Programming Project

*Note: Sub-titles are not captured in Xplore and should not be used

1st Elyes Mahjoubi
MsF, Programming
UNIL

2nd Jérôme van Eck Duymaer van Twist
MsF, Programming
UNIL

***Abstract*—Through this project, we propose a program that is able to estimate the price of different types of options (European and American), as well as forecasts of the volatility or price movements of either a stock or a commodity, which can be chosen by the user. In order to give these estimates, we used data from Yahoo finance. The applied theories in order to estimate the European options are Monte-Carlo simulations and Black Scholes. In order to estimate the American options, we had to use a different theory, we applied binomial option pricing. We also apply a GARCH model analysis in order to estimate future forecasts of volatility.**

I. Introduction

Option pricing is not something that you can do by just whipping out your calculator. You need to have data and analyse it and apply certain principles in order to get correct values. This is why we chose to create a program that is able to get data online and give the user the price of an option (call or put and European or American) on a stock of his choosing. This program is also able to estimate different forecasts such as the volatility of a chosen stock or commodity or even let the user visualize the movement of a certain stock/commodity.

First of all, in this paper, we are going to do a more profound description of what the program actually does and how it is able to do so. Then, we are going to explain the methodology and the theory behind all the calculations that the program is doing. In addition, we are then going to talk about how was the theory and the code implemented into the program. Next, we will talk about the way the code has to be maintained and keep the code base up to date. Finally, we are going to talk about the results obtained.

II. Description

When you run the app, the user is asked to enter a command.For all the commands, the user will be asked if he wants to include the dividend yields into the calculations. Including them will make the calculations take a bit longer as the program has to look on the website for the information. If the user doesn't know the commands, it is possible to enter the command: $'help'$. This command outputs the detailed list of command that the user can use. The first command (*1*) is to estimate an European option. with this command, the user is able to decide whether he wants an estimation based on a Monte-Carlo simulation or a standard estimation using Black Scholes. This command can price either a put or a call, the user is the boss. The user is also able to decide which asset to base the option on. The program is able to fetch the data of that particular asset on Yahoo finance. The user then has to decide the variables he wants for his option (strike price, the expiration date). The program then outputs the average price of the option, the maximum price, minimum price as well as the probability of exercising the option.

With the command *2*, the program will initiate an estimation of the future forecast of the volatility of a chosen stock or commodity which is chosen by the user. As for the European option, the user is asked to choose an asset. Then there will be a GARCH model analysis of said asset and the user will be asked for the expiration date desired.The app will then output the conditional volatility for that time duration.

Now if the user wants to see a screening menu for European options, he will have to use the command *3*. The program will ask the user if he is a Buyer or a Seller. It will then ask which stock composites the user wants to use (NASDAQ, DOW or SP500). The program is then going to ask whether the user wants a screener of American or European options and it will finally load a wide range of assets and display the probability of exercising a call or a put option. Based on these probabilities, the program will make a selection and output the probabilities as well as the strike price of these assets onto an excel spreadsheet.

For the pricing of an American option(command *4*), it is not priced the same way as the European option. The user has to input the same data as for the European option, however the method used to estimate the price is binomial option pricing. The program will then output the price of the American Option. It will finally ask you if you want to download the calls and puts that are available in the market in two files named: $Calls$ and $Puts$ followed by the ticker of the chosen asset

As for the second to last command (*5*), it allows the user to visualize the movement of the stock or commodity of his choice from a certain point in time (also chosen by the user) up to 01-01-2019.

Finally, when the user has had enough with the pricing of options and looking at his favorite asset's forecast, he can use the final command which is *'exit'*. It is self explanatory, when the user enters that command the program will stop running.

III. Methodology

In this section, we are going to talk about the theory applied in order for the program to do its calculations. We used the theory from others courses of the MsF. However the main sources were the course of Fixed income and credit risk, as well as Empirical methods in Finance.

First of all, in order to price the European options, we offer the user the possibility to use a Monte-Carlo simulation. A Monte-Carlo simulation is computational algorithm that relies on doing repeated random sampling in order to obtain the statistical values of a certain phenomenon (here the different possible movements of an asset). This simulates a lot of different possible asset price evolution and then generates an average option price based on these simulations using Brownian motion.

First of all, for the brownian motion, we have a stochastic process S_t which follows a Geometric Brownian Motion (GBM) if it satisfies the Stochastic differential Equation (SDE):

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

with:

- W_t being a Wiener process (Brownian motion)
- μ is the percentage drift
- σ is the percentage volatility we extracted from the AR(1)-Garch(1,1) model

we get the following analytic solution under Itô's interpretation with S_0 being an arbitrary initial value:

$$dS_t = S_0 exp((\mu - \frac{\sigma^2}{2})t + \sigma S_t dW_t)$$

When we apply Itô's calculus, we obtain:

$$d(lnS_t) = (lnS_t)' dS_t + \frac{1}{2}(lnS_t)'' dS_t dS_t = \frac{dS_t}{S_t} - \frac{1}{2}\frac{1}{S_t^2} dS_t dS_t$$

with $dS_t dS_t$ being the quadratic variation of the SDE.

$$dS_t dS_t = \sigma^2 S_t^2 dt + 2\sigma S_t^2 \mu dW_t dt + \mu^2 S_t^2 dt^2$$

When dt $\to 0, dtwillconvergeto0 fasterthandW_t$. Therefore, we can simplify with:

$$dS_t dS_t = \sigma^2 S_t^2 dt$$

Now when we plug in the value of dS_t into it and simplify, we obtain the following:

$$ln\frac{S_t}{S_0} = (\mu - \frac{\sigma^2}{2})t + \sigma W_t$$

All we need to do now is to use an exponential and multiply by S_0 to get the result:

$$dS_t = S_0 exp((\mu - \frac{\sigma^2}{2})t + \sigma S_t dW_t)$$

The solution S_t follows a log-normal distribution with the following expected value and variance:

$$E(S_t) = S_0 e^{\mu t}$$
$$Var(S_t) = S_0^2 e^{2\mu t}(e^{\sigma\sigma t} - 1)$$

This model will allow us to simulate the evolution of S_t

Now in order to price an option without using a Monte Carlo simulation, we used the Black Scholes formula.

The formula to price an option (call and put respectively) with Black Scholes is the following:

$$C(S,T) = e^{-qT} S_0 N(d_1) - e^{-rT} K N(d_2)$$
$$P(S,T) = e^{-rT} K N(d_1) - e^{-qT} S_0 N(d_2)$$

with:

$$d1 = \frac{1}{\sigma\sqrt{T}}[ln(\frac{S_0}{K}) + (r + \frac{1}{2}T)]$$
$$d2 = d1 - \sigma\sqrt{T}$$

$C(S,T)$ being the price of a call option and $P(S,T)$ the price of a put.

- S_0 is the actual value of the underlying
- K is the strike price
- T is the time that is left before the option expires
- r is the risk free rate
- σ is the volatility of the underlying.
- q is the dividend yield of the underlying.

We computed the delta which helped us to find the option price change for the European calls and puts using the formula:

$$\Delta_{call} = \Phi(d_1)\Delta_{put} = \Phi(d_1) - 1 \quad (1)$$

*$Delta_{put}$ varies between 0 and -100%,$Delta_{call}$ varies between 0 and 100%

Deltas permitted us to estimate the maximum option price for the best/worst trajectory the Monte carlo simulation using Browenian motion predicted. For the second command, in order to estimate a forecast of the future volatility of an asset, we used the GARCH model and an AR(1) process.
For the AR(1) process, which means Auto-Regressive process, it is the first order auto-regressive process. r_t, a time series. ϵ_t is a white noise with $E[\epsilon_t] = 0, V[\epsilon_t] = \sigma_\epsilon^2 < \infty$ and $Cov[\epsilon_t, \epsilon_{t-k}] = 0, \forall k \neq 0]$.

$$E[r_t] = \mu$$

$$V[r_t] = \gamma_0 = \sigma_\epsilon^2 \sum_i^\infty {}_{=0} \theta_i \theta_{i+k}$$

which gives us:

$$Corr[r_t, r_{t-k}] = \rho_k = \frac{\sum_i^\infty {}_{=0} \theta_i \theta_{i+k}}{\sum_i^\infty {}_{=0} \theta_i^2}$$

If the number of θ-weights is infinite, we can assume that the weights are summable. Which means that r_t is stationary. The AR(1) process assumes that the first lag of the process r_{t-1} is useful to forecast r_t. We get the following:

$$r_t = \phi_0 + \phi_1 r_{t-1} + \epsilon_t$$

Now if we call L the lag operator with which $Lr_t = r_{t-1}$, then:

$$(1 - \phi_1 L) r_t = \phi_0 + \epsilon_t$$

which will then give us:

$$r_t = \frac{\phi_0}{(1-\phi_1)} + \frac{1}{1-\phi_1 L}\epsilon_t = \frac{\phi_0}{(1-\phi_1)} + \sum_{i}^{\infty} {}_{=0}\phi_1^i \epsilon_{t-i}$$

As long as $|\phi_1| < 1$, this linear filter will converge. It is called the weak stationarity condition.

Under stationarity, we get the following moments of r_t: $E[r_t] = \mu = \frac{\phi_0}{(1-\phi_1)}$
and $V[r_t] = \gamma_0 = \sigma_\epsilon^2 \sum_i^\infty {}_{=0}\phi^{2i}{}_1 = \frac{\sigma_\epsilon^2}{1-\phi_1^2}$

For the GARCH model which stands for $Generalized\ Auto-regressive\ Conditional\ Heteroskedasticity$ The GARCH model is defines as the following:

$$r_t = \mu + \sigma_t z_t = \mu + \varepsilon_t \tag{2}$$

$$\epsilon_t = \sigma_t z_t$$

$$\sigma_t^2 = \omega + \sum_{i}^{p} {}_{=1}\alpha_i \epsilon_{t-i}^2 + \sum_{j}^{q} {}_{=1}\beta_j \sigma_{t-j}^2$$

The constraints on the parameters of the GARCH to ensure strictly positive variance are:

- $\omega > 0$
- $\alpha_i \geq 0$ for $i = 1, ..., p$
- $\beta_i \geq 0$ for $j = 1, ..., q$

The unconditional variance of r_t is then given by :

$$\sigma^2 = \frac{\omega}{(1 - \sum_i^p {}_{=1}\alpha_i - \sum_j^q {}_{=1}\beta_i)}$$

The process ϵ_t is therefore stationary if and only if the sum of α_i and β_j is less than one and $\omega < \infty$..

We find the Parameters using a Maximum Likelihood regression:

$$f(\varepsilon_0, ..., \varepsilon_T; \theta) = f(\varepsilon_0; \theta) f(\varepsilon_1, ..., \varepsilon_T | \varepsilon_1; \theta) \tag{3}$$

$$= f(\varepsilon_0; \theta) \prod_{t=1}^{T} f(\varepsilon_t | \varepsilon_{t-1}, ..., \varepsilon_0, ; \theta) \tag{4}$$

$$= f(\varepsilon_0; \theta) \prod_{t=1}^{T} f(\varepsilon_t | \varepsilon_{t-1}, ; \theta) \tag{5}$$

$$= f(\varepsilon_0; \theta) \prod_{t=1}^{T} \frac{1}{\sqrt{2\pi\sigma_t^2}} \exp\left(-\frac{\varepsilon_t^2}{2\sigma_t^2}\right) \tag{6}$$

We finally get:

$$L(\theta) = \sum_{t=1}^{T} \frac{1}{2}\left[-\log 2\pi - \log(\sigma_t^2) - \frac{\varepsilon_t^2}{\sigma_t^2}\right] \tag{7}$$

The parameters are estimated following:

$$\hat{\theta}_{ML} = \arg\max_{\theta} \mathcal{L}(\theta | r_t)$$

Thanks to these 2 theories we are then able to forecast the volatility of a chosen asset.

As for the third command which prices an American option, we had to rely on the binomial option pricing model. This method uses a discrete time model of the of the variation of the price of the underlying asset.This model traces the evolution in price of the underlying by a binomial tree. Each node of the tree represents a possible price of the asset at a certain time up to the expiration date such as on the figure below.

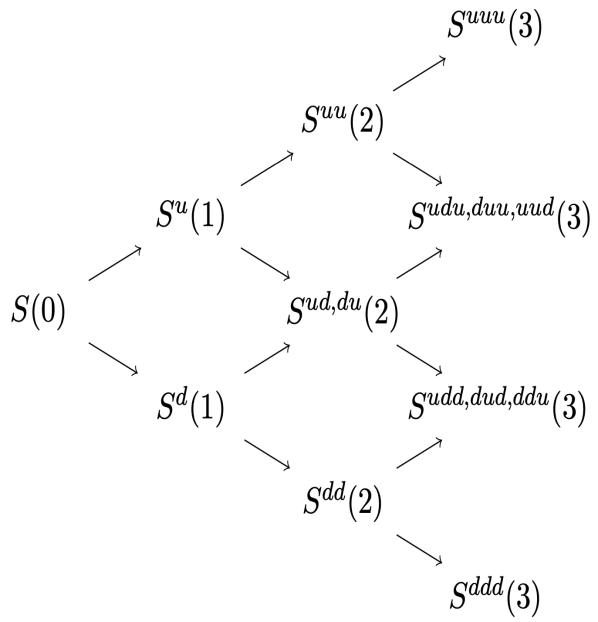

Fig. 1. Binomial Tree

At each node, we assume that the price will either go up or down by a specific factor (respectively u and d). Let S be the current price, then the price in the next period will be either $S_u = S * u$ or $S_d = S * d$. In order to calculate these factors, we need:

- σ, the underlying asset's volatility
- t , the time duration between two nodes (in years)

Using the fact that the variance of the log of the price is $\sigma^2 t$, we get:

$$u = e^{\sigma\sqrt{t}}$$

$$d = e^{-\sigma\sqrt{t}}$$

$$= \frac{1}{u}$$

From this we can then set the price value of the asset at any node:

$$S_n = S_0 * u^n$$

where n is the number of up minus the number of down ticks.

IV. Implementation of the algorithm

We have used the following packages in order to make the program efficient:

- Math
- Random
- Datetime
- Matplotlib
- Mplfinance
- Numpy
- Pandas
- Yfinance
- Arch
- ExcelWriter from Pandas
- Pandas datareader
- Scipy.stats
- Yahoo fin

In order to implement all the theories above so that we get a working program, we had to separate the code into multiple parts. The main part is a while loop which will react a certain way depending of what the user wants. The while loop then contains branches of if, $elifs$ and $else$. This is how it is implemented:

```
while True:
    today = date.today()
    user = input("Please enter a new command('help' for a summary of the commands): ")
    # guide him
    if user == "help":...
    elif user == "1":...
    elif user == '2':...

    elif user == '3':...

    elif user == '4':...

    elif user == '5':...

    elif user == "exit":...

    else:
        print('You typed on a wrong command.Please retry')
```

Fig. 2. while loop implementation

First of all, when the loop starts, it will ask the user for a command. The user has a certain set of choices. As we do not expect from any user that would not be familiar with the app the first time they use them, there is a $'help'$ command. This command outputs the list of all possible commands that are available to the user. In total, there are 6 commands.

1) Estimate the price of an European option.
2) Estimate the expected future forecast of a volatility of an asset
3) A screen option selector
4) Estimate the price of an American option
5) Visualize the price movement of an asset during a certain period in time
6) exit the app

which looks like this:

```
Please enter a new command('help' for a summary of the commands): help
================================================================================
To estimate an European option type 1:
================================================================================
To estimate the expected future forecast of a volatility of a stock/commodity type 2:
================================================================================
To use a screen option selector type 3:
================================================================================
To estimate the price of an American Option type 4:
================================================================================
To Visualize a stock/commodity movement type 5:
================================================================================
to leave the app, simply type 'exit'
```

Fig. 3. Commands

When the user chooses the first command, he will be able to select a certain amount of details which will let the app price the chosen option. To prevent the app from crashing if the user enters a value which is not valid, each question is embedded in a while loop. This means that for as long as the user types a wrong value, the app will repeat the question and explain what the user did wrong. Once the user has entered all the values required to price correctly the option, the program will begin calculating. The calculation of the European options is then implemented into two classes called $EuropeanPut$ and $EuropeanCall$. Then each class will have functions in order to calculate the delta as well as the price of this option. Below is the skeleton of the European call option class.

```
class EuropeanCall:

    def call_delta(
            self, asset_price, asset_volatility, strike_price,
            time_to_expiration, risk_free_rate, q
    ):...

    def call_price(
            self, asset_price, asset_volatility, strike_price,
            time_to_expiration, risk_free_rate, q
    ):...

    def __init__(
            self, asset_price, asset_volatility, strike_price,
            time_to_expiration, risk_free_rate, q
    ):...
```

Fig. 4. European option class

Finally, the program will print a graph of the Monte-Carlo simulations (only if it was asked by the user), as well as the average price for the option, the maximum and minimum price and the probability of exercise.

For the forecast of volatility command, it is pretty much done the same way as above. First of all, the program asks the user for the asset he wishes, which will automatically download data of this asset from Yahoo finance and print a GARCH analysis to the terminal. It will then ask for all the other values which the user can chose. As for the first command all these questions are embedded into a loop which breaks only when the user enters a valid input for that particular question and displays an error message otherwise. Once all the values are entered by the user, the program uses

the following class: $OptionTools()$. Here is the skeleton of this class which shows every calculation it is able to do:

```python
class OptionTools:

    def __init__(self):
        pass

    # Return annualized remaining time to maturity and days to maturity for simulations
    def compute_time_to_expiration(self, Y, M, D):...

    # Testing the model before implementation,in the project we don't need the case as we are using real datas
    def generate_random_option(self, n, call=True):...

    # Simulate options, returns a set of OptionSimulations
    def simulate_calls(self, n_time_steps, n_options, strike_price, initial_asset_price, drift, delta_t,
                       asset_volatility, risk_free_rate, time_to_expiration, q):...

    def simulate_puts(self, n_time_steps, n_options, strike_price, initial_asset_price, drift, delta_t,
                      asset_volatility, risk_free_rate, time_to_expiration, q):...

    # Takes a set of option simulations returns a vector output of average option price at end of option life, max
    # simulated price, initial simulated price, and min simulated price
    def simulation_analysis(self, option_simulations):...

    # Returns the probability of exerise after simulation, takes set of option simulations
    def probability_of_exercise_calls(self, option_simulations, call=True):...

    def probability_of_exercise_puts(self, option_simulations, call=False):...

    # Takes an option simulation set, chart each sample path and the respective variable
    def aggregate_chart_option_simulation(self, option_simulations, asset_prices, option_prices, option_deltas):...
```

Fig. 5. Option tools class

This class has a few methods which calculate values that are useful in order to transform the input into usable values. Here typically, the method $compute\ time\ to\ expiration()$ will take the date input from the user and turn it into a number of days. The program will output a graph of the volatility in two parts. First it will show the evolution of the volatility from past data which was downloaded from the internet. Then it will continue the graph with the forecast up to the date chosen by the user. Finally, the program will print the expected conditional volatility for the chosen date.

When the user asks for the third command, the screening, just like for the others there is a few inputs asked to the user which are also embedded into loops in order to prevent the app to crash and to help understand the user what kind of input he is supposed to enter. When all this is done, the program will start downloading data from Yahoo finance and print the probability of exercise of a call as well as a put which are computed by the methods $probability\ of\ exercise\ call()$ and $probability\ of\ exercise\ put()$ which are into the $OptionTools$ class shown above. When the program is done downloading the data and printing out the probabilities of exercising, it will then create an excel file which will be named: $ScreenOutput.xlsx$ which contains a selection of 8 stocks with their probabilities of exercising the call and put, as well as the strike price of each asset. The calculations for this is made using the classes $Screener\ EU$ or $Screener\ US$ depending on the choice of the user

	Stock	Probability of executing a call	Probability of executing a put	Strike Price
0	AAPL	0	1	478.7073
1	BA	0.76	0.16	107.9638
2	HD	0.12	0.92	290.2054
3	INTC	0	0.98	82.05905
4	MSFT	0	1	242.93
5	NKE	0.06	0.9	108.7236
6	WMT	0.06	0.94	145.5601
7	XOM	0.72	0.16	38.13989

Fig. 6. Screenoutput.xlsx using the Black-Sholes Model

This excel spreadsheet will be saved in the same folder where the code script is.

If the user wants to use the command 4, as always, he is going to be asked for a few questions which are embedded into loops to prevent the app from crashing. When all the questioning is done, the program will use the $compute\ time\ to\ expiration()$ method from the class $OptionTools()$ to calculate the number of days to maturity. It will then call the method $binomial\ tree()$, from the class $American\ Option\ Pricing$, which takes as input the variables of the option (The date today and maturity chosen, the dividend yield, the current price, the volatility, the risk free rate and the strike price).

```python
# American option pricing using binomial trees:
class American_Option_Pricing:

    def binomial_tree_call_draw(self, N, q, T, S0, sigma, r, K, call=True):...

    def binomial_tree_put_draw(self, N, q, T, S0, sigma, r, K, call=False):...

    def __init__(self, N, q, T, S0, sigma, r, K):

        self.N = N
        self.q = q
        self.T = T
        self.S0 = S0
        self.sigma = sigma
        self.r = r
        self.K = K
        self.binomial_tree_Pricing_call_d = self.binomial_tree_call_draw(N, q, T, S0, sigma, r, K)
        self.binomial_tree_Pricing_put_d = self.binomial_tree_put_draw(N, q, T, S0, sigma, r, K)
```

Fig. 7. Binomial tree function

This function will then create the binomial tree for the chosen asset using a nested for loop. When it is done calculating, the program then prints the price of the American option as well as the binomial tree for the asset and the binomial tree for the option.

For the fifth command, the user will go through the loops of questions in the exact manner as he has done above. The program will then load the data from Yahoo finance of the selected asset. Then the methods $percentB\ belowzero()$ and $percentB\ abovezero()$ from the class $technical\ chart$ will be called in order to created the signals of whether the user should buy or sell.

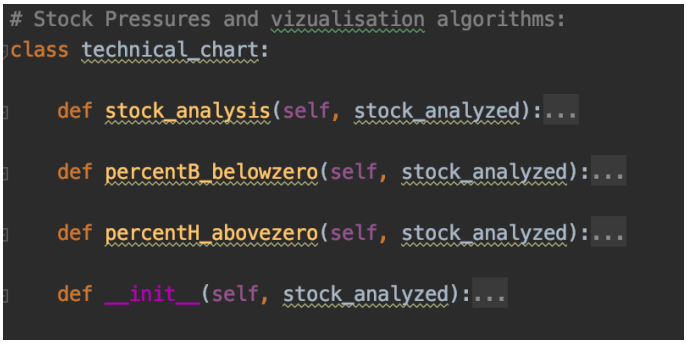

Fig. 8. Class containing the function creating the signal to buy/sell

Finally, the program will display a graph showing the price evolution of the said asset as well as the low Bollinger Band and the high Bollinger band which tells you if you should sell or buy.

Finally, we come to the last option of the while loop, which is called $exit$. By typing exit, the program will execute the branch for this condition and it will break the loop which will result into the program stopping.

In summary, we have a while loop which is making the program run and lets the user use certain commands. Then the commands will lead to calculations which are all computed from classes which helps the program run faster than with simple functions. The last possible command $exit$ will break the loop and stop the program from running.

V. Maintaining the code

The code has been written in a rather intuitive way, so that it will be relatively easy to understand how it works for someone who has not worked on it and wants to use it. Someone reading it should also be able to understand what each part of the code does without too much trouble.

Here is the code skeleton:

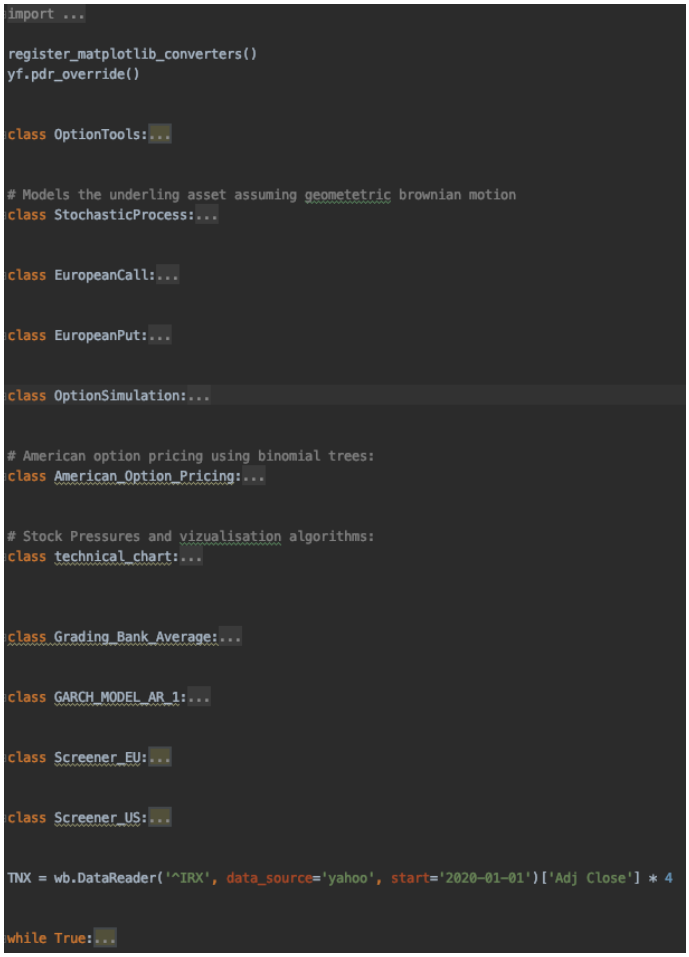

Fig. 9. Code skeleton

As we can see, the code was made in an efficient way. The functions used to do the calculations are all embedded into their respective classes. This allows the code to run faster than if it was coded only using functions.

The program has been uploaded onto GitHub. This will allow it to be open source and will be available to anyone who wishes to use it. It will also allow the community to maintain the code.

Github Link:
https://github.com/EM51641/SuperOption.git

In order to make sure every class that has been implemented into the project, we have used unit testing. Every class has been tested individually so that we can validate that every piece of code performs as intended. In order to do so, the classes have been written into a separate file and then tested in order to make sure that it worked as intended. When we made sure that the class was doing what it was supposed to do, it then was implemented into the project and adapted in order to make it work correctly. Doing this unit testing has helped us prevent bugs and to make sure that we were obtaining the correct results.

VI. Results

The technical chart classes shows impressive results with a work done in 0.025 seconds using more than 2520 daily data. OptionSimulation takes time with 12.5 seconds in average(for a 3.5 months horizon which means approximately 100 forecasts per simulation), but it can be explained due to it's expensive computations (we simulate 500 Brownian motions, compute the calls/puts, the different deltas according to the option type and evaluate the maximal, minimal, initial price of the option. The dividend yield takes a lot of time to import: 3.56 seconds even if the yf.pdroverride() hijacks the process (the importation of data from yahoo would be a lot slower without it). The simple Put Call computation takes only 0.0004 seconds to work, this is spectacularly fast. The Garch-Arch class takes 0.20 seconds (for a 3 months forecast) which is also good as we are using more than 10 years of daily data and forecasting conditional volatility. The binomial pricing model class takes 0.00024 seconds in average to work using a 7 open month (without taking week-ends into account, which means we are using 21 days per month) which is a good result. The importation of different banks recommendations takes 3.35 seconds, but the Grade class, which averages all the grades, works in only 0.00013 seconds. Importing the whole NASDAQ stocks components list which contains about 1100 tickers from internet takes 3.77 seconds to import. The screening method can be fast if we use the American pricing models without the dividend yield, it takes 1 second to import data and treat them. If we import dividend yields, it increases to 3.5 seconds. For the European options model, it takes 9.41 seconds to treat the data, but here we reduced the simulations to 100 in order to keep the speed of the program bearable. When we remove the dividend, the speed of execution is reduced to 4 seconds which is approximately equal to the speed of the binomial pricing class.

The importation measurements can be improved as the tests were executed with a connection of 8MB.Alternatively the ideal for these expensive computations is to make them run on a GPU which are a lot more efficient for computations than CPU (graphics in games are only a question of matrix multiplications after all).Solutions like that can today be implemented.For example some python packages can permit to run your code on the computer GPU.Another solution can be to rent GPU server at a reasonable price(We can find 5*4GB GPU servers at 50$ per month).

If we had to redo this project, there are a few things we would have tried to do differently. One of them is taking some time in order to combine the python code with c++ in order to make the calculations faster thanks to parallelism. It would have also been interesting to implement other models such as LTSM (Long short-term memory), which is an artificial recurrent neural network architecture that is used in deep learning. It would have been useful to make predictions.

VII. Conclusion

To conclude, we have managed to create a functioning program able to price European as well as American options among some other features. It was not an easy task and we have had to learn a lot along the way but it was worth it as we have acquired a useful set of skill for the future. This project was really interesting to do even though we had to rework the code several times to make it a bit more efficient.

VIII. References

Jondeau, Eric. "Modeling Volatility: GARCH Models and Extension". Empirical Methods in Finance, University of Lausanne, Ecublens, April 27, 2020.

Jondeau, Eric.Ser-Huan Poon.Michael Rockinger "Financial Modeling Under Non-Gaussian Distribution". Springer Finance.

Rockinger, Michael. Fixed income and credit risk, University of Lausanne, Ecublens, Spring semester, 2020.