Option Pricing Model using Black-Scholes Model

Simulation of a Geometric Brownian Motion

Ester Hlavnova

Main Idea and Aim of the Program

Option Pricing Calculator is one of the fundamental tools in the field of quantitative finance. For my project, I implemented the Black-Scholes Model – a widely used formula for estimating the price of European options, i.e. finding the value of puts and calls. Implementing this model challenged my Java skills in different areas of programming.

On one hand, the main part consists of an Option Pricing Model that is divided into two subsections, in which the user is prompted to select between computation or a simulation. Both choices offer three buttons. For example, in the computational part, the user can access an option pricing calculator as well as directly calculate implied volatility or the Greeks. Furthermore, as for the part with simulation, one can display a graph of volatility smile or a payoff diagram, as well as simulate Geometric Brownian Motion. All options are user-friendly and based on an interaction with a user, where one inputs the values of a parameter – in most of the cases parameters are deterministic for calls and puts – and the program calculates or simulates the desired output based on the selected choices. This part of the program challenged my Java skills in creating classes, inner-classes, basic data structures (mostly lists and arrays), and functions. In addition, it was very exciting to conduct a simulation in Java, because I had to implement a generic graph class to visualize series and diagrams.

On the other hand, I also opted to include an introduction to options. User is offered a basic explanation of what option contracts are, and can eventually test himself with a quiz that is made user-friendly and interactive. In this part, I was excited to exploit Java GUI. I implemented JButtons, JRadioButtons, JTextFields, JComboBox and others, as well as different layouts for frames.

My program in detail

I have decided that the best way to explain what my program does is to follow the program outputs/GUI windows, and stepwise comment on the choice buttons, outcomes and conceptual explanations about quantitative finance. In the introductory window of my program, user can decide to continue with an 'Introduction to Calls & Puts', or directly continue to the 'Option Pricing Model'. I will first describe the Introduction button in Part A and then Option Pricing Model in Part B.

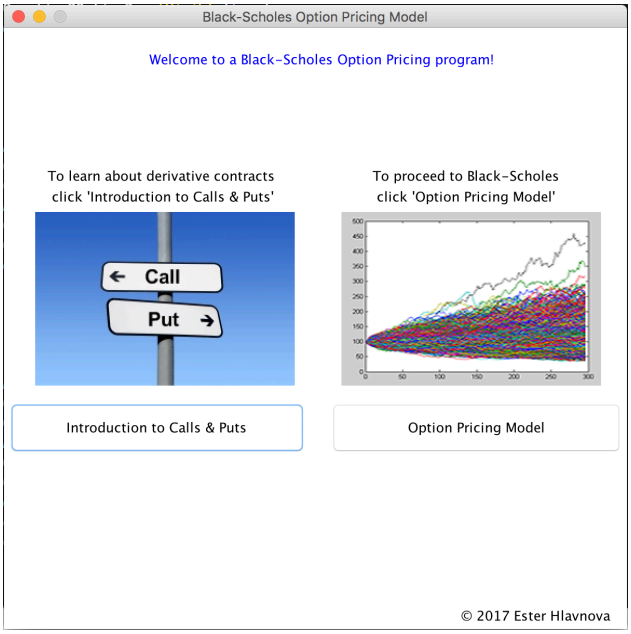

Figure 1: Introductory welcome window with two choices

PART A – Introduction to Calls & Puts

The reason I opted to include the Introduction button was to make the program more understandable and beneficial for somebody without a previous exposure to derivative contracts. This part consists of several slides that I prepared in advance using PowerPoint, and imported them as pictures into the JFrame and implemented buttons for navigation (Figure 2).

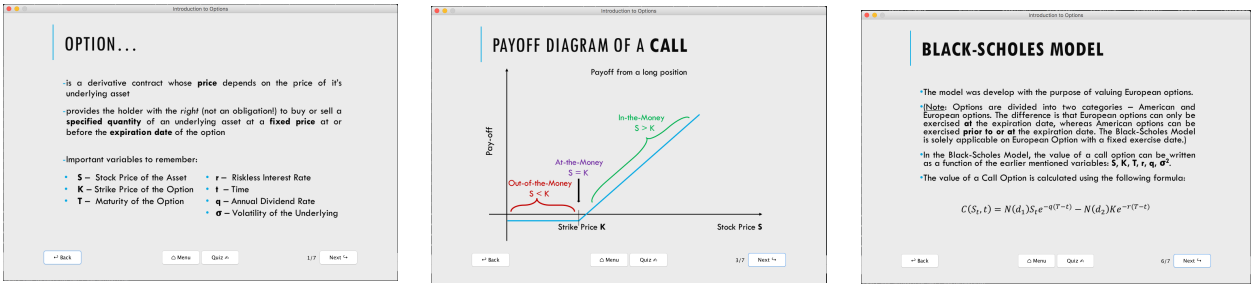

Figure 2: 'Introduction to Options' Window. Example of slides that I used to explain concept of calls and puts.

I prepared the slides in PowerPoint because it was more flexible to construct diagrams or mathematical formulas. While these slides are supposed to introduce the user with some ideas of finance, I further implemented a button 'Quiz' that has 4 parts with questions on the relevant topic.

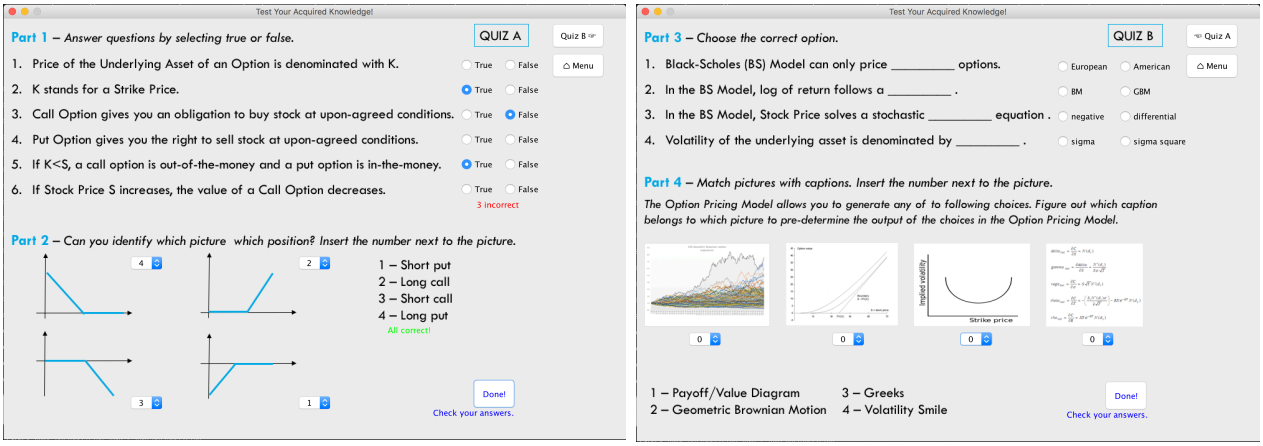

Figure 3: Quiz Window. Offers the user a two-section quiz – Quiz A and Quiz B, with 4 parts in total.

The quiz has 4 parts and the questions vary in type, for example true/false questions in part 1, choice questions in part 3, and matching pictures in part 2 and 4. For this purpose, I installed different types of buttons and used JRadioButton in Part 1 and Part 3, and JComboBox in Part 2 and Part 4. I tried to make the questions interesting and useful. However, they might be a little bit too challenging because the introductory slides do not cover everything in enough depth, however, for time reasons I didn't explain it better (I would have liked to but it isn't the point of the course), but I hope it was still useful and that the user learned some basics.

One thing I enjoyed about the quiz was making it interactive and implementing the 'Done' button that checks user's answers. You can see in Figure 3 on the picture on the left that has a little red label '3 incorrect' that updates every time user changes answers and clicks Done. Similarly, it does the same for the Part 2 questions where I selected all options correctly to demonstrate the little green label 'All correct!' which appears once the answers are correct.

User can move between Quiz A and Quiz B by clicking on the right button, as well as go back to main menu. One thing I would like to point out and clarify is the way I decided to code the GUI in the case of this introduction part of my program. Rather than using specific layouts (such as e.g. Border Layout or Flow Layout), I chose to compute and set the location myself manually because I preferred to make the frames pretty and esthetic. The way I handled adding buttons on the top of the imported picture was that I set the layout of the frame to null and added the buttons directly to the image. I am aware that this is not the most efficient way, but the complexity of the program wasn't so high that it was impossible. Also, I used specific layouts from the packages later in the simulation part.

PART B – Option Pricing Model

I would like to continue with Option Pricing Model (see Figure 1 for the introductory window). This is the main part of my program that has several subclasses implemented to compute the rather advanced mathematical concepts. Selecting the button 'Option Pricing Model' takes the user to a new window where he can select from multiple choices – three for each simulation and computation (see Figure 4).

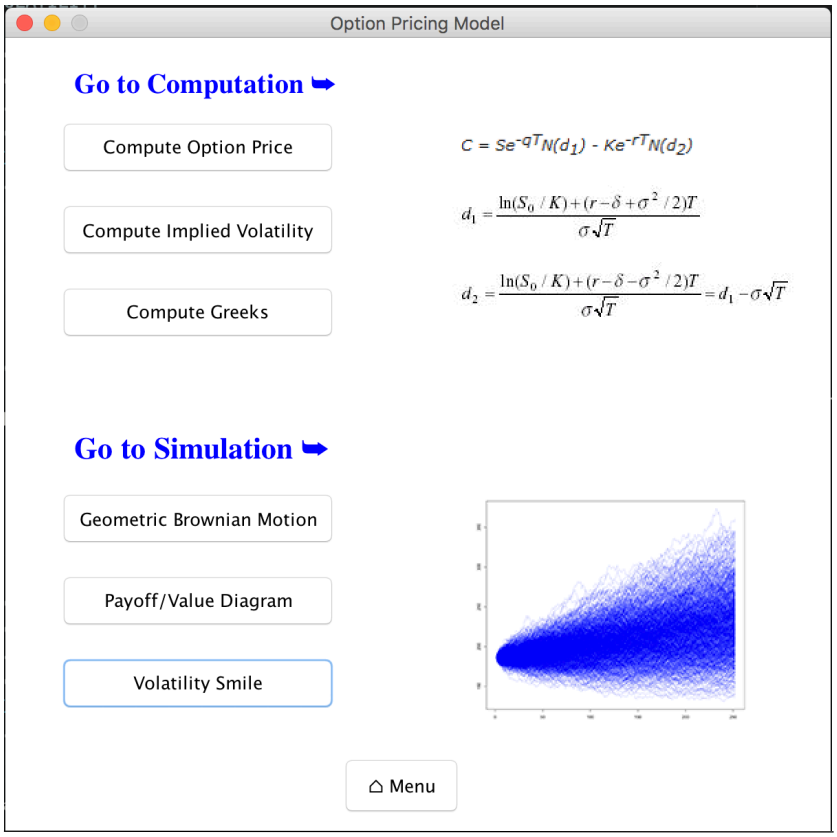

Figure 4: Option Pricing Model Window

○ GO TO COMPUTATION

In the part 'Computation', the user has the choice to compute Option Price, Implied Volatility and the Greeks.

a) Computing the Option Price

To estimate the value of an option, we apply the Black-Scholes formula. This formula allows to express the value of an option as a function of the following variables:

- **S** Stock price/current value of the underlying asset
- **K** Strike Price of the option

- **t** Time
- **T** Maturity of the option
- **r** Riskless interest rate
- **q** Annual dividend rate
- **σ** Volatility of the underlying asset

The Black-Scholes Model uses the assumption that the underlying stock price follows Geometric Brownian Motion. This indicates that S (stock price) solves the stochastic differential equation:

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

with μ being the drift, σ the volatility and W_t a Brownian motion.

The formula for calculating the value of the Call Option is as follows:

$$C(S_t, t) = N(d_1)S_t e^{-q(T-t)} - N(d_2)Ke^{-r(T-t)}$$

where $\mathrm{N}(\mathrm{x})$ denote the standard normal cumulative distribution function:

$$\mathrm{N}(\mathrm{x}) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\mathrm{x}} \mathrm{e}^{-\frac{\mathrm{z}^2}{2}} \mathrm{dz}$$

and $\mathrm{N}'(\mathrm{x})$ denote the standard normal probability density function:

$$\mathrm{N}'(\mathrm{x}) = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{\mathrm{x}^2}{2}}$$

The variables d1 and d2 are as follows:

$$d_1 = \frac{1}{\sigma\sqrt{T-t}} \left[\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t) \right] \; and \; d_2 = \; d_1 - \sigma\sqrt{T-t}$$

The price of a Put option can be deduced from the above formula using the Put-Call Parity:

$$P(S_t, t) = N(-d_2)Ke^{-r(T-t)} - N(-d_1)S_t e^{-q(T-t)}$$

User chooses a call or a put and then will be required to input in a text field the value of the following variables: S, K, t, T, r, q, **σ.** The calculation will be done using the Black-Scholes formula

as shown above. The output will be the price of the option in the same currency as the one specified with the stock price S. In addition, you can see that the program works directly for both call (upper part) and puts (lower part), and the user inputs the values of the parameters accordingly.

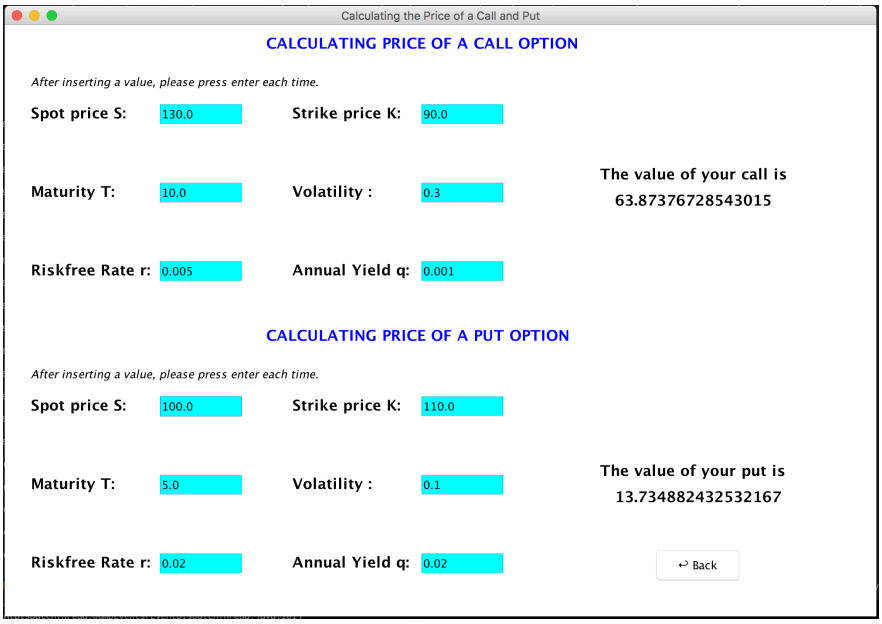

Figure 5: Calculating the Price of a Call and a Put. The user updates parameters by choice.

b) Computing the Implied Volatility

User inputs following variables - S, K, t, T, r, q, P, where P is the price of the option as quoted on the market. In the past 10 years, the implied volatility became an alternative to the historical volatility, which is of great usage in the financial institutions.

The idea behind implied volatility is that markets are giving us the spot price of options and therefore the volatility that is implied by this quotation. Indeed, the Black-Scholes formula is a function of sigma: $C(\sigma)$ and $P(\sigma)$. Hence, we can compute the value of a call for a given volatility and try to match the option spot price, thus looking for the implied volatility σ^* that satisfies $C(\sigma^*, K) - C_{market}(K) = 0$. This can be solved using root solving algorithms. In my implementation, I coded three methods: Newton-Raphson, Secant method and Regula Falsi. As for the simplicity of the Newton-Raphson, I chose to use this one because it only requires one initialization value, set by default to 0.05. I also use a maximum number of iterations of 300 and a convergence threshold $\epsilon = 10^{-3}$.[1]

[1] More information on the algorithms can be find on https://en.wikipedia.org/wiki/Newton%27s_method, https://en.wikipedia.org/wiki/Secant_method and https://en.wikipedia.org/wiki/False_position_method.

It is important to clarify several points to understand the behavior of the implied volatility calculator. First, a solution is not always possible. For instance, if a call spot price is extremely high but the spread between the spot price and strike price of the asset is extremely low, then there exists no solution. For this reason, the calculator might answer Infinity or –Infinity, which means that the algorithm diverged by absence of solution. Also, root finding algorithms do not guarantee finding a solution even if a solution exists. Each algorithm has its own flaws and circumstances in which it can diverge (mostly due to extreme value of estimated derivative). A divergence would also be outputted by the calculator as ± Infinity. However, in the context of normal options (not exotic options), the price has a quasi linear relationship to the strike price *K,* hence the algorithm converges almost always if a solution exists.

Note: We can see that all the parameters remained similar as in the above a) option. We only exchanged the input/output variables, and while volatility became an output, option price was taken as a input parameter.

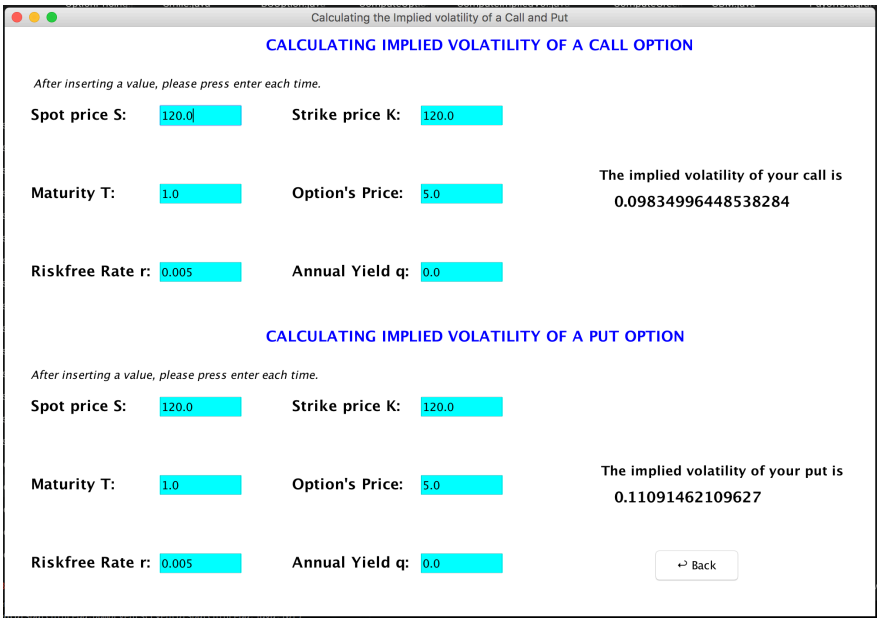

Figure 6: Calculating the Implied Volatility. The user updates parameters by choice.

c) Computing the Greeks

The Greeks are the partial derivatives of the option price according to the Black-Scholes model regarding each variables taking into account. Obviously, the formula are generally different for Calls and put, here we give the formulas for call options, but more can be found here: https://en.wikipedia.org/wiki/Greeks_(finance)#Formulas_for_European_option_Greeks.

We first introduce for practical convenience the time to maturity as $\tau = T - t$.

Also, Φ denotes the standard cumulative distribution function, and ϕ its derivative, the standard probability distribution function. Hence:

$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{z^2}{2}} dz$ and $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$.

With C the price of a call option as follows, $C(S_t, t) = N(d_1)S_t e^{-q(T-t)} - N(d_2)Ke^{-r(T-t)}$, we have:

- $\Delta = \frac{\partial C}{\partial S} = Se^{-q\tau}\Phi(d_1) - e^{-r\tau}K\Phi(d_2)$, the derivative regarding the stock price.
- $\nu = \frac{\partial C}{\partial \sigma} = Se^{-q\tau}\phi(d_1)\sqrt{\tau}$, the derivative regarding the volatility.
- $\Psi = \frac{\partial C}{\partial q} = -Se^{-q\tau}\left(\frac{\sqrt{\tau}}{\sigma}\phi(d_1) + \tau\phi(-d_1)\right) + Ke^{-r\tau}\phi(d_2)\frac{\sqrt{\tau}}{\sigma}$, the derivative regarding the annual yield.
- $\Theta = \frac{\partial C}{\partial \tau} = qSe^{-q\tau}\Phi(d_1) - rKe^{-r\tau}\Phi(d_2) - \frac{S\phi(d_1)\sigma}{2\sqrt{\tau}}$, the derivative regarding the time (also called time decay).
- $\rho = \frac{\partial C}{\partial r} = K\tau e^{-r\tau}\Phi(d_2)$, the derivative regarding the risk-free rate.
- $\Gamma = \frac{\partial^2 C}{\partial S^2} = \frac{\partial \Delta}{\partial S} = e^{-q\tau}\frac{\phi(d_1)}{S\sigma\sqrt{\tau}}$, the second derivative regarding the stock price.
- Volga $= \frac{\partial^2 C}{\partial \sigma^2} = \frac{\partial \nu}{\partial \sigma} = \nu\frac{d_1 d_2}{\sigma}$, the second derivative regarding the volatility.

To give an example of the derivation of such formula, we can derive the equation of Ψ, that is not easily accessible online:

First of all, $\frac{\partial d_1}{\partial q} = \frac{-\tau}{\sigma\sqrt{\tau}} = -\frac{\sqrt{\tau}}{\sigma}$ and $\frac{\partial d_2}{\partial q} = \frac{\partial d_1}{\partial q} = -\frac{\sqrt{\tau}}{\sigma}$.

Also, $\frac{\partial \Phi(X)}{\partial X} = \phi(X)$ so $\frac{\partial \Phi}{\partial q} = \frac{\partial \Phi}{\partial X}\frac{\partial X}{\partial q} = \phi(X)\frac{\partial X}{\partial q}$.

Thus, $$\Psi_c = \frac{\partial C}{\partial q} = -S\tau e^{-q\tau}\Phi(d_1) + Se^{-q\tau}\phi(d_1)\frac{\partial d_1}{\partial q} - e^{-r\tau}K\phi(d_2)\frac{\partial d_2}{\partial q}$$

$$= -Se^{-q\tau}\left[\tau\Phi(d_1) + \frac{\sqrt{\tau}}{\sigma}\phi(d_1)\right] + e^{-r\tau}K\phi(d_2)\frac{\sqrt{\tau}}{\sigma}$$

Also, in the same fashion, we can determine its put equivalent:

$$\Psi_p = \frac{\partial P}{\partial q} = Se^{-q\tau}\left[\tau\Phi(-d_1) - \frac{\sqrt{\tau}}{\sigma}\phi(-d_1)\right] + e^{-r\tau}K\phi(-d_2)\frac{\sqrt{\tau}}{\sigma}$$

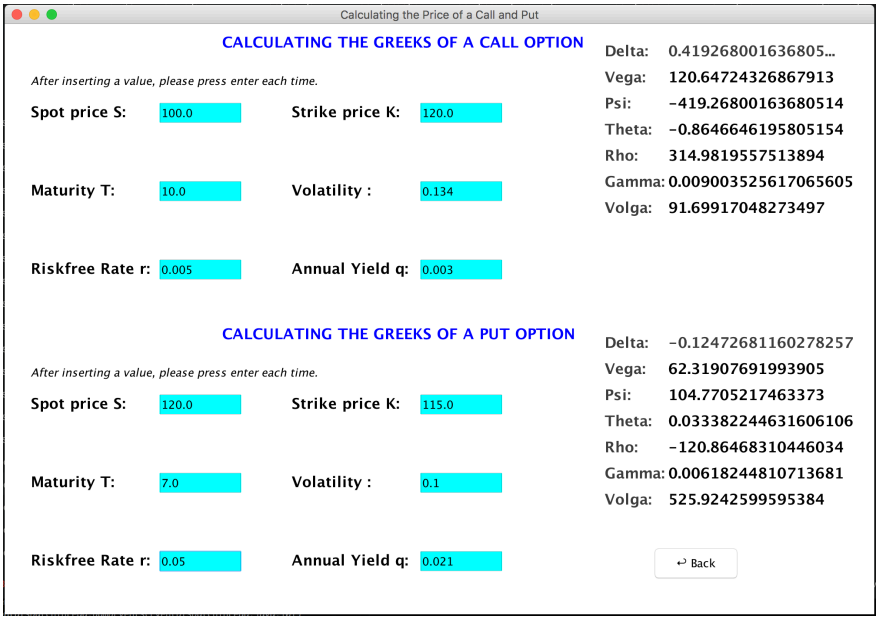

Figure 7: Calculating the Greeks. The user updates parameters by choice.

- .**GO TO SIMULATION**

a) Geometric Brownian Motion

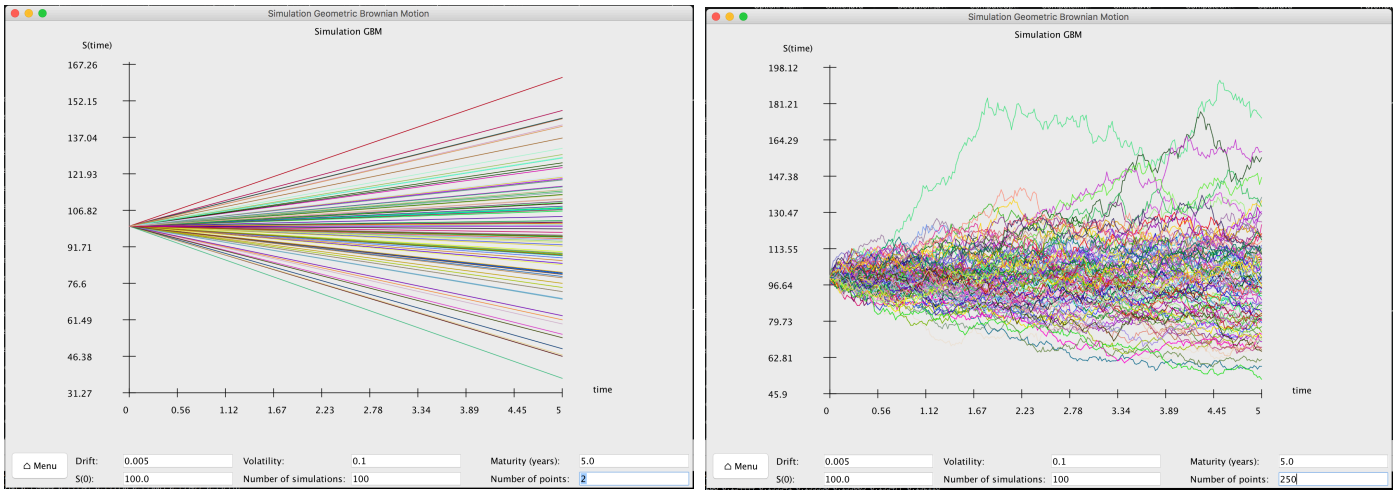

Figure 8: GBM varying parameter *Number of Points*. On the left, 2 points. On the right, 250 points.

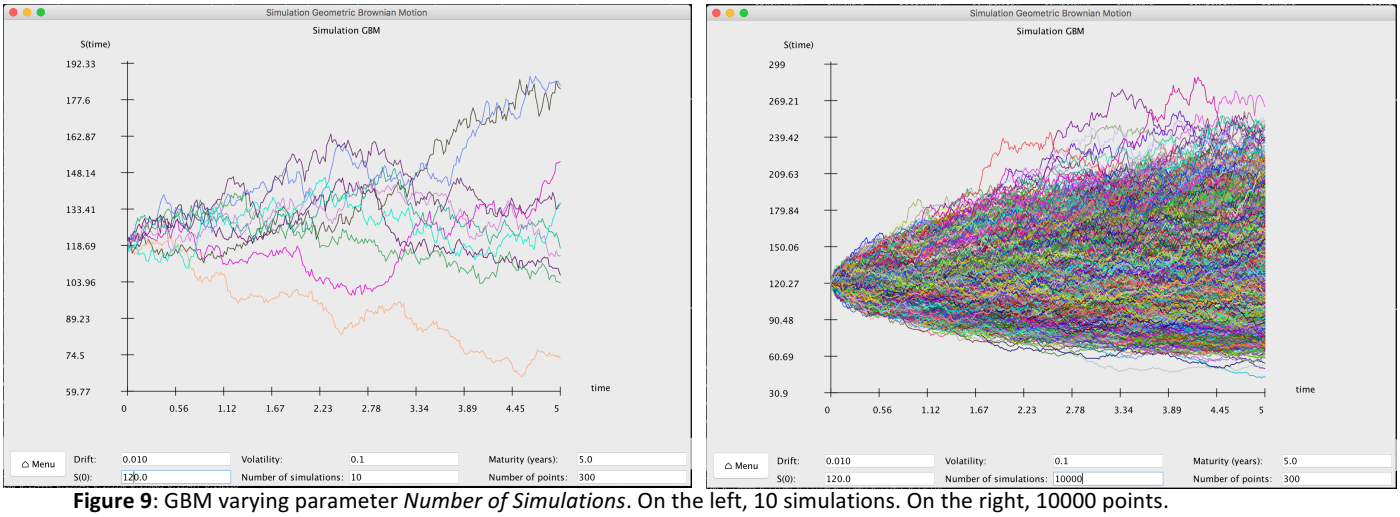

Figure 9: GBM varying parameter *Number of Simulations*. On the left, 10 simulations. On the right, 10000 points.

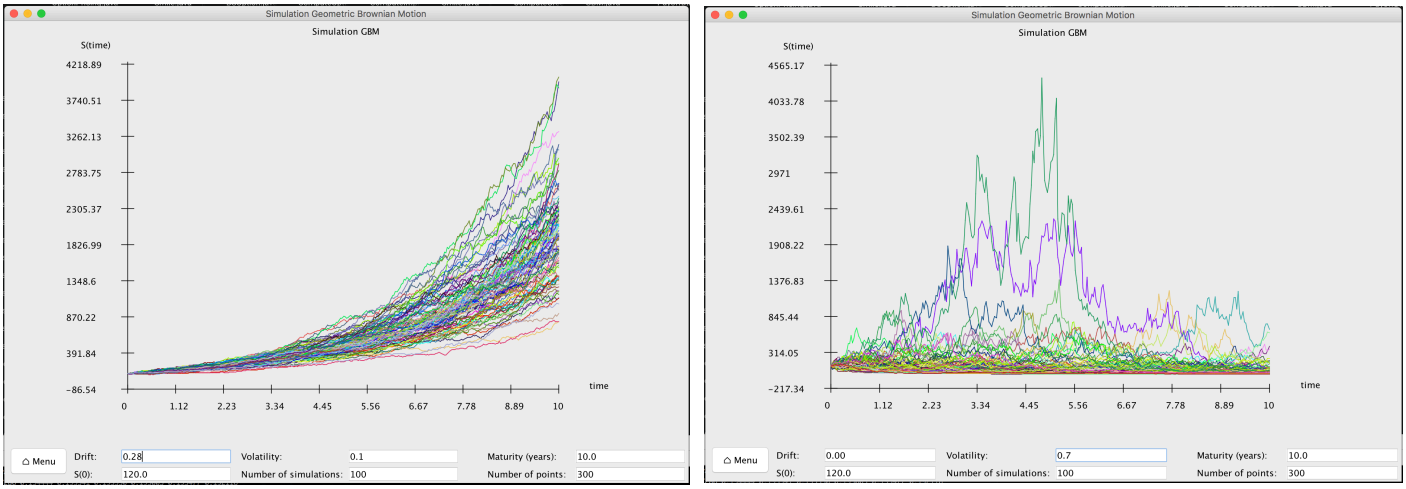

Figure 10: GBM varying parameters. On the left, parameter varied high *Drift* to 0.28. On the right, high *Volatility* to 0.7.

Demonstration of the Geometric Brownian Motion when adjusting different parameters:
In the Figures 8 to 10, you can the impact of different parameter shifts of the underlying asset on the Geometric Brownian Motion.
The simulation parameters are the number of points (number of time steps) and the number of simulations. Figure 8 and 9 shows the impact of the variation in these parameters. Higher number of points makes the simulation more precise and higher number of simulation helps to understand, on average, the behavior of such processes. For example, we can see on the right diagram of Figure 9 that, as expected by the underlying assumption of the GBM, the volatility is increasing over the time. Another important parameter is the drift (see Figure 10). It is the slope of the Brownian Motion. Negative drift would mean that on average over the time, the process decreases. On the Figure 10 (left), we see a rather extreme drift of 0.28 that skews the graph to the right, as opposed to a more realistic drift of 1% as inputted in the Figure 9 on the right picture.

b) Payoff/Value Diagram

Input from the user is the same as when calculating the value of options - S, K, t, T, r, q, σ**.** The output is the payoff diagram, i.e. the comparison between the price of the option at maturity and the price before maturity. Different curves on Figure 11 represent different times in the option's life. For a time that is equal to the maturity of the option (T-t=0), we observe a linear payoff as shown in the introduction slides at the beginning. The close the option is to its maturity, the closer will the curve be to the linear payoff. This logic works both for puts and calls.

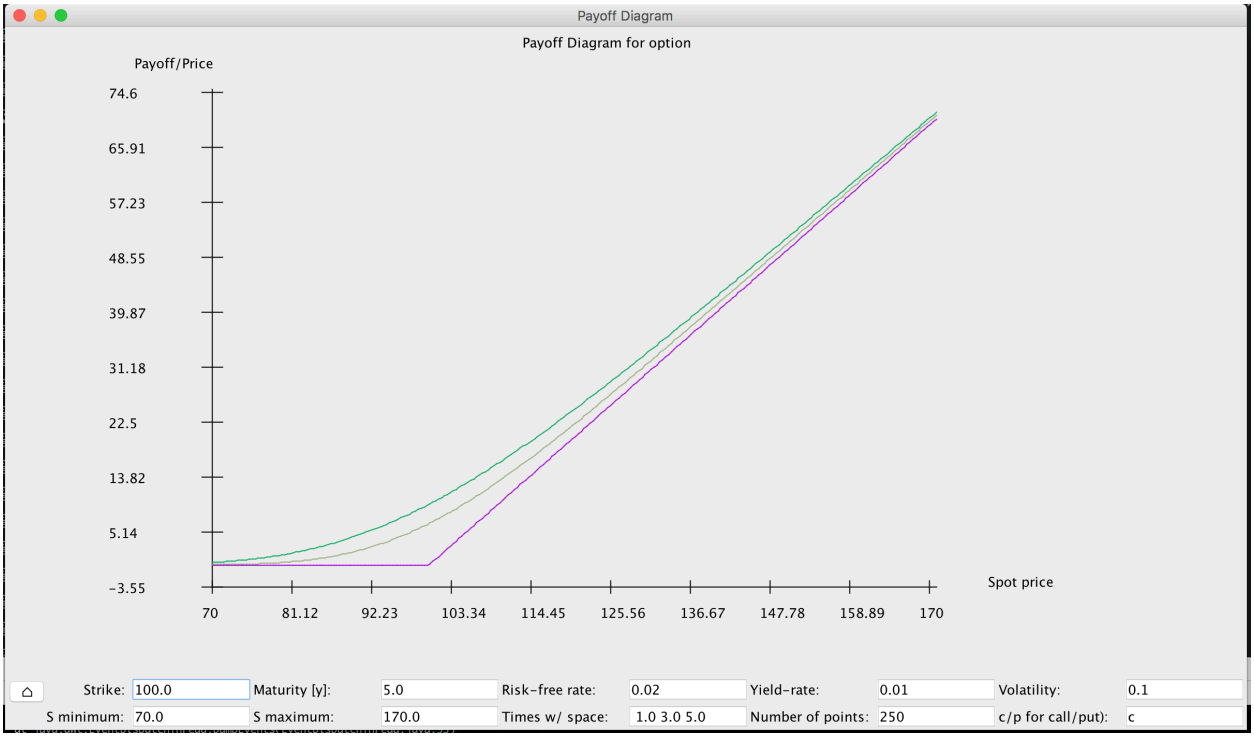

Figure 11: Plotting the Payoff Diagram for a call.

c) Volatility Smile

The volatility smile is a feature that is observed on the markets - for options with different strike prices, the implied volatility is not constant but rather looking as a smile with its lowest value obtained for a strike price equal to the spot price (option at-the-money). However, this feature only works when applied on real market data; were we to simply simulate random data, the graph would not display such feature, because most of the time, there would not exist a solution to the implied volatility equation (same as in the computation part, where we calculated the implied volatility).

Therefore, we use real market data. For each strike price, there is an equivalent option price. We input as default data the parameters of the volatility smile of the December 2017 Call option on GOOG. All parameters were estimated using Yahoo finance, and treasury.gov for the risk-free rate.

We can observe a volatility smile on this example. We can of course modify some characteristics, such as r, q, S or times, while maintaining a solution. However, for example changing call to a put option would create an error as we would need new market data for the put to have a solution of implied volatility.

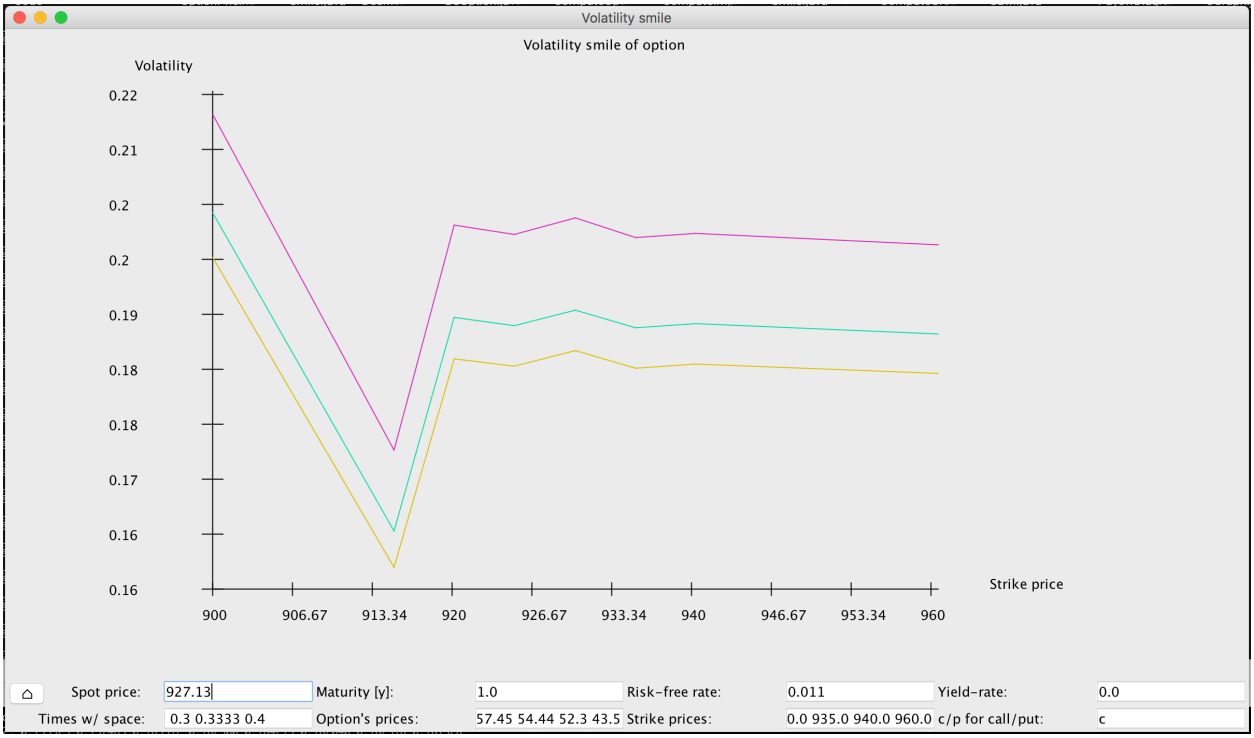

Figure 12: Calculating the Greeks. The user updates parameters by choice.

Notes on the Code

A brief structure of my code is as follows:

The program is built in two parts: GUI and back-end. The main GUI file is OptionPricing.java that takes care of the general interface and calls the relevant windows in the menus. The two submenus are called with class launchIntro() that extends a JFrame (use class for counter of slides) and the function launchPricing(). In launchIntro, the quizzes are launched using functions launchQuiz() and launchQuizB(). After, 6 subclasses extending JFrame launch the Option Pricing model applications: GBM(), PayoffDiagram() and Smile() for the simulation part, and ComputeOptionPrice(), ComputeImpliedVol() and ComputeGreeks() for the computation part. The simulation functions are all implemented as part of Class BSOption defined in BSOption.java. This class represent an Option with a type 'c' or 'p' for call and put, a strike price, a spot price, a risk-free rate, a yield rate, a volatility, a price and a maturity. This is also where the root-finding algorithms are implemented. Then comes the GBM.java where the simulation of Geometric Browmian Motion occur with an output of type List<double[]>. Then comes a similar principle in Smile.java where the smile is displayed. Finally, the Payoff Diagram is coded again in a similar fashion in PayoffDiagram.java. One important aspect is that all this programs have a built-in GUI to bind the parameter text field to the model in order to create an interactive visualization. The binding is made using JTextField that have actionListeners that trigger the update of the concerned parameter and the update of the graph. Finally, one main component of the application is the JGraph class implemented in JGraph.java that extends JPanel. This class uses a JPanel filled by paintComponent(g) in order to display the axes, the intervals and corresponding values, the title and the graphed value. It is important to note that the type of data taken-in by JGraph as a class attribute is a List<double[]>, that allows to draw multiple series (for instance 100 simulations of a GBM). The same class is used for all the visualizations of simulations.I would also like to clarify that I coded the assignment using the Atom text editor. I do not have Cloud9 with GUI, and was therefore not able to test my program in the Cloud9 ID. However, using terminal on my computer, I had no problem compiling the code. See README.txt.

Conclusion

In summary, my program in a main part contains an Option Pricing Calculator based on the Black-Scholes formula and generates simulation of a Geometric Brownian Motion. Additionally, based on the input parameters from the user, the program can also calculate implied volatility as well as Greeks, and display a diagram of option payoff and volatility smile. All calculated and simulated outputs are a well-known concept in the field of quantitative finance and derivative contracts, and are using into great extent the parameters from the Black-Scholes formula. Moreover, user has the possibility to obtain an introduction to derivatives and test his knowledge in a GUI implemented quiz.