Report Group 2 Assignment

Giacomo Pasquetto (14129A), Matteo Rizzardi (14107A), Riccardo Tanchis (11203A)

Presentation and analysis of the dataset

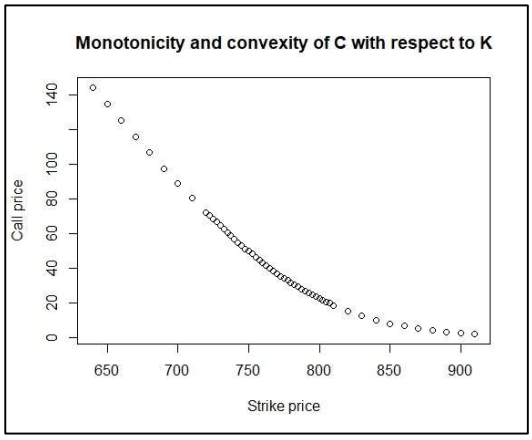
Our dataset is composed of 58 call options: some in the money, some out of the money and some at the money, with the same underlying asset (GPR_2) and same maturity (26 days). We noticed that the column "option type" is composed only of capital "C"; this detail allowed for easier coding.

The first step in our assignment is to determine the Log-Moneyness k of our option prices. We kept only the options that lied in the interval $\boldsymbol{k \in [-0.2, 0.2]}$, resulting in removing the first three calls of the dataset because they were deeply into the money ($k > 0.2$).

It is important to notice that the continuously compounded interest rate is on a yearly basis while our time to maturity is 26 days, therefore we have created the variable "daily_interest", which is the daily interest rate.

Point 1

We, then, cleaned our dataset removing options that did not satisfy "**No-arbitrage/Merton's bounds**" $S_t \geq C_t(S_t, K, T) \geq max\{S_t - K*D(t, T);0\}$. Before moving to point 2, we also extracted from the dataset those options that did not satisfy monotonicity and convexity constraints relying on the graphical approach.



The call prices look monotonically decreasing in K and they are also a convex function of the strike price. From the graph, it looks like there is no arbitrage opportunity. After all computations, the number of the eliminated call options is reported in the following output. Notice that **there were no excluded call options by the Merton's constraints.**

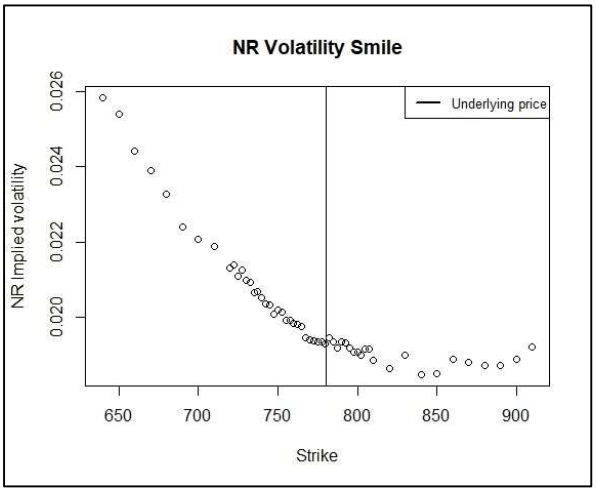
```
$`Total number of eliminated calls
[1] 3

$`Percentage of calls eliminated`
[1] 0.05172414

$`Number of remaining calls`
[1] 55

$`Percentage of remaining calls`
[1] 0.9482759
```

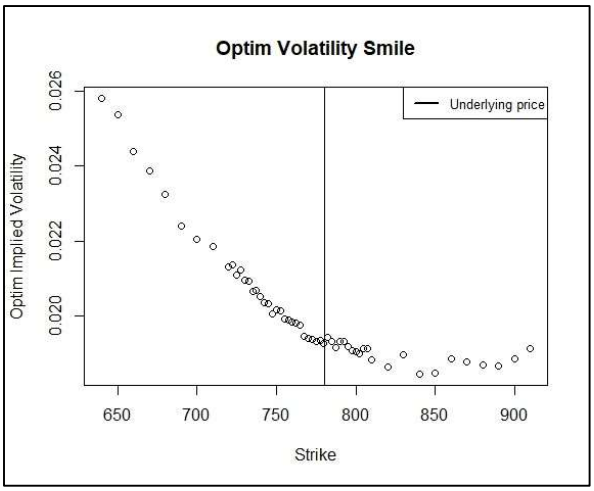
Point 2

In this point we utilized the cleaned dataset to solve two problems:

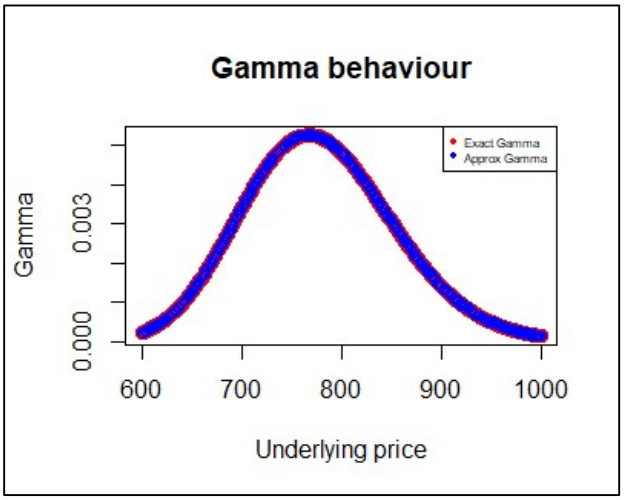
- Initially, we obtained the implied volatilities using the **Newton-Raphson** algorithm. Then, we compared the results with the implied volatilities generated with the R function **optim**. For the latter we initialized it with a lower bound of 0.001 and an upper bound of 1. However, we had to change the lower bound because numerical computation gave us the same result for almost every option: around the lower bound. Thus, we restricted the possible values of sigma between 0.01 and 1 and we found the solutions coinciding with the ones obtained with the **Newton-Raphson** method. In fact, the Mean Squared Error between the implied volatilities appeared very small and both the plots look more like a volatility skew.

```
> Optim_NR_MSE
[1] 2.560506e-10
```





- Secondly, using the volatility nearest at-the-money level, we compared the behaviour of the approximated Gamma with the exact Gamma for varying value of the underlying asset S_0. The former was obtained with finite difference method**,** using **the second order central finite difference method**, while the latter was calculated with the theoretical formula of Gamma. As we can see from the following plot, they show an identical behaviour as the underlying price changes.



Point 3

In the third part, we had to calibrate the value of sigma for the Black and Scholes Model, based on the Mean Squared Error. First, we defined the "**BS_mkt_MSE**" function, which represents the MSE between the Black and Scholes price and the market price. Subsequently, we applied the R function "**Optim**" to the last function to find the best value for sigma (with an initial guess of 0.10), which is:

```
> calibrated_sigma
[1] 0.01972289
```

With this value, we had to price a Bear Strategy for two Call Options with given strikes $K_1=S_0*0.975$ and $K_2=S_0*1.025$, with a time to maturity of 60 days. This strategy consists in a long position in the call option with the higher strike (K_2) and a short position in the call option with the lower one (K_1). The final payoff of this strategy is always non-positive, thus, in a no-arbitrage context, the holder must be compensated with a positive amount of money at time t_0.

Condition	Payoff at Time T
$S_t \leq K_1$	0
$K_1 < S_t \leq K_2$	$K_1 - S_T$
$S_t > K_2$	$K_1 - K_2$

Using the **first fundamental theorem of asset pricing**, at time t_0, we have:

$$C_{t0}(S, K_2, T) - C_{t0}(S, K_1, T) \leq 0$$

Meaning that we receive money at time t_0 to hold the portfolio. In our case, we pay:

```
> Bear_strat_price
[1] -18.64994
```

Meaning that we receive 18.64994$ at time t_0.

Point 4

In this final point, we had to compute the Monte Carlo price for a **Floating Arithmetic Asian Call Option,** with time to maturity 30 days and the final payoff defined as: $max\left\{\frac{\sum_{j=0}^{N} S_{t_j}}{N+1} - S_{t_N}; 0\right\}$.

To do so, we started by simulating 500 underlying asset's price dynamics using the **Euler Simulation Scheme** of a Geometric Brownian Motion, considering a time-step $\Delta=5$ days, the calibrated σ, and a 5-day average rate of return of the asset $\mu=0{,}001$.
This means that one trajectory is:

Day 0	Day 5	Day 10	Day 15	Day 20	Day 25	Day 30
S_0	S_1	S_2	S_3	S_4	S_5	S_6

We did this by creating the function "**MC_price_GBM**". This function also computes the mean of the simulated underlying prices and takes the difference with the last simulated prices. Finally, the function computes the final discounted payoffs of the 500 simulated options: these are the prices of the options for each simulated underlying asset.

By taking their mean we obtain the Monte Carlo price of the option.
The final step was then to add the 95% percent confidence interval.

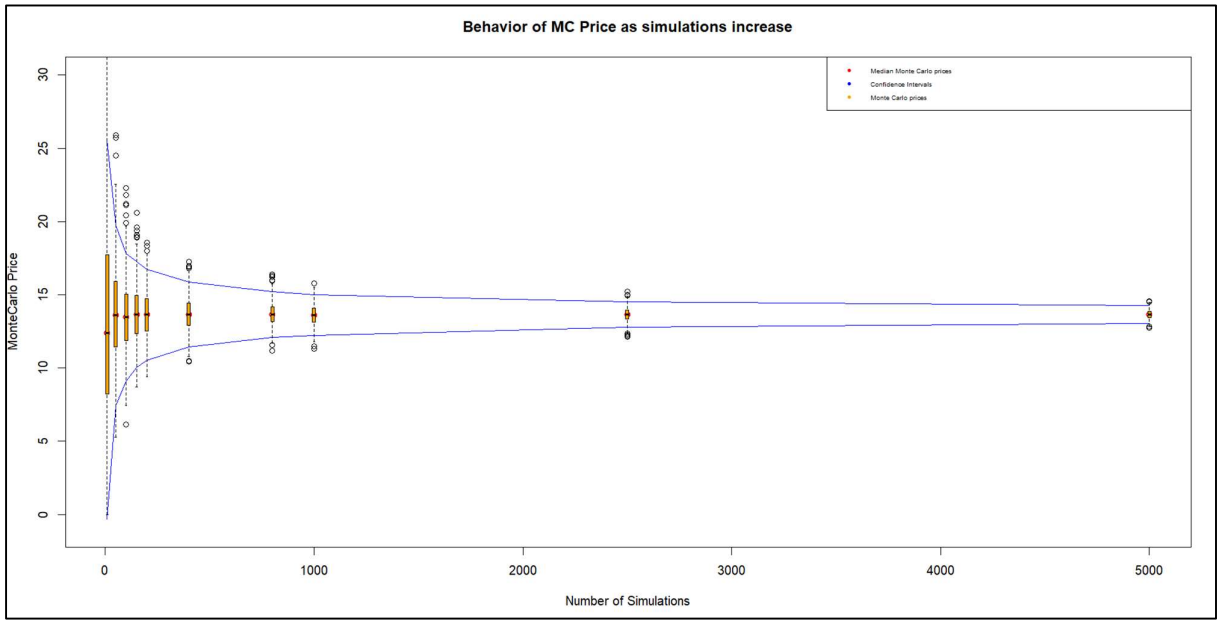
However, if we compute "**MC_price_GBM**" only once, and the number of simulations is low, then we have a highly varying Monte Carlo price. That is, we have a very wide confidence interval, thus, by running the code multiple times, the price would oscillate a lot. Of course, as the number of simulations increases, the problem is fixed. However, this becomes a problem when we want to analyse the behaviour of the Monte Carlo price as the number of simulations increases (we are forced to show the Monte Carlo price with a low number of simulations), because the Monte Carlo prices calculated with a low number of simulations will be different every time the code is run. Therefore, to fix the variability in these Monte Carlo prices, we have also created the function "**M_simulations_MC_prices**", which computes the "**MC_price_GBM**" function M times, and takes the median value for the Lower Bound, for the Monte Carlo price, and for the Upper Bound. At the end of the process, the output is the following:

```
> MC_price_interval_with_Mreps
           Lower Bound Montecarlo Price Upper Bound
nsim:  500    11.70866         13.67139    15.66009
```

In the final part, we have implemented the function "**MC_behaviour**", which takes as an input a vector containing a different number of Monte Carlo simulations instead of a single value (500 for the previous part). In this way, we can plot the Monte Carlo prices as a function of the varying number of simulations. Just like the previous step, running the code just one time brought us to obtain different values of Monte Carlo prices for a low number of simulations, giving a different plot each time it has been run. The implementation of the function "**M_simulations_MC_behaviour_plot**" aims to fix this problem, in fact, it computes the function "**MC_behaviour**" M times, with the same approach of the function "**M_simulations_MC_prices**", giving us more robust results. The outputs of this function are the matrix and the plot reported below. The latter contains the following arguments:

- Median Monte Carlo prices (red dots)
- Confidence intervals (blue lines)
- Monte Carlo prices (yellow boxplots)
- Outlier prices (white dots)

```
> mat
            Lower Bound Montecarlo Price Upper Bound
nsim:  10     -0.3211834         12.37544    25.55290
nsim:  50      7.4469269         13.59723    19.73743
nsim:  100     9.0922354         13.46210    17.80885
nsim:  150    10.0455102         13.65763    17.26710
nsim:  200    10.5398508         13.64630    16.73172
nsim:  400    11.4533970         13.64826    15.85204
nsim:  800    12.1088683         13.66074    15.21738
nsim:  1000   12.2313304         13.62340    15.00251
nsim:  2500   12.7785471         13.66080    14.54037
nsim:  5000   13.0332524         13.65772    14.28152
```



In conclusion, we can observe that, as the number of simulations increases, the confidence interval of the Monte Carlo price gets narrower and narrower. This is because the Monte Carlo price we estimated has a deviation from the true expected value E[g(x)] of order $1/\sqrt{n}$, where n is the number of simulations.