École nationale de la statistique et de l'administration économique Paris

C++ Project

Pricer

Students:
Chengzhi SUN
Georgii NIKISHIN
Mohamed Aziz JOUINI
Julien SKLARIK

Teacher:
Prof. Roxana DUMITRESCU

We sincerely thank our teacher Prof. Roxana Dumitrescu for her C++ lessons.

January 2023

Contents

1 Introduction

In this project we developed a program to "price" options using three different methods: one using the Black-Scholes-Merton (BSM) model, the second using the Monte Carlo method and the third using PDE. We started by studying financial theory in order to fully understand the operation and distinction between these two methods before implementing them in C++.

On the one hand, the BSM model allows us to value options using an explicit formula, which is only possible in certain cases and under certain conditions. In particular, we considered European calls and puts for this model. On the other hand, the Monte Carlo method allows us to value options when there is no explicit formula. In this project, we focused on the valuation of six products: calls, puts, binary ("digital calls", "digital puts" and "double digital calls") and Asian options, although the BSM and Monte Carlo methods are valid in many other cases. Finally, as a bonus, our program provides a PDE valuation for European vanilla options.

We will first describe the features of the program, including the user case and the financial explanation, then its architecture, and finally the problems encountered and the possibilities for improvement.

2 Program features

2.1 Program use case

Here is the user interaction loop:

1. A command to be executed is requested from the user.
2. The user gives either a pricing command or an auxiliary command (i.e. "quit" or "help").
3. The program executes the command.
4. If there is an error, a message is displayed. Otherwise, an answer is given.
5. The program goes back to step 1 and waits for the user's next action unless the command "quit" is selected.

Example:
When the user selects PDEcall or PDEput, we ask him to enter the parameters of the option: time to expiry, risk-free interest rate, volatility and a maximum strike price. As an output, the user receives a plot of the market value of the option as a function of a spot price and a time to expiry. Here are examples of outputs obtained.

Figure 1: Payoff of a call option

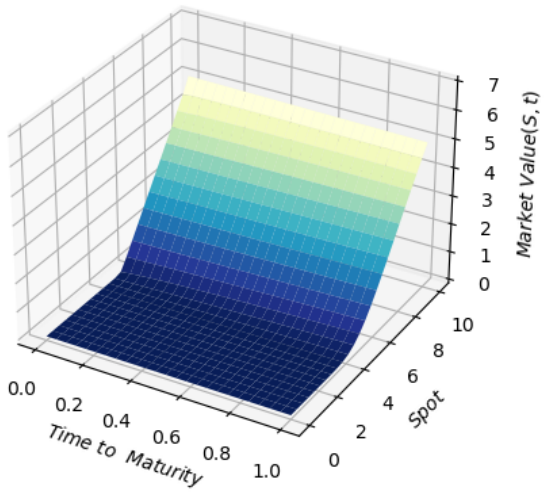
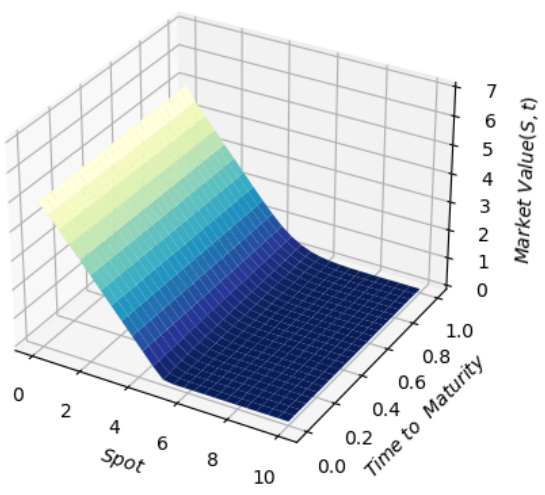


Figure 2: Payoff of a put option



2.2 Financial explanation

Option pricing theory is a probabilistic approach to assigning a value to an option contract. Models used to price options account for variables, such as current market price, strike price, volatility, interest rate, and time to expiration. Some commonly used models to value options are the Black-Scholes model and the Monte-Carlo simulation.

Notation:

S : Price of the underlying (stock price)

K : Strike price

r : Risk-free interest rate

μ : Tendency

σ : Volatility

t : Current time

T : Maturity

f : Payoff

2.2.1 Black-Scholes-Merton model

Under the risk-neutral probability measure $\mathbb{Q}$, the process of the stock price is given by:

$$\mathrm{d}S_t = \mu S_t \mathrm{d}t + \sigma S_t \mathrm{d}W_t$$

where W_t follows a standard Brownian motion

The Black-Scholes pricing theory, which assumes stock prices follow a log-normal distribution, tells us that the price of a vanilla option, with expiry T and payoff f, is equal to

$$e^{-rT}\mathbb{E}^{\mathbb{Q}}(f(S_T))$$

The payoff of the call option is $C_T = max(S_T - K, 0)$ and the payoff of the put is $P_T = max(K - S_T, 0)$.

Therefore the analytical option price has the following functional form:

$$c = S\mathcal{N}(d_1) - Ke^{-r(T-t)}\mathcal{N}(d_2)$$

where

$$d_1 = \frac{log(\frac{S}{K}) + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

and $\mathcal{N}(.)$ represents the cumulative normal distribution.

2.2.2 Monte-Carlo pricing method

The Monte-Carlo method provides a way to price a path-dependent option without analytical formula. The main idea of the Monte-Carlo method is to take advantage of the Law of Large Numbers. Given the payoff f, when the simulation number N is large enough, the price of the option

$$e^{-rT}\mathbb{E}^{\mathbb{Q}}(f(S_T))$$

can be approximated by

$$\overline{X}_N := e^{-rT}\frac{1}{N}\sum_{i=0}^{N} f(S_T^i)$$

where:

N : The number of simulations

S^i : The i^{th} simulation of the price of asset S

With the Central Limit Theorem (CLT), one has the following:

$$\sqrt{N}(\overline{X}_N - \gamma) \to \mathcal{N}(0, \sigma^2)$$

Then, one can write the discrete geometric Brownian motion under $\mathbb{Q}$:

$$S_{t_{i+1}} = S_{t_i} e^{(r-\frac{1}{2}\sigma^2)(t_{i+1}-t_i)+\sigma(W_{t_{i+1}}-W_{t_i})}$$

One can define $\delta t_i = t_i - t_{i-1}$ and simulate independent, normally distributed ε_i with mean 0

and standard deviation 1.

Let $S_{t_i} = log(S_{t_i})$. Then, for $i \geq 1$

$$s_{t_{i+1}} = s_{t_{i-1}} + (r - \frac{1}{2}\sigma^2)\delta t_i + \sigma\sqrt{\delta t_i}\varepsilon_i$$

and

$$\sqrt{\delta t_i}\varepsilon_i \stackrel{\mathcal{L}}{=} W_{t_{i+1}} - W_{t_i}$$

Then, with $S_{t_i} = e^{s_{t_i}}$, we can simulate the stock price at any time t.

Digital Options

Digital options are close to the mainstream vanilla options. Their difference is that their pay-off at expiry, $f(T)$, only takes two values. For the digital call option, we have

$$f(S(T)) = \begin{cases} 1 & \text{if } S(T) > K \\ 0 & \text{otherwise} \end{cases}$$

while for a put option, one has the following

$$f(S(T)) = \begin{cases} 1 & \text{if } S(T) < K \\ 0 & \text{otherwise} \end{cases}$$

Double Digital Options

Double digital options are like the classical digital options, but with two strike prices, the lower bound strike price K_L, and the upper bound strike K_U. The payoff function for holding such an option (call) is defined as follows:

$$f(S(T)) = \begin{cases} 1 & \text{if } K_L < S(T) < K_U \\ 0 & \text{otherwise} \end{cases}$$

Asian Options

Asian options are path-dependent options. The payoff of this option is determined by using the underlying asset's average price over a set period of time. For the case of discrete monitoring,

the payoff function for holding such an option (call) is defined as follows:

$$f(S(T)) = \max\left(0, \frac{\sum_{i=1}^{n} S_i}{n} - K\right)$$

2.2.3 Partial Differential Equation (PDE) option pricing

The Black-Scholes PDE for a call option equation is

$$-\frac{\partial C}{\partial t} + rS\frac{\partial C}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial^2 S} - rc = 0$$

where C is the price of the option as a function of stock price S and time t.

In the PDE section, we do a finite discretisation to reduce the continuous problem to the system of algebraic equations. Thus, one has the following:

$$-\frac{C^{n+1} - C^n}{\Delta t} + rS_j \frac{C_{j+1}^{n+1} - C_{j-1}^n}{2\Delta x} + \frac{1}{2}\sigma^2 S_j^2 \frac{C_{j+1}^n - C_j^n + C_{j-1}^n}{\Delta x^2} - rc = 0$$

where Δx is a change of a spot price.

Thereby, one can write the price of the option in the following way:

$$C_j^{n+1} = \alpha_j C_{j-1}^n + \beta_j C_j^n + \gamma_j C_{j+1}^n$$

where

$$\alpha_j = \frac{\sigma^2 j^2 \Delta t}{2} + -\frac{rj\Delta t}{2}$$

$$\beta_j = 1 - \sigma^2 j^2 \Delta t - r\Delta t$$

$$\gamma_j = \frac{\sigma^2 j^2 \Delta t}{2} + \frac{rj\Delta t}{2}$$

Hereby, we can implement the solution of a PDE by the finite difference method.

3 Program architecture

3.1 Fundamental structure

The code is structured around the *Console* and *Command* classes. The *Console* class **contains the user interaction loop**.

A distinction is made between the pricing commands which are used to determine the price of an option and the auxiliary commands which participate in the operation of the program: "help" displays a help message to detail the available commands with their descriptions, and "quit" terminates the program.

For greater flexibility, we have separated the pricing commands from the console loop. We can therefore code many pricing commands without needing to modify the code of the console too much. Auxiliary commands, on the other hand, are methods[1] of the *Console* class because they depend on *Console* attributes. All pricing commands are **child classes** of the *Command* class. The **parent class** *Command* contains everything that is common to all the commands, and the child classes contain their respective specific functions. For instance, the functions "getName()" and "getDescription()" are two **virtual commands** of *Command* specialised in each child class.

As the list of parameters can vary from one function to another, we decided to store them in a **container**. All parameters are character strings (entered by the user on the keyboard) in the application. As parameters can be numbers or dates, transformations take place in the code when needed. The type of container that we chose allows the parameters to be indexed by their name in order to be able to request their value from their name, like with Python's dictionaries.

Executing a command is done using the operator ()[2] of the *Command* class, it works as follows:

1. The command displays its name and a small description.
2. It requests its parameters from the user (such as strike price, volatility, spot price, interest rate and date to maturity).
3. It does the calculation with its parameters and displays the calculated price of the option.

[1] A method is a function whose first parameter is a pointer to the class instance (accessible with "**this**").
[2] N.B: The operator **operator ()** is used to make an object "callable" like a function.

4. If everything went well, it returns true.

5. If there is an error, it returns false, allowing the *Console* to signal the issue.

3.2 Additional class for pricing

To store and use the parameters of the option, we collect information in the 'Option' class, which also inherits a 'Payoff' functor that could be used to calculate the payoff of the option.

3.2.1 Classes for classical Call and Put options

Cumulative distribution function (CDF) for a standard normal variable:
To calculate the CDF of a standard normal distribution, we used the *erfc* function called from *cmath* C++ library. This function calculates the complementary error.
The definition of the complementary error function is:

$$ERFC(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt$$

The formula of the cdf using the ERFC function is

$$CDF(x) = 1 - \frac{ERFC(x/\sqrt{2})}{2} = \frac{ERFC(-x/\sqrt{2})}{2}$$

Date class:
The most important function to describe in the date class is period_diff(). It is a function that returns the difference between a specified date and the current date in years using the library **ctime**.
This means that **the user never has to enter the current date** because our algorithm will retrieve it automatically.
The **difftime** function calculates the difference in seconds between two **time_t** values. The result is then divided by the number of seconds in a year to get the difference in years.

3.2.2 Classes for Monte Carlo method

As explained in the slides of the lecture (1), to generate random numbers to apply the Monte Carlo method, using the Mersenne Twister algorithm is much better than the rand function.
We created therefore a static method corresponding to the Box-Muller algorithm

using Mersenne Twister, in a parent class. This way, each Monte Carlo option pricing child class can use gaussian random variables.
We also set the Heaviside[3] function as static for the same generalisation reason.

3.2.3 Classes for Partial Differential Equations.

To read the parameters for this module we have created a class *PDEparams*, the user enters a number of spatial differencing points and a number of temporal differencing points which are the discretisation parameters to solve the Black-Scholes equation. Be careful here, because in this section user enters the maximum possible spot price instead of the current spot price.

In this section, we have two main parts. First, the PDE classes. Here we have the class *ConvectionDiffusionPDE*, which is an abstract class used by the inherited classes. The *BlackScholes* class calculates the coefficients for the convection-diffusion PDE using the parameters from the *Option* class. Then there are the Finite Difference Method (FDM) classes. Here we have an abstract class *FDMbase* which contains the discretisation parameters. The last class, *FDMEulerExplicit*, provides exact methods for solving the PDE. This class inherits from the *FDMbase* class. It also has methods for calculating step size, initial conditions and boundary conditions.

For plotting, we use a 'display.py' file which uses matplotlib and pandas. As a result, the user receives a graph.

[3]The Heaviside function is a function returning 1 if the given value is positive and 0 otherwise.

4 Discussion of issues encountered

4.1 Resolving errors

Throughout our programming we encountered many bugs, most of which we solved by user testing. For example, to validate the basic functionality of our program, we started by creating a command called "commandTest", inheriting from *Command*, which only works if the user types "ok". It was important to us that our program wouldn't crash if the user typed something wrong or, for example, pressed "enter" without writing anything. "CommandTest helped us to test these details.

As for the PDE part, the most difficult problem was to combine the existing code structure with the parts needed to solve the PDE. For example, we had to modify the "Option" class by adding the inherited "type" field, which indicates the type of option ("type" is equal to 1 for calls and 0 for puts). We also had to create a separate class to store the sampling parameters. We also had a problem with displaying information. We solved this by creating a separate Python file, 'display.py'. This file plots the market value of the option as a function of the spot price S and time T.

To know whether our work is correct or not, we compared the results of our program with the returns of online pricers, and they were very similar. Here are the links to the websites that we used to compare and verify the results of our program:
Link to a pricer using the Black Scholes Merton model
Link to a pricer using the Monte Carlo method for Asian options

4.2 To go further

In order to optimise our program, one idea could be to ensure that the pricing commands are not created when the Console object is constructed, but rather when it is first used. In particular, by modifying the *Concole* class, we could optimise our code so that the pricing commands remain created after being called, so that they do not have to be recreated each time. To do this, we could use a *Singleton* class to keep the commands created in one instance, using conditional loops.

As far as the PDE part is concerned, we still have some problems when users enter a volatility greater than 0.3 or a time to expiration greater than 2. This is due to the use of the finite

increment method, as the equation is very sensitive to these parameters and "explodes" when they increase significantly. In addition, the PDE result is also sensitive to the sampling parameters and we can also observe an "explosion" if they are too large (greater than 50). In addition, our PDE command can, by empirical calculation, give fundamentally incorrect results, such as negative values for the market value of the option, when the real value is very close to 0. We therefore recommend using the following values: $temporal\_differencing\_points = spatial\_differencing\_points = 30$.

5 Conclusion

Thinking together about a code structure during an analysis phase before starting to write the program enabled us to build our application in the best possible way from the start. Helping each other by meeting to work together also helped to ensure that each team member learned from the project. During this project we learnt how to code in C++, how to efficiently use object-oriented programming methods and the financial valuation theory of the Black-Scholes model, the Monte Carlo and the Partial Differential Equations methods. We spent a lot of time developing this application and we are proud of the result of our teamwork.

6 Bibliography

1. C++ Lectures of our teacher, Prof. Roxana Dumitrescu.
2. "C++ FOR QUANTITATIVE FINANCE" By Michael L. Halls-Moore is used for Payoff, FDM and PDE classes.
3. "Pricing Exotic Options Using C++" By Tawuya D R Nhongo for exotic options.
4. We learned about Digital and Asian pricing methods on *quantstart.com*.