**Break-Even Volatility**

*Maxime Bouin, Marcello Mak*

**Aim and Methodology**

The aim of the project is to compute the “break-even” volatility smile for one-year maturity options on SPX Index just by using the historical closing price of the index and to compare the resulting smile with the current quoted volatility skew.

To find the volatility smile, we computed the total PNL of a delta hedged option position throughout the option’s life across different levels of strike. For each strike , we solve the below equation to find the target break-even volatility .

According to the project instruction, we implemented 2 methodologies in computing the PNL:

1. Consider a daily delta-hedging strategy, the total PNL of the delta hedged call option for a long position is

where and are the option price, option delta and underlying price on day respectively.

1. According to the Black-Scholes Robustness Formula, the total PNL of a delta hedged call option position can be expressed as follows:

where and are the option gamma and underlying price on day t, while the and are the implied volatility and the change in time in terms of year respectively.

To find the root of the PNL equation above, we implemented the Bisection method, which is simple and straight forward with the below algorithm:

* Set the initial bound for the volatility
* Calculate the midpoint volatility
* Calculate the absolute difference between the PNL with volatility and 0, i.e.
* If it is within the tolerance, the midpoint volatility is our result.
* If it is not the case, check the sign of
* If and share the same sign, replace by
* If and share the same sign, replace by
* Repeat step 2 – 4 until the is less than the tolerance level

**Program Structure**

Our program is written in 3 major files:

* main.cpp
* break\_even\_volatility.cpp
* break\_even\_volatility.hpp

*break\_even\_volatility.hpp / .cpp*

The files mainly hold the 2 self-defined classes and the auxiliary functions used in the class methods.

* Class time\_series

It is mainly used to read data from csv files with dates of format “dd/mm/yyyy”. By construction, it reads and stores the data and its corresponding dates in 2 separate vectors (a data vector with type double, a date vector with type time\_t). It also includes some member functions for the extraction of data:

* get\_data: return the whole data vector stored
* get\_date: return the whole date vector stored
* get\_datapos: return the iterators of the starting point and ending point of a fraction of the data vector according to the input ending date and length of the period (in terms of calendar days)
* get\_data: return a vector of data extracted from the whole data vector according to the input starting iterator and ending iterator
* get\_date: return a vector of date extracted from the whole date vector according to the input starting iterator and ending iterator
* Class option

It is mainly used to calculate the price and the greeks of a vanilla option. It takes underlying data as a time series object and takes interest rate as a time series object or a constant double. It also takes an identifier to distinguish the type of the option (1 for call, 0 for put). It includes the below member functions:

* BS\_price, BS\_delta, BS\_gamma
* Price and Greeks: return the path of option price according to the maturity date and life term of the option specified when initializing the option object
* get\_underlying\_data: return the underlying price path according to the maturity date and life term of the option specified when initializing the option object
* get\_date: return the date of the data according to the maturity date and life term of the option specified when initializing the option object
* get\_rate: get the rate data with dates matched with that of the underlying data
* modify\_vol: modify the private member volatility of the object
* get\_volatility: return the value of private member volatility
* modify\_strike: modify the private member strike of the object

*Note: The option class object will try to match the date of interest rate and underlying data (with a 7-day date back limit, i.e. the constructor will look up to 7 dates before the underlying data date to find the match interest rate) by construction. In case there is no match interest rate data, an error will be raised.*

* Auxiliary functions
* c\_str\_timet: convert date string with format “dd/mm/yyyy” into time\_t object, it is used in both time\_series class and option class to read input date.
* normalCDF and normalPDF: return the CDF and PDF value of normal distribution, they are mainly used in calculating price and Greeks in the option class

*main.cpp*

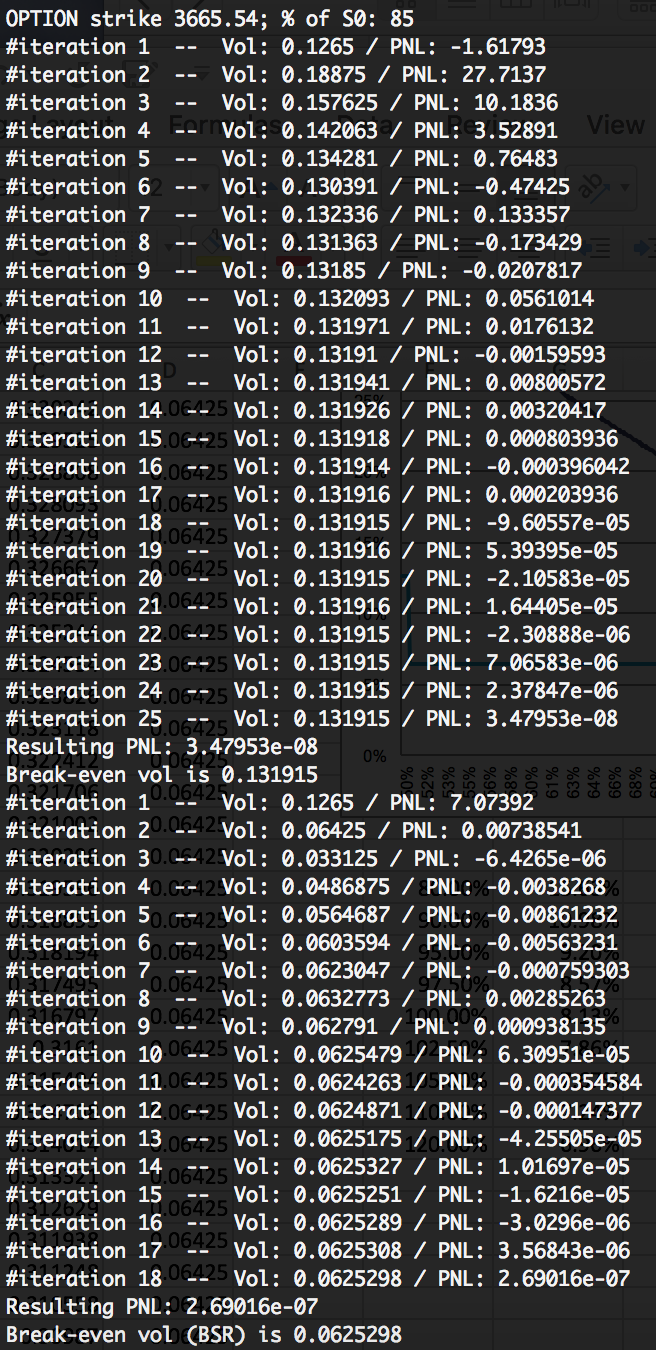
The main.cpp mainly holds the main function, a function for calculating delta hedged PNL, a function utilizing bisection method to find break-even vol and some auxiliary functions used.

* PnL\_Hedged
* It calculates the daily PNL of a delta hedged option position based on the 2 PNL computing formula mentioned:



* We then multiply it by the quantity of option we buy/sell (positive or negative number).
* It takes an option object, quantity of option position and a Boolean as input. The Boolean is used to identify the PNL calculation method to be used.
* breakeven\_vol
* It implements the bisection method mentioned above to find the break-even volatility of the input option. It takes an option object, tolerance level, initial upper bound and lower bound of the volatility and a Boolean as input. Same as above, the Boolean is used to identify the PNL calculation method to be used.

*Note: The function will return a nan if the initial upper bound and lower bound yield a pair of PNL with the same sign as no root could be found by bisection method in this case.*

* main
* The main utilizes the above classes and functions to compute the break-even volatility of an option across different levels of strike based on the 2 PNL calculation methods. User can change the parameters of the calculation in the input area of the main:
* Data filename for both underlying and interest rate
* ****Note that both underlying and interest rate data should be stored in a csv file with 2 columns: date (“dd/mm/yyyy”) and data without header. The csv files should be stored under the same directory as the cpp files
* User can choose to use a constant interest rate instead of a series of interest rate data by commenting the line 28 and 56, and put the interest rate value on line 29
* Target date and the term of the volatility
* Target strike boundary and number of steps within the boundary
* Initial volatility boundary levels
* Tolerance level
* Output file name
* The result will be both printed on the console as follows and saved in a csv file under the same directory of the cpp file.
* Result on the console for a given strike:
* Strike level on the first line (both absolute and %)
* Number of iterations for both methods
* Method 1 on the upper part
* Method 2 (Black-Scholes Robustness) on the lower part
* Resulting Break-even volatility and the associated PNL
* Auxiliary functions
* get\_dir: return the path of the directory of the cpp files

*Note: uncomment line 6 of main.cpp for windows user to include the required header file*

* linspace: return an equally spaced vector with the specified bound and number of steps

**Data and Result**

We considered a scenario that we are selling a S&P500 call option and applying daily delta hedge.

We took daily closing price of S&P 500 total return index from 16 December 2016 to 18 December 2017 as the underlying data, while for the risk free interest rate, we took daily LIBOR 3M data in the same period.

On top of floating interest rate, we have also computed the result with constant interest rate at 0.997% (LIBOR 3M rate on 16 December 2016) for testing.

For floating interest rate, we have the below result:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Strike level** | **80.0%** | **90.0%** | **95.0%** | **97.5%** | **100.0%** | **102.5%** | **105.0%** | **110.0%** | **120.0%** |
| Vol 1 | 15.63% | 10.98% | 9.20% | 8.57% | 8.13% | 7.86% | 7.67% | 7.24% | 6.96% |
| Vol 2 (BSR) | 3.31% | 6.34% | 6.57% | 6.72% | 6.88% | 7.02% | 7.13% | 7.03% | 6.80% |

The Break-even Vol (Vol 1) is the break-even volatility found by using method 1 to calculate the PNL, while the Break-even Vol (BSR) (Vol 2) is the one found by utilizing method 2 (Black-Scholes Robustness formula).

From the above graph we can see that the result of using method 1 is pretty close to the result of using method 2 (BSR) for options close to at-the-money. However, on the deep in-the-money side (low strike level at around 80% of the underlying), the break-even volatility from method 2 tends to become flat and deviates from the one derived from method 1. This may due to the fact that when the option is deep in-the-money or out-of-the-money, the option gamma becomes very close to 0, the resulting PNL calculated from this method will be very close to 0 across all volatilities. This undermines the root finding power of the bisection method, and in fact, the algorithm stops pretty quickly after a few iterations for low strikes and results with a seemingly flat break-even volatility when the option is deep in-the-money or out-of-the money.

We have also computed the result using the constant interest rate and got the below result:

The result of method 1 is out of our expectation, as for most of the strike level before 99%, it could not find a root for the PNL. When we looked into the PNL calculated by the first method, we found that the PNL are strictly positive across different volatility levels for most of the cases. This explains why the bisection method failed to find a root for the PNL. Although the PNL are strictly positive for strike below 99%, we discovered that they are very close to 0 at extreme value (boundary value) of volatility.

For method 2, the calculation of PNL is not affected by the interest rate level, as the formula is derived with the assumption of 0 interest rate. This explains why the result remains the same as the one we got from the constant interest rate.

All in all, as shown by the result we got from the floating rates, we see that the break-even volatility is skewed, with higher value for options with lower strike (deep in-the-money) and with lower and relatively flat value for options with higher strike (out-of-the-money).

**Arbitrage**



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Strike level** | **80.0%** | **90.0%** | **95.0%** | **97.5%** | **100.0%** | **102.5%** | **105.0%** | **110.0%** | **120.0%** |
| Vol 1 | 15.63% | 10.98% | 9.20% | 8.57% | 8.13% | 7.86% | 7.67% | 7.24% | 6.96% |
| Quoted 1Y Imp Vol | 20.93% | 17.02% | 15.24% | 14.38% | 13.50% | 12.55% | 11.53% | 9.91% | 9.94% |

As we can see from the results we have presented previously, the implied volatility we found is generally lower than the market quoted one. If we suppose that our estimations are correct and the market volatility will repeat itself in the coming year, then it means that the implied volatility on the market is overvalued. A strategy of arbitrage could be to be a seller of volatility through the selling of 1Y options with a wide range of strikes for example.