Interactive Visualization of Volatility Surfaces using the Black-Scholes & Merton Model*

1st Bourdeix Corentin
dept. Financial engineering
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland

2nd William Martin
dept. Financial engineering
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland

3rd Kassi Franck Atte Aka
dept. Financial engineering
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland

***Abstract*—In this project we propose an interactive implied volatility surface that will enable investors, traders, or risk managers to keep an eye on the surface moves. The surface can be viewed through an interactive graphical user interface, where the user can set several parameters. The implied volatility has been computed using the Black-Scholes-Merton formula.**

***Keywords*: Black-Scholes, Merton, Surface, Visualization, Volatility**

I. Introduction

Implied volatility (IV) is of major importance in option pricing, risk management and portfolio management. Indeed, this measure, reflects option traders' view of the realized volatility of the security. For this reason, implied volatility is often regarded as a benchmark for realized volatility forecast.

II. The role of Implied volatility in Financial markets

A. Relevant literature

The volatility implied in an option's price is widely used as the option markets forecast of future return volatility of the underlying asset. When options market are efficient, that is roughly speaking when prices incorporate all information available, implied volatility can be seen as an efficient volatility forecast [1], [2]. Even when the market is not efficient, it has been shown that implied volatility has a significant impact in future volatility predictions. Christensen and Prabhala [3] show that while implied volatility is a biased forecast of volatility, it outperforms historical information models when forecasting volatility.

B. Practitioner usage

For practitioners, implied volatility reflects investors beliefs. Indeed, implied volatility generally increases in bearish markets, when investors believe equity prices will decline over time. IV decreases when the market is bullish, and investors believe that prices will rise over time. Bearish markets are considered to be undesirable, hence riskier to the majority of equity investors, therefore IV is used as a tool for gauging the overall market condition and provides guidance for trading.

Implied volatility is one of the deciding factors in the pricing of options. Buying options contracts lets the holder buy or sell an asset at a specific price during a predetermined period. Options with high implied volatility will have higher premiums and vice versa. Implied volatility is also part of several quantitative option strategies. One of them is the volatility statistical arbitrage. The objective of this strategy is to take advantage of differences between the implied volatility and realized volatility of the underlying asset. Such strategy is implemented by trading a delta neutral portfolio of an option and its underlying [4].

In recent years volatility has become a new asset class for investors looking to diversify their portfolio strategy. The complexity of volatility offers a wide range of investment opportunities and has developed from a niche investment for institutional investors to an asset class accessible for retail investors. A common way to bet on volatility is by using markets volatility indices. Most of them are based on implied volatility, the most famous is the VIX which is based on the S&P500 index options [5] [6].

III. The Black Scholes & Merton model

The implied volatility of an option contract is the value of the volatility of the underlying instrument which, when input to an option pricing model, will return the theoretical value of the option equal to the current market price. Hence, finding the implied volatility requires assuming a pricing model for the option. Several option pricing models exist, they often differ by their assumptions on the behaviour of the underlying volatility and return process. In the project we chose the most commonly used option model, namely the Black Scholes & Merton model which assume a constant volatility process [7].

A. The model

For the following parts of this section, we claim the following assumptions:

- The economy is in continuous-time.
- The riskless rate is constant and equal to r so that the price of the riskless asset is $S_{0t} = e^{rt} \ \forall \ t \geq 0$
- Uncertainty is generated by a standard $\mathbb{P}$ - Brownian motion $(W_t)_{t\geq 0}$ with natural filtration $\mathbb{F} = (\mathcal{F}_t)_{t\geq 0}$
- The ex-dividend stock price evolves according to

$$dS_t = S_t \left(\sigma dW_t + \mu dt - \delta dt\right)$$

where the constants $(\sigma, \mu, \delta) \in \mathbb{R}_{++} \times \mathbb{R}^2$ represent respectively the volatility, the drift and the dividend yield of the stock.

B. The call option

Consider a European call option with maturity T and strike k written on one unit of the stock. At the expiration date the option price must equal the payoff:

$$(S_T - k)^+ = \max\{S_T - k; 0\}$$

in all states of nature to prevent any form of arbitrage. Conjecture that the price of the call $c \in C^{1,2}([0,T) \times \mathbb{R}_+)$, together with Itô's lemma this conjecture implies that:

$$\begin{aligned} dc(t, S_t) &= \frac{\partial c}{\partial t}(t, S_t)\, dt + \frac{\partial c}{\partial s}(t, S_t)\, dS_t + \frac{1}{2}\frac{\partial^2 c}{\partial s^2}(t, S_t)\, d\langle S\rangle_t \\ &= \mathfrak{L}c(t, S_t)\, dt + \sigma S_t \frac{\partial c}{\partial s}(t, S_t)\, dB_t \end{aligned}$$

with the differential operator

$$\mathfrak{L} \equiv \frac{\partial}{\partial t} + (\mu - \delta)s\frac{\partial}{\partial s} + \frac{\sigma^2}{2}s^2\frac{\partial^2}{\partial s^2}$$

C. The partial differential equation (PDE)

No arbitrage implies that the option price function must therefore satisfy the Black-Scholes-Merton PDE:

$$rc = \frac{\partial c}{\partial t} + (r - \delta)s\frac{\partial c}{\partial s} + \frac{\sigma^2}{2}s^2\frac{\partial^2 c}{\partial s^2}, \quad \forall (t, s) \in [0, T) \times \mathbb{R}_+$$

subject to

$$c(T, s) = \max\{s - k; 0\}, \quad \forall s \in \mathbb{R}_+$$

proof here [7]

The PDE can be resolve using the Feynman-Kac formula: Consider the auxiliary probability $\mathbb{Q} \neq \mathbb{P}$ under which

$$\frac{dS_t}{S_t} = (r - \delta)dt + \sigma dW_t^Q \Longrightarrow d\hat{S}_t + \delta\hat{S}_t dt = \sigma\hat{S}_t dW_t^Q$$

Itô's lemma and the PDE give

$$\begin{aligned} d\left(e^{-rt}c(t, S_t)\right) &= e^{-rt}\sigma S_t \frac{\partial c}{\partial s}(t, S_t)\, dW_t^{\mathbb{Q}} \\ &+ e^{-rt}\left\{\frac{\partial c}{\partial t} + \frac{\partial c}{\partial s}(r - \delta)S_t + \frac{\sigma^2}{2}S_t^2\frac{\partial^2 c}{\partial s^2} - rc\right\}dt \\ &= \sigma\hat{S}_t\frac{\partial c}{\partial s}(t, S_t)\, dW_t^Q \end{aligned}$$

which shows that the discounted price of the call is a martingale under the auxiliary probability measure $\mathbb{Q}$. Since the discounted price process

$$\left(e^{-rt}c(t, S_t)\right)_{t\in[0,T]}$$

is a $\mathbb{Q}$ -martingale we have that the unique solution to the PDE can be represented as

$$\begin{aligned} e^{-rt}c(t, S_t) &= E_t^Q\left[e^{-rT}c(T, S_T)\right] \\ &= E_t^{\mathbb{Q}}\left[e^{-rT}(S_T - k)^+\right] = e^{-rT}\int_k^\infty (x - k)\mathbb{Q}_t\left[S_T \in dx\right] \end{aligned}$$

This allows to obtain the option price by computing an expectation rather than by solving a PDE.

D. Computing prices

With $\tau \equiv T - t$ we have that

$$S_T = S_t \exp\left[\sigma\sqrt{\tau}\mathbf{n} + \left(r - \delta - \frac{\sigma^2}{2}\right)\tau\right]$$

where $\mathbf{n} = \left(W_T^{\mathbb{Q}} - W_t^{\mathbb{Q}}\right)/\sqrt{\tau}$ is normally distributed under $\mathbb{Q}$ with mean zero and variance equal to one independently of $\mathcal{F}_t \Rightarrow$ Combining this with the Feynman-kac representation of the solution then shows that

$$c(t, s) = e^{-r(T-t)}\int_{\mathbb{R}}\left(se^{\left(r - \delta - \sigma^2/2\right)\tau + \sigma\sqrt{\tau}n} - k\right)^+\phi(n)dn$$

Theorem: The arbitrage price of a European call with maturity T and strike price k is uniquely given by

$$c(t, s) = e^{-\delta\tau}s\Phi\left(d_+(\tau, s)\right) - e^{-r\tau}k\Phi\left(d_-(\tau, s)\right)$$

where the thresholds $d_\pm$ are defined by

$$d_\pm(\tau, s) = \frac{1}{\sigma\sqrt{\tau}}\left(\log\frac{e^{(r-\delta)\tau}s}{k} \pm \frac{\sigma^2}{2}\tau\right)$$

and $\Phi : \mathbb{R} \to [0, 1]$ is the cumulative distribution function of a standard normal random variable.

E. The put option

Following the same procedure, it can be shown that an European put with maturity T and strike price k is uniquely given by

$$p(t, s) = e^{-r\tau}k\Phi\left(-d_-(\tau, s)\right) - e^{-\delta\tau}s\Phi\left(-d_+(\tau, s)\right)$$

IV. Data and Methodology

We have seen that the implied volatility is of major importance for practitioners and that there are several empirical evidence of its pertinence when it comes to forecasting future volatility. In this section we will discuss the data, methodology and algorithms we use to extract and represent the implied volatility.

A. Data

In order to compute the implied volatility for a particular index, various inputs are needed. These include:

- The bid and ask price of call and put options of the index of interest
- The strike price of these options of the index of interest
- The maturity date of these options of the index of interest
- The dividend yield of the index of interest
- The yield curve, that is the yields of bonds over time on which the index is applied

If, for instance, we wish to compute the implied volatility of the S&P index, we would need the entire list of option contracts (strike, bid, ask prices, maturity dates), the dividend yield of that index as well as the U.S treasury bonds' yield curve.

To this end, we have created a custom *Python* class that queries the *Yahoo Finance* database using HTTP requests. The structure of the response is well structured and allowed us to retrieve all contracts for a particular index, provided the correct ticker symbol. After parsing the results into a *pandas DataFrame* and cleaning the data for it to be readily usable, the options can then be used efficiently for the next module which computes the implied volatility. Unfortunately, the *Yahoo Finance* website does not have data of options for all major indices, namely those of the German stock index *DAX* or the benchmark French stock market index *CAC 40*. However, other websites have these data but this would imply creating other custom *Python* classes using different APIs or web-scraping techniques.

The dividend yield however is a variable that is neither available on the *Yahoo Finance* website nor accessible via HTTP queries. Since creating another custom module to extract one value from a another website seemed to be a laborious endeavour (with the need of performing this task for all individual indices we include in our application), we therefore decided to have this variable set as an input attribute when creating the class. This means the divided yield will need to be input by the user via the graphical user interface. We consider this a relatively good choice of concept since having the user choose the dividend yield is more robust and reliable than having to web-scrape a number of websites according to the size of the list of indices we will allow the user to select from.

The yield curve is a delicate matter. Indeed, we cannot have the user input a yield curve as the implementation and robustness of this method would be on one hand cumbersome and tedious for the user and on the other strenuous on the back-end (verification of feasible inputs and assertion of input types). To this end, we have deemed reasonable to create another custom *Python* class implementing individual web-scrapers to extract various yield curves for the indices they are applied to (i.e. treasury bonds for the U.S. market).

Finally, we match the maturities of the yield curve to the expiration dates of our options to create one table containing all relevant data for the implied volatility extraction. In this table, we compute the option price as the mid-price between the current bid and ask prices (average) of the option being quoted.

This procedure is performed every time an instance of these classed is created, which we will see more explicitly when discussing the structure of the graphical user interface.

B. Implied volatility extraction

The Black Scholes & Merton (BSM) formula tells us that the price of the European put and call are function of the time to maturity of the option $\tau = T - t$, the strike k and underlying stock price s and volatility σ. Extracting the implied volatility consists in finding the volatily such that given an option maturity and underlying stock price, the model price matches the market price P. Hence we have:

$$\sigma_{implied} = g(\tau, P, k)$$

The BSM formula does not have a closed-form solution for its inverse. To find the implied volatility we use a numerical root finding technique, the Newton's method.

1) The Newton's method: The Newton's method is an algorithm used to find the root of a function, i.e $f(x) = 0$. The algorithm can be described as follow:

- Step 1: Start from a point x_0 and set a tolerance level $\epsilon > 0$ (e.g. $\epsilon = 10^{-6}$). Set $k = 0$
- Step 2: While $|f'(x_k)| > \epsilon$ compute

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

- Step 3: If $|f(x_{k+1})| < \epsilon$, then accept x_{k+1} as the solution. Otherwise, go back to step 2 for a new iteration.

2) Application: let $f(\sigma, \tau, k, s)$ be the BSM formula that gives the price of an option (call or put) given the volatility, time to maturity and strike. Define the function

$$H(\sigma) = f(\sigma, \tau, k, s) - P$$

where τ, k, s, P are known and P is the market price of the option. The implied volatility solves the equation

$$H(\sigma) = 0$$

Define

$$vega(\sigma) = \frac{\partial H}{\partial \sigma} = \frac{\partial f}{\partial \sigma} = e^{-\delta\tau} s\sqrt{\tau}\phi\left(d_+(\tau, s)\right)$$

Note that $vega$ is the same for the put and the call option.
Using the Newton's scheme, the implied volatility is found as follows:

- Step 1: Start from a point $\sigma_0^{implied}$ and set a tolerance level $\epsilon > 0$ (e.g. $\epsilon = 10^{-6}$). Set $k = 0$
- Step 2: While $\left|H'\left(\sigma_k^{implied}\right)\right| > \epsilon$ compute

$$\sigma_{k+1}^{implied} = \sigma_k^{implied} - \frac{H\left(\sigma_k^{implied}\right)}{vega(\sigma_k^{implied})}$$

- Step 3: If $|H(x_{k+1})| < \epsilon$, then accept $\sigma_{k+1}^{implied}$ is the solution. Otherwise, go back to step 2 for a new iteration.

C. Bounds for implied volatility

The BSM model is a quite restrictive model since it does not allows the implied volatility to depend on the strike and the maturity and the price of the underlying. As a result, the model is not able to price all options observed on the market. In order to avoid negative and very large implied volatility we should only consider IV extracted from option prices that are consistent with the BSM model. Following the work of Jim Gatheral et al. [8], we select options using the following bounds on the implied volatility:

$$-2\Phi^{-1}\left(\frac{1-c}{2}\right) \leq \sigma_{implied} \leq -2\Phi^{-1}\left(\frac{1-c}{1+e^{k}}\right)2$$

where k is the strike and c the price of the call option. For put options, similar bounds can be computed using the put-call parity replacing c by $c^* = p + e^{-\delta\tau}s - e^{-r\tau}k$

D. Type of plot

Following the procedure described in previous sections, we extract the implied volatility of the options for a given market index. The results are stored in a *pandas DataFrame* with the corresponding strike, time to maturity and option price.

In order to represent the implied volatility surface in 3 dimensions, we use the *Plotly* library. The reasons we chose *Plotly* are simple. Firstly, the former enables to generate beautiful interactive 3-dimensional graphs that allow the user to zoom and rotate. Hence with this type of plot, the user can quickly see where are the interesting areas of the surface and immediately retrieve the relevant coordinates. Secondly, it is compatible with the *Dash* library that will allows us to build an interactive user interface, which we will see later in this paper.

To generate the surface we create a function that takes as input a X-vector of shape N, a Y-vector of shape M and a Z-matrix of shape (N, M).The Z-matrix gives the volatility for each coordinate of the graph. In the following section we will discuss the challenges we faced when computing this matrix

E. Data dimensions

The major issue we faced when generating the volatility surface was the non-uniformity of the data. Indeed, we are not sure that one strike will be available for all maturities and vice versa in order for the Z-matrix to have values at every coordinate. Hence we need to make the data uniform, that is for coordinates where there are no values, we need to find a way to approximate what value it should have been. This uniformity is obtained by calculating the average of the volatility in a maturity and a strike interval. We cut the map X, Y in squares and we compute the average volatility in each square, then we forced the volatility to be in a certain row and a certain column in our map. This process enables to have a smooth surface.

After analyzing the data we observe that there are more distinct values for the strike as there are for maturities. It will therefore be preferable to have a higher precision on the maturity axis due to the few number of different values for the maturity. Hence, we will not create squares to compute the average volatility but rather intervals in the strike range. We compute the average of the volatility for the same maturity in each interval of the strike, then we apply the following algorithm :

- Step 1: initialize the strike vector (where each component is unique and sorted)
- Step 2: compute h the step between two values of strike
- Step 3: for m different maturities do :
- Step 4: for s different strikes do :
- Step 4.1: select the volatility for maturity m and strike $\in [s - \frac{h}{2}, s + \frac{h}{2}]$
- Step 4.2: compute the average of the values for the volatility selected and store the result

With this algorithm we obtain a matrix where row indices represent a maturity, and columns a strike value. The components of the matrix are the average of the implied volatility. To make sense with the averaging algorithm above, we compute the half number of the different strikes we have in the data and we impose a constant step between each strike value in our grid.

It is sometimes possible to not have any observations in an interval. We can imagine that in our data options with a six-month maturity are only available for high and low strikes but not for moderate ones. We therefore need to smooth the results. We choose to do this only on the strike axis because we have more data on this axis, and thus the smoothing can be more precise. We then perform a linear regression between two non-empty intervals and we use the results of the regression to fill the empty components of the matrix. We do a linear regression by parts to fill the empty observations: this approximation is justified from the fact that the implied volatility will not greatly fluctuate if we move a little on the strike axis.

To summarize, we need to regroup for each different value of maturity the volatility by strike and compute the average. When one component of the matrix is empty, we need to fill it with a linear approximation. We thus have the following algorithm where maturity, strike and volatility define the data we get:

- Step 1: initialize the maturity vector of shape M (where each component is unique and sorted)
- Step 2: initialize the strike vector of shape N (where each component is unique and sorted)
- Step 3: create a empty list $MISS$
- Step 4: create the volatility matrix of shape (N, M)
- Step 5: compute h the step between two consecutive

strike values

- Step 6: for m different maturities do :
- Step 6.1: for s different strikes do :
  Select the volatility where maturity is m and strike $\in [s - \frac{h}{2}, s + \frac{h}{2}]$
  Compute the average of the volatility selected and save it in the volatility matrix in cell $[s, m]$
- Step 6.1.1: if the volatility at cell $[s, m]$ is still empty (i.e no volatility has been selected) then :
  append s to the list $MISS$
- Step 6.2: end for
- Step 6.3: initialize a variable pointer to 1
- Step 6.4: while pointer $\leq$ number of elements in $MISS$:
- Step 6.4.1: create a empty list $MIDDLE$
- Step 6.4.2: append the element of $MISS$ pointed by pointer
- Step 6.4.3: while $MISS$(pointer +1) = pointer +1 do :
  add $MISS$(pointer +1) at $MIDDLE$
  pointer = pointer+1
- Step 6.4.4: end while
- Step 6.4.5: initialize a variable i to 1
- Step 6.4.6: initialize a variable ending as the maximum of the first element of $MIDDLE$ less one and zeros
- Step 6.4.7: initialize a variable starting as the minimum of the last element of $MIDDLE$ more one and the number of element of the strike vector
- Step 6.4.8: initialize a variable coefficient as $\frac{Volatility(ending) - Volatility(starting)}{Strike(ending) - Strike(starting)}$
- Step 6.4.9: for s in $MIDDLE$ do :
  $Volatility(m, s) = Volatility(starting) + h \times coefficient \times i$
  i= i+1
- Step 6.4.10: end for
- Step 6.4.11: pointer = pointer+1
- Step 6.5: end while
- Step 7: end for

Hence with this algorithm we can make the data uniform and obtain one vector of maturity, another with the strike and a volatility matrix with no empty components. Now, we just have to plot it.

F. Adjustments to the 3D plot

We can finally plot the 3D graph using the *Plotly* library where the X-axis represents the maturity that we transformed into a string type, the Y-axis the strike and the Z-axis the volatility. We set the background color of the plot to dark grey in order for the viewing of the graph to be more comfortable. The coordinates of the graph are displayed when hovering the mouse over the graph. The result is depicted in Fig.1.

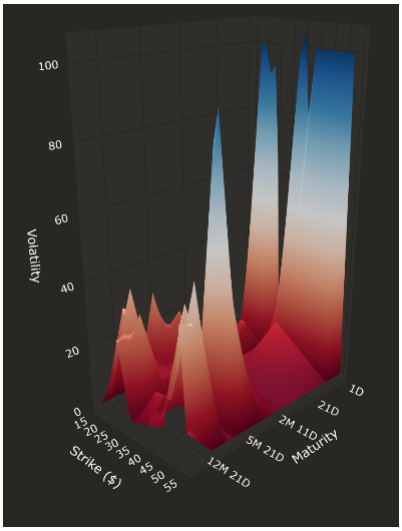

Fig. 1. 3D plot of the volatility surface using the *Plotly* library

V. User interface implementation

In order to provide a truly interactive experience to the user, we have decided to create a graphical user interface (GUI) with the help of the open-source *Dash Python* library. *Dash* is a tool to create web-based data analytic applications using only *Python*, therefore getting around the *HTML*, *CSS*, and *Javascript* tools that make all websites we visit on a web browser beautiful and interactive.

Websites are made of *HTML* objects, designed with *CSS*, and made interactive with *Javascript*. To this end, *Dash* enables the creation of *HTML* objects with *Python* classes. Thus, it is also important to be knowledgeable on how websites are constructed. This can be easily done by inspecting web elements on any website. In order to design our interface, we wrote a *.css* file containing the styling of each of our *HTML* objects which is then served by *Dash* to the website. For the interactivity of objects such as displaying the volatility surface when a button is pressed, *Dash* introduces *callbacks* which are decorator methods of the *Dash* application and allow to change the states or attributes of *HTML* objects when the user interacts with another *HTML* object. For our application, we

have added the following *HTML* objects in our GUI with their described utility:

- Tabbed sections: as the name suggests, this is to separate the application into different sections. For conviviality, the user can navigate through two tabs. The first displays a small description of what the GUI is about and how to use it. The second tab allows the user to choose various inputs for plotting the volatility surface and to display the aforementioned 3D plot
- A drop-down menu is used to give the user the possibility to select a index among a plurality of indices. When a index is selected, the appropriate yield curve will be associated with it when creating the table of options and yields described beforehand
- An numeric input allowing the user to enter the dividend yield (in percentage) of the index selected. As discussed before, having the user provide this value makes the code more robust and resilient
- A button allowing to plot the volatility surface. When the button is pressed, the states of all other *HTML* objects are taken into account (choice in drop-down menu, numeric) and the plot of the volatility surface is displayed. The time for displaying might vary depending on the internet connection (to download the options and bonds) and the computing power of the computer on which the application is running
- A *Graph* object specific to *Dash* containing the 3D plot of the volatility surface to be displayed. At first when the user opens the application this container is empty since no inputs have been selected by the user yet

When the button to plot the volatility surface is pressed, the following sequences of code is run:

1) Create an *Python* object which takes as attribute the numeric input of the user corresponding to the dividend yield of the index and the selected ticker symbol corresponding to the selected index from the drop-down menu
2) Create a list of available expiration dates available on *Yahoo Finance* through a first *HTTP* request. These requests are repeated for a maximum number of retries time in order to correctly retrieve the data in case of connection errors. These dates are stored in different types. Firstly, epoch time which corresponds to the total number of seconds which have elapsed since January 1, 1970; this is a standard way of writing time across programming languages as the need to specify a time-zone becomes redundant. Secondly, the *string* approach which will be used when displaying the label of maturity on the volatility surface. This can easily be done using *timestamp* objects in *pandas*, where the display and print methods of these objects have been conveniently overwritten to show the date in human-readable format
3) For every expiration date available, download the corresponding options and concatenate the results in a table. This is done by sending a second set of *HTTP* requests to the Yahoo Finance database. A column with the maturity date is added in order to differentiate options with the same price but different expiration date. Moreover, a column with the type of option (call/put) is also added. Finally the price of the option is computed by taking the average between the bid and ask prices. The requests here are also repeated a number of times in order to circumvent connection errors with the database. If the number of maximum retries has been reached, the application reads a back-up file containing the most recently downloaded options of the index selected. If the request is successful, the options are saved in a back-up *.csv* file in case of future failed requests
4) After the options are created and placed inside the results table created in the previous point, we now web-scrape the yield curve corresponding to the index selected. The yield curve is then merged intelligently into our result table such that each row of options has a corresponding risk-free rate. Note that a different website is scraped for the yield curve depending on the value selected in the drop-down menu. A back-up of most recently downloaded yield curves is saved in *.csv* files in case the web-scraping fails or if the website is unavailable
5) The results table containing the options and yields, as well as the dividend yield are then sent to a method created to compute the implied volatility (see section on extraction of the implied volatility)
6) A 3D plot using *Plotly* is then generated by following the procedure described in the previous section and displayed in the container of the *Graph object*

Once all steps of code are run, the user will not only be able to see the 3D plot but also interact with it. This means that the user is able to click-and-hold on the plot and rotate the image in any of the 3-axis directions. Moreover, when hovering the mouse over data points of the plot, the user is able to see specific details such as the value on all 3 axes. This is useful for an experimented practitioner to carefully asses the surface drawn. From this point on, the user can select new values of dividend yield or ticker symbol of the index and plot new volatility surfaces.

VI. Deployment

At the current state of the development of this application, the former can only run locally on the machine from which it is executed. Indeed, the open-source version of *Dash* only permits a local distribution of the application, thus the user needs to execute the code on his or her machine and open a web browser with the following URL address: http://127.0.0.1:8050, which is the common address for local hosting. In the scope of our project, we have made it so that a web page is automatically opened when the script is executed.

A. Real-life deployment

In order to fully deploy the application over the world wide web (WWW), one would need to deploy the application on

a production Linux/Unix server (as opposed here to a development server). Several tools for managing and dispatching clients over the internet do this freely such as *Heroku* or *Gunicorn* but their free-tier versions only allow handling of a limited number of visitors on the web page at once. Moreover, in order to make our application scalable, replicable and cross-compatible between operating systems, one can user *Docker*. *Docker* is a famous tool used to deploy applications, such as what we have created, easily and rapidly when going into production.

Finally, as these tools are outside the scope of this paper and our field of expertise, we decided to leave our application as a local executable.

B. System installation

We have however control on how our application should be installed on a system, whether it be on a Linux machine for production or any other machine used to execute our program. To this end, we use the library *setuptools* in order to build and distribute our package *volatility_surface* which is available on *Github*. The installation of the package can be done in 3 easy steps:

1) *git clone* the repository from *Github* in order to create a local copy on the machine
2) Create a virtual environment and activate it. Most common tools to create environments are *anaconda*, *virtualenv*, *pip*
3) *cd* to the copied repository and run the following command: *pip install .* (with the full stop at the end)

After installing the package, the module can be used anywhere (as opposed to running scripts at the direct parent folder of the package or modifying messy *PATH* environment variables) as long as the virtual environment is activated. The *CSS* styling is read from an external source (https://codepen.io) in order for the package to operate correctly when installed as a package outside the scope of the repository of *Github*. Indeed, the *CSS* can normally be served from the *assets* folder which can be found in the root repository, however when installing the package elsewhere, this folder will not necessarily be present when executing the *main* script. Moreoever, this is a reasonable implementation decision since the application will in any case need internet connectivity (to access the *Yahoo Finance* database for instance). Having an installable package as described here allows better portability and flexibility on any system when deploying, testing, or developing the package. Moreover with this type of install method, the package can more easily be deployed when using *Docker* images.

VII. Code maintenance

As this was a collaborative work, *Github* was used to host and version control the code used for the application.

The package listed on *Github* is dependent on other open-source libraries such as *Dash*, *pandas*, and *scipy*. As these libraries evolve and change, backwards compatibility might be an issue in the long-term. This is why our package is hosted on *Github* to enable new or old collaborators to participate in this open-source project. Thus, improvements, modifications and changes can easily be done.

VIII. Results

When the user runs the file *main_app.py*, a new page on a web-browser is opened at the URL http://127.0.0.1:8050/. This first page presents our program, its goal and how to use it (Fig.2).

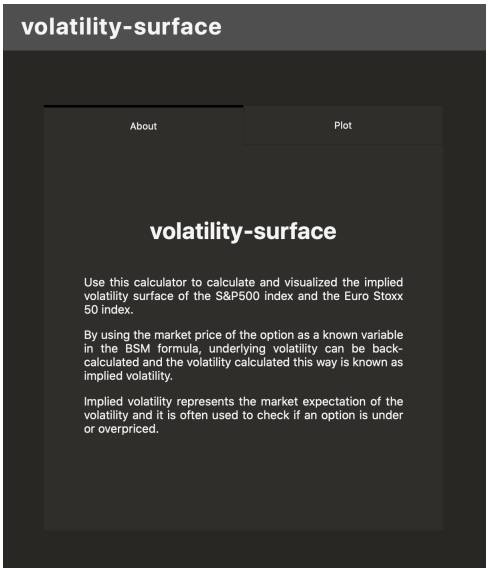

Fig. 2. First page of the web application

Then on the second tab *plot* of the window which we can see in Fig.2, the user can select the market index via a drop-down menu and input the dividend yield in percentage (Fig.3). The default value is 0%, and the minimum and maximum values accepted are respectively -10% and 10%. After selecting the parameters, the user can then press the *PLOT* button to visualize the 3D volatility surface.

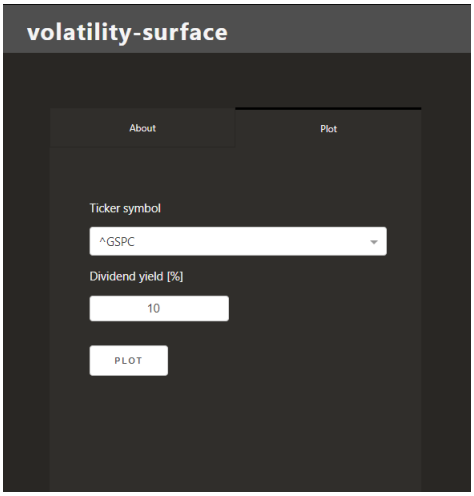

Fig. 3. Second page of the web application

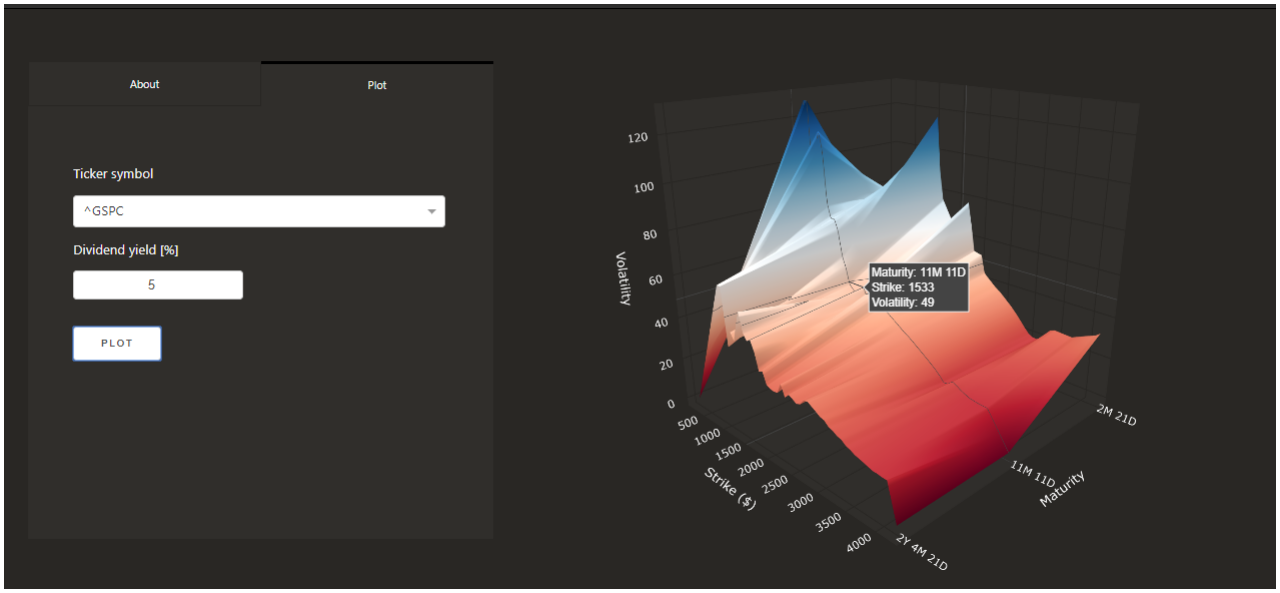

Fig. 4. Visualization of the volatility surface via a web browser

On the volatility surface the user can hover the mouse over the plot and obtain the coordinates of each point as shown in Fig.4, allowing the user to obtain detailed information of the surface. We choose a dark background color in order to not bother the viewing experience of the user, who could be using this interface several times per day.

Finally, the user can move the graphic with the right click of the mouse, rotate it with the left click and zoom with the scroll of the mouse.

IX. Conclusion

Being able to keep an eye on the implied volatility of major indices is essential for the professional activity of practitioners and specialists alike. Indeed for some time, implied volatility is playing a major role in the portfolio strategy of investors and has become a crucial factor in the pricing of options. Even more so when today's financial sector relies heavily on information being readily available as fast as possible.

Throughout this project, we have created a user-friendly graphical user-interface to visualize the implied volatility of various major indices. The implied volatility is computed by using the Black-Scholes & Merton model which requires access to data such as option prices and their maturity date of major indices, the dividend yield of the former as well has the risk-free rate of the market in which the index operates. In this context, we created an portable framework for downloading data through various websites with the help of available APIs or custom-built web-scrapers.

The implied volatility can only be meaningful when it can be beautifully and clearly visualized with a user interface that is cleanly defined. To this end, we exploited the open-source frameworks *Plotly* and *Dash*. The sleek design of our interface allows the user to comfortably visualize and analyse volatility surfaces, namely with the integration of a dark-mode color schema.

Having developed this application with *Dash* opens the possibility of deploying it online by running it on a production server. Being able to access this application anywhere via a laptop or smartphone allows the user to be informed at all times which is essential to any professional willing to advance in his or her career. We have however not fully deployed our application in the scope of this project as this is outside our field of expertise.

Finally, several improvements could have been made to our application if time and deeper competences permitted. Firstly, various custom classes could have been built to integrate more major indices. Secondly, being able to visualize more than one volatility surface at once could be practical when comparing various markets. Lastly as mentioned before, the application could be deployed and hosted on a website which would allow any user to access our visualization platform on any connected device.

This project is a good depiction of how multidisciplinarity is essential when creating such an application. From the mathematical aspects of finance to the algorithmic side for visualizing the volatility surface, both the back-end and front-end development have been worked on. Moreover, more knowledge on the TCP/IP and IT departments would have allowed us to fully deploy our application as a final product.

REFERENCES

[1] C.R. Harvey, R. Whaley, "Market volatility prediction and the efficiency of the S&P 100 index option market", Journal of Financial Economics, vol. 31,pp. 43–73, 1992.

[2] A. Sheikh, "Stock splits, volatility increases and implied volatility", Journal of Finance, vol.44,pp. 1361–1372,1989.

[3] Christensen, B. J. and N. R. Prabhala, "The relation between implied and realized volatility", Journal of Financial Economics, vol. 50,p. 125—150,1998.

[4] Carol Alexander and Anca Dimitriu, "Indexing and Statistical Arbitrage",The Journal of Portfolio Management,vol.31,p. 50–63,2005.

[5] Dan Galai and Menachem Brenner," New Financial Instruments for Hedging Changes in Volatility", Financial Analysts Journal,vol. 45,p. 61–65,1989.

[6] Campbell R. Harvey and Robert E. Whaley,"Dividends and S&P 100 index option valuation", Journal of Futures MArkets,vol. 12, p. 123–137,1992.

[7] F. Black and M. Scholes,"The Pricing of Options and Corporate Liabilities", The Journal of Political Economy, Vol. 81,p. 637–654,1973.

[8] G. Gatheral, I. Matic,R. Radoicic and D. Stefanica, "Tighter Bounds for Implied Volatility", International Journal of Theoretical and Applied Finance, Vol. 20,1750035-1,2017.

[9] Yahoo Finance - Stock Market Live, Quotes, Business & Finance News. Accessed 7 May 2020, from https://finance.yahoo.com/