C++ project on option pricing

REISSE Louis, HUANG Frederic, BENHANA Bilal, MATHIEU Adrien

30 décembre 2023

Résumé

This dissertation delves into the realm of financial derivatives pricing, employing C++ as the programming language for model implementation and analysis. The project is structured into distinct yet interconnected parts, each contributing to a holistic understanding of the pricing process.

Mathematical Setup :
The foundation of the project begins with a rigorous exploration of the mathematical concepts underpinning financial derivatives pricing. Emphasis is placed on understanding stochastic calculus and key financial mathematical principles. This section serves as the theoretical backbone, providing the necessary groundwork for subsequent model development.

Black-Scholes Model :
The Black-Scholes model, a cornerstone in options pricing theory, is introduced. This part elucidates the assumptions and derivations of the model.

Pricing Model :
Building upon the Black-Scholes foundation, a comprehensive pricing model is developed theoretically, sheding light on the importance of the Monte Carlo method in order to efficiently price options, when closed-form are not provided.

Construction of a Replication Strategy :
A critical aspect of derivative pricing is the construction of a replication strategy, which involves creating a portfolio of underlying assets to mimic the payoff of the derivative. This section explores the replication technique and strategy.

Presentation of the Code :
The practical implementation of the pricing model and replication strategy is showcased through detailed code presentations in C++. The code is structured for clarity, modularity, and efficiency, allowing for easy comprehension and potential future extensions.

In conclusion, this project not only provides a comprehensive overview of financial derivative pricing but also bridges the gap between theoretical foundations and practical implementation through the use of C++.

Table des matières

1 Mathematical setup

We consider a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, a maturity $T > 0$ and a standard brownian motion $B = (B_t)_{t \leq T}$.
We define a filtration $(\mathcal{F})_t$, the filtration generated by the browian motion.

Let X an Itô process written as follow :

$$dX_t = b(t, X_t)dt + \sigma(t, X_t)dB_t, X_0 = x_0 \tag{1}$$

where $b, \sigma : [0, T] \times \mathbb{R} \leftarrow \mathbb{R}$ are such as there is existence and uniqueness of the SDE.

Theorem 1.1 (Ito's formula)**.** *Let $f \in \mathcal{C}^{1,2}(\mathbb{R}+, \mathbb{R})$ and X an Itô process written as follow, then :*

$$df(t, X_t) = \partial_t f(t, X_t)dt + \partial_x f(t, X_t)b_t(X_t)dt + \partial_x f(t, X_t)\sigma_t(X_t)dB_t + \frac{1}{2}\partial_{xx} f(t, X_t)\sigma_t(X_t)^2 dt \tag{2}$$

Theorem 1.2 (exponential martingale)**.** *Let X an Itô process such that*

$$\forall t \in [0, T], \int_0^t X_u^2 du < \infty$$

we set :

$$Z_t^X = e^{\int_0^t X_u dB_u - \frac{1}{2}\int_0^t X_u^2 du} \tag{3}$$

Then Z^X is a local martingale.

Theorem 1.3 (Girsanov theorem)**.** *If Z^X is a true martingale, then we can construct a probability measure, $\mathbb{Q}$ on $(\Omega, \mathcal{F})$ defined as follow :*

$$\frac{d\mathbb{Q}}{d\mathbb{P}} = Z_T^X$$

Under this new measure, $W_t := B_t - \int_0^t X_u du$ is a brownian motion.

2 Black-Scholes-Merton model

We consider our initial probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and our brownian motion B.

Theorem 2.1 (Black-Scholes-Merton model)**.** *Let S a stochastic pricing process of an option , μ its drift, σ its volatility. In the Black-Scholes model, we get that :*

$$dS_t = S_t(\mu dt + \sigma dB_t), S_0 = x \tag{4}$$

We can find an explicit formula for this solution, and we get :

$$S_t = e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma B_t} \tag{5}$$

Theorem 2.2 (Model under the neural risk probability)**.** *We consider the Black-Scholes model and we set $X_t = -\lambda t$, where $\lambda = \frac{\mu - r}{\sigma}$ with r an interest rate. Then, by the Girsanov theorem, we obtain a new probability $\mathbb{Q}$ under which $W_t := B_t + \lambda t$ is a brownian motion.*

Under this new probability, we can rewrite the dynamic and the explicit form of the pricing process as follow :

$$dS_t = S_t(rdt + \sigma dW_t), S_0 = x \tag{6}$$

and

$$S_t = xe^{(r - \frac{1}{2}\sigma^2)t + \sigma W_t} \tag{7}$$

3 Pricing model

We still consider our initial probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and our new probability space obtain by the Girsanov theorem $(\Omega, \mathcal{F}, \mathbb{Q})$.

We consider a measurable function ϕ and depending on the underlying price S.

In our project, we will only consider path-independant options, then the option's payoff will be under the following form : $\phi(S_T)$ at the maturity T.

Proposition 3.1 (Pricing formula)**.** *Set $p(t, S_t)$ the value of an option at time t, $t \leq T$ and based on the underlying asset S.*
Then we get :

$$p(t, S_t) = e^{-r(T-t)}\mathbb{E}[\phi(S_T)|\mathcal{F}_t] \tag{8}$$

After rewriting, we get :

$$p(t, S_t) = e^{-r(T-t)}H(S_t) \tag{9}$$

where $H(x) := \mathbb{E}[\phi(xe^{(r-\frac{1}{2}\sigma^2)(T-t)+\sigma(W_T-W_t)})]$

Démonstration. Application of the conditional expectation properties, et the browian motion property : for W a brownian motion, $W_t - W_s$ is independant of $\mathcal{F}_s, \forall t \geq s \geq 0$. □

Proposition 3.2 (Monte Carlo estimation)**.** *We recall that, if X is an integrable random variable, then we can estimate $\mathbb{E}[X]$ by*

$$\frac{1}{n}\sum_{k=1}^{n} X_k$$

where the X_k are iid and have the same law as X.

Proposition 3.3 (Pricing by Monte Carlo)**.** *We can estimate the value of $p(t, x)$ by*

$$\tilde{p}_n(t, x) = \frac{1}{n}e^{-r(T-t)}\sum_{k=1}^{n}\phi(xe^{(r-\frac{1}{2}\sigma^2)(T-t)+\sigma\sqrt{T-t}Z_k}) \tag{10}$$

where the Z_k are iid and have the same law as $\mathcal{N}(0, 1)$.

4 Construction of a replication strategy

Definition 4.1 (Replication strategy / portfolio)**.** *A replication strategy for a payoff option $\phi(S_T)$ is given by a pair (x, ϕ) where x corresponds to the initial value of the strategy (initial value of the replication portfolio), and $\psi = (\psi_t)_t$ is the quantity of risky assets contained in the portfolio.*

Proposition 4.1 (Porfolio characterization)**.** *If we denote as V_t the porfolio at time t, we get, under the model's hypothesis, that $V_t = p(t, S_t)$.*
The porfolio corresponds to the value of the option we possess.

Definition 4.2 (Self-financing condition)**.** *We say that a porfolio $V_t^{x,\psi}$ verifies the self-financing condition if its present value, ie. $\tilde{V}_t^{x,\psi}$, equals $\tilde{V}_t^{x,\psi} = x + \int_0^t \psi_r \sigma \tilde{S}_t dW_t$, where $\tilde{S}_t := S_t e^{-rt}$.*

Remark. *We can explain the self-financing principle as the fact that the value of the portfolio varies only via the variation of the risky asset (the asset that follows the Black-Scholes dynamic).*

Remark. *In particular, we note that the discounted portfolio is a (local) martingale.*

Theorem 4.1 (Duplication)**.** *Under the model's hypothesis, we get that :*

$$x = p(0, S_0), \psi_t = \frac{\partial}{\partial x} p(t, S_t)$$

Démonstration. We recall that, on the one hand $\tilde{V}_t$ has the following form :

$$\tilde{V}_t^{x,\psi} = x + \int_0^t \psi_r \sigma \tilde{S}_t dW_t$$

and, on the other hand, that

$$V_t = p(t, S_t)$$

Then, by applying the Itô's formula to the function

$$f(t, S_t) := e^{-rt} p(t, S_t)$$

we obtain :

$$\tilde{V}_t = p(0, S_0) + \int_0^t e^{-ru}(-rp(u, S_u) + \partial_t p(u, S_u) + r\partial_x p(u, S_u) S_u + \frac{1}{2}\sigma^2 \S_u^2 \partial_{xx} p(u, S_u))du \quad (11)$$

$$+ \int_0^t e^{-rt} \partial_x p(u, S_u) \sigma S_u dW_u \quad (12)$$

However, $\tilde{V}_t$ being a (local) martingale, we get, by the martingale representation theorem, that

$$-rp(u, S_u) + \partial_t p(u, S_u) + r\partial_x p(u, S_u) S_u + \frac{1}{2}\sigma^2 S_u^2 \partial_{xx} p(u, S_u)) = 0$$

and then we get :

$$\tilde{V}_t = p(0, S_0) + \int_0^t \partial_x p(u, S_u) \sigma \tilde{S}_u dW_u \quad (13)$$

By direct identification, we obtain the values of x and ψ_t.. □

Remark. *We define the Δ of an option as follow :*

$$\Delta = \partial_x p(t, S_t)$$

As a result, the Δ is interpreted as the asset quantity of the option's duplication portfolio.

5 Presentation of the C++ code

5.1 Project Architecture and OOP Implementation Overview

Our Monte Carlo Option Pricing project is thoughtfully organized, utilizing object-oriented programming principles and a modular file structure to enhance maintainability and readability. The key files include main.cpp, option.cpp, option.h, and pricing.h, each contributing to the project architecture.

5.1.1 File Structure

- **main.cpp** : As the main entry point, main.cpp orchestrates the option pricing simulation. This file handles parameter setup, function calls from other files, and result display or logging.
- **option.cpp** and **option.h** : the heart of our object-oriented design lies in option.cpp and option.h. These files contain the generic Option class, representing European vanilla options. Within the same files, we find both the Call and Put classes, derived from Option.
- **pricing.h** : This file houses declarations and definitions related to pricing methods. It encompasses functions or classes responsible for generating Brownian motion, and other pricing-related computations.

5.1.2 Object-Oriented Design

- **generic option class** : The Option class, defined in option.h and implemented in option.cpp, serves as the base class. It includes virtual methods (payoff, Setdelta, and price) to be overridden by derived classes.
- **derived put and call class** : Both the Call and Put classes, residing in the same option.cpp and option.h files, inherit from Option. They override the virtual methods to provide option type-specific implementations.
- **polymorphism** : Polymorphism is achieved through virtual methods, enabling the use of pointers or references to the base Option class for working with objects of derived classes.
- **copy constructor** : A copy constructor within the Option class facilitates easy creation of a Call from a Put and vice versa, contributing to code flexibility and reuse.

5.1.3 Suggestions for Further Improvement

- **performance optimization** : Explore opportunities for performance optimization within the option.cpp and pricing.h files, such as vectorization or parallelization techniques.
- **testing and validation** : Thoroughly test functions within the option.cpp and pricing.h files to ensure accuracy and reliability.
- **real data comparison** : Compare model outputs with real market option prices for validation and improvement, focusing on methods within the option.cpp and pricing.h files.
- **historical volatility** : Implement a mechanism within pricing.h to calculate historical volatility based on past market data for a more realistic simulation.
- **extension to exotic options** : Enhance the project's versatility by extending the Option hierarchy to include exotic options. Introduce new classes for exotic options, implementing their specific payoff functions, and integrate these into the Monte Carlo simulation.

5.2 More details on code :

To begin with, we recall that this project has for purpose to determine the price of a random European vanilla option (Call and Put), with their replication strategy by using Monte-Carlo Method and Simulations.

Therefore in the end, we will have at each time t smaller than the maturity T of the option :

- The price of the option

— The payoff of the option
— The delta of the option

We will be able to get the put informations if we have already calculated a call and get the call informations if we have already calculated the put with the same parameters.

So, to get all these informations with Monte-Carlo Method, we have to implement some functions to get a brownian path.

In order to get a brownian path, we implemented few functions :

— a standard normal distribution 'standardnormaldist' that creates a N-array of values following a N(0,1)
— the function 'BM' that takes an N-array from 'standardnormaldist' and transforms it into a Brownian Motion with a time step s and N values.

Now we can create a Brownian Motion easily with these function.

With this 'normal' brownian motion, we will create a new brownian motion that will have for parameters the risk-free rate, the volatility, the initial price of the underlying asset in order to simulate a potential path for the future price of the underlying asset. Function : 'underlying-trajectory'

The simulation of Monte-Carlo consists in the simulation of a lot of these paths for the underlying trajectory, then we will take the mean for each time step of the price value of all the trajectory in order to get a 'mean' trajectory that should in theory tend to the real future trajectory of the underlying asset according to the law of large number.

So, we created 'MonteCarloSimulation' that takes the mean of all payoff (and not trajectory), and with a discounting term, we get the present value of the mean payoff for each time step t.

With the 'deltatrajectory' function, we can get the trajectory in the time of the delta by differentiating the value of the option by the price of the underlying asset. We simply do the ratio of the variation of the price of the option between two time step over the variation of the price of the underlying asset between two time step.

Moreover, in order to be sure that our values are near from the reality (from the theory here), we implemented the CDF of the normal distribution and, according to the formulas of the Black-Scholes Models, we implemented price() and Setdelta for both call and put, that gives the theoric price of the option and its delta.

What we could optimize or where we could go further on this code :

We could have tried to compare our code with some realistic european option values on the stock markets. We could have made a code to implement the implicit volatility and use it to determine option price by ourselves and compare with the stock values. We could also implement a code that gives historic values of the volatility, by doing a regression on the log-values of the price of the underlying asset on the time.

5.3 Problems encountered

During the development of our Monte Carlo Option Pricing project, we encountered an issue related to multiple definitions of functions declared in the 'pricing.h' file. This problem

arises when the header file is included in multiple source files, leading to conflicts during the linking phase. To resolve this issue, we strategically applied the ‘inline‘ keyword to the function definitions in ‘pricing.h‘. The ‘inline‘ keyword suggests to the compiler that it should attempt to generate inline code for a function rather than generating a separate copy of the function for each source file that includes the header. This solution effectively mitigated the problem of multiple definitions, ensuring a seamless integration of the pricing methods across the project while maintaining code consistency and avoiding linker errors.