along a timeframe. Going from the coming days, to weeks, to years. This series is represented by a set of **OptionDatePair**s. These series can be generated for Weekly datasets, Monthly datasets or in LEAPS (Long Term Equity Anticipation Securities)
5. The user can click on a value in (4) and the value for theoretical price will appear. There's also a "strategy builder" button which will take the user to the next figures.
6. The Navbar, currently very few ideas for this. Maybe the strategy builder button will be moved here.



1. The user can add an **Option** to their **Position.** Long/Short, Put/Call and strike price. Once done, they press the **+** button to commit.
2. The rundown of the current **Position,** can hold multiple options.
3. A visualisation of the payout plot for the specified position. The blue lines being the share values needed for the option to break even. The grey line parallel to the Y axis represents the current

stock price and the one parallel to the X axis represents whether the option will be in the money or not depending on its underlyings value.

4. This slider can be used to change the strike price for the desired **Option**, this alters the overall **Position** and therefore the plot must change. This will be done dynamically so it looks seamless.
5. More data about the **Position** is shown here, including the strategy name, the net win/loss, credit/debit and margin.
6. Similarly to the first figure, there are very few ideas for the navbar, a link to the home page is likely to be here.

Timescales

Work Package	Objectives	Start Date	End Date
1	1-9	1/10/23	5/12/23
2	10-12	1/12/23	5/2/24
3	13-14	1/2/24	5/4/24

I have divided my objectives into three main “Work packages” of (in my opinion) equal complexity. I’m doing this aggregation as there is no way for me to predict when I will finish an exact objective, I just know what I can do given the timeframe.

Work package 1 focuses on the basic implementation of the binomial algorithm, a UI built for this simple calculation which includes a graphical and text output.

Work package 2 focuses on the basic implementation of the strategy builder. By the end of this work package, I should be able to plot one option on the chart. Give the max return/loss and the net margin/debit/credit.

Work package 3 expands on 2. After this is completed, I should be able to build complex strategies (e.g. a straddle) by allowing the user to plot multiple options on the same graph. It’ll implement the strategy recognition. Furthermore, if I have any more time I will polish the whole program, front and backend.

REFERENCES

[1] https://en.wikipedia.org/wiki/Derivative_(finance)

[2] http://www.untag-smd.ac.id/files/Perpustakaan_Digital_1/FINANCE%20An%20Introduction%20to%20Financial%20Option%20Valuation.pdf