Arthur BAGOURD

Break Even Volatility

Structure of the code

Data manipulation

I defined a class *tsh* (time series handler) that is useful to manipulate time series and that can be initialized directly via data from a CSV file. It consists of a vector of dates and a vector of data of type double as well as a string telling the underlying. This structure allows to access the different elements of the time series. We use it to manipulate spot prices from which we are going to compute option prices.

Financial products

Though I defined a class *instrument* from which we can derive child classes *stocks* and *options*, I am only using the last one in this project. However it would be useful for further improvements of the model.

An *option* is defined by its type (Call or Put), a maturity, a strike, an interest rate, a volatility, an underlying represented by a *tsh* object. Several standard methods allow to compute the price of the option at a given date, to get its delta, its gamma. I also chose to define methods to get the pnl from holding the option, whether it is unhedged or delta hedged. Finally come the methods that compute the break-even volatility of the option given a start date, either by a classic delta hedging strategy, or by a robust delta hedging strategy. To do so, the *get_BE_vol* method is using a dichotomy method. We first compute the PnL from a high vol (hb=1) and a low vol (lb=0) and then compute the PnL by changing the volatility, until the PnL is null (modulo a given tolerance).

There are also two additional methods that allow to retrieve the volatility skew or the volatility surface of the option, given a range of strikes and of maturities.

Volatility

I defined two other classes, a *volatility_skew* and a *volatility_surface*. The first one consists of a vector of strikes and a vector of volatilities for a given maturity. The second one consists of a vector of strikes, a vector of maturities and a vector of vector for the volatilities.

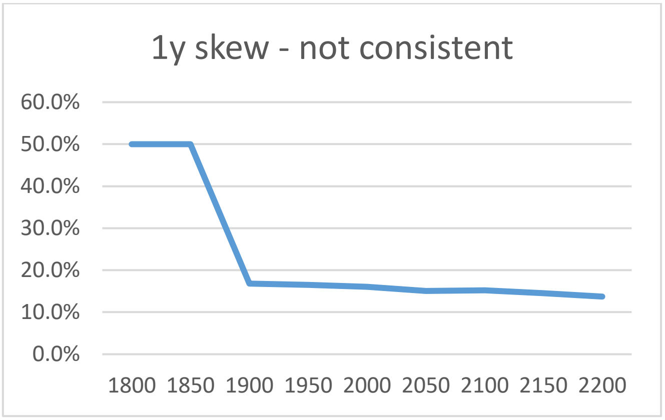
Outputs

The issue of DOTM and DITM options

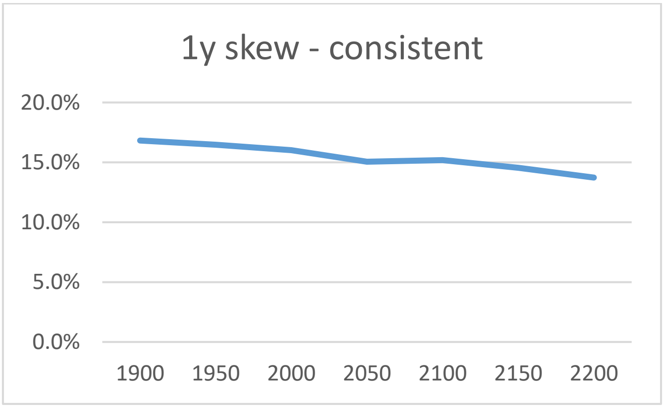
When the strike is very low or very high, the method to compute the pnl does not yield a volatility. This is because the PnLs obtained from a very low vol (low_bound = 0) or very high (high_bound = 1) are of the same sign. Therefore in this case we just output a given volatility that is not relevant.

Volatility skew

Using data from the 2nd of Jan. 15 to the 4th of Jan. 16 we get the following 1 year skew, which is obviously wrong as the two volatilities for the lower strikes are not consistent. This is explained by the precedent paragraph.



Now, removing these two inconsistent volatilities we get the following skew



The skew has the right shape though it is different from the skew we can find on Bloomberg (quoted skew).

Volatility surface

Looking at the graph of the volatility surface with data from the 2nd of Jan. 15 to the 04th of Jan. 16 with maturity 0.5, 0.7 and 1 years and the same strikes than previously, it does look coherent. However I would need to look at historical surfaces to see if the volatility level is the same.

Limits

With data from 2015 to 2017 I got a very different skew and a very different break-even volatility depending on the method on which I am computing it. Maybe I should get historical data from 2010 and average the break-even volatility obtained on the different periods (weighted average if better).

The skew and surface I obtain are different from the markets ones, this might be explained by the different parameters I did not take into account (dividend yields, weekends etc).

Future Work

Getting the right BEV

The first thing to do would be to make the break-even volatility method more robust, especially to high and low strikes for which we cannot get a consistent volatility.

Graphic representation

Graphic representation is very often an important and useful feature as it allows to see directly from the plot the different implications for the trader. For this project, I outputted the surface and the skew in csv files and used excel to get the graphs. However it would be interesting to automatically design the surface and the skew (using gnuplot for example).

Portfolio class

I started to build a class *portfolio* which would allow to hold multiple *instruments* in various quantities, each *instrument* being hold through an *instrument_holder* that define the instrument, its hold quantities over time and its price over time using *tsh* objects.

Then we could compute the different hedging strategies for a *portfolio* holding multiple assets, and compute the break-even volatility for each option. This would be useful to get a more realistic view of a portfolio and its hedging etc.

Other pricing methods

In this project I used the Black Scholes formula to compute the price of the option, but we could easily imagine adding other pricing methods (Heston, SABR etc). This would give the user more flexibility in terms of pricing. I would also add methods that take into account dividends, which I did not do here.

Backtesting framework

Finally, we could extend the tool for backtesting purposes. Creating a class *backtesting* that would hold a *portfolio* built by the user during a given period, using a given hedging strategy and adding other fine-tuning features could make this tool more powerfull. Moreover we could add different performance measures (Sharpe ratio, max drawdown etc) to really allow the user to see the complete results of his backtesting portfolio.