C++ Option Pricing Project Presentation

Adrien MATHIEU, Louis REISSE, Bilal BENHANA, Fréderic HUANG

January 12, 2024

Introduction

- C++ programming project defense
- Pricing European options using Monte Carlo methods

Table des matières

Mathematical framework

Consider a probability space $(\Omega, \mathcal{F}, \mathbb{Q})$, a Brownian Motion W, and the stochastic pricing process S following the dynamic :

$$dS_t = S_t(rdt + \sigma dW_t), S_0 = x \tag{1}$$

and that can be explicitly solved

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

Pricing model

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.

Model (Pricing model)

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 :

$$p(t, S_t) = e^{-r(T-t)}\mathbb{E}[\phi(S_T)|\mathcal{F}_t] \tag{3}$$

After rewriting, we get :

$$p(t, S_t) = e^{-r(T-t)}H(S_t) \tag{4}$$

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

Monte Carlo simulation

Model (Pricing by Monte Carlo)

We can estimate the value of $p(t,x)$ by

$$\tilde{p}_n(t,x) = \frac{1}{n} e^{-r(T-t)} \sum_{k=1}^{n} \phi\left(x e^{(r-\frac{1}{2}\sigma^2)(T-t)+\sigma\sqrt{T-t}Z_k}\right) \tag{5}$$

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

Replication porfolio

- We denote as V_t the porfolios value at time t. 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.
- 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 $\tilde{S}_t := S_t e^{-rt}$.
- Under the model's hypothesis, we get that :

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

- We find x using our pricing model, and once we get $\tilde{p}_n(t, x)$ we can compute ψ_t by finite difference methods.

Project Architecture Overview

- **File Structure:**
  - `main.cpp` - Main entry point orchestrating the simulation.
  - `option.cpp` and `option.h` - Core files with `Option`, `Call`, and `Put` classes.
  - `pricing.h` - Declarations and definitions for pricing auxiliary function.

Code summary

This code has for purpose to determine the price of a random European plain vanilla option for both call and put. This code will deliver in the end for each time t before maturity T :

- The price of the option `price` and `MonteCarloSimulation`
- The payoff of the option : `payoff`
- The theorical delta of the option : `delta` and `Setdelta`
- A simulation of the underlying trajectory process : `underlying-trajectory`
- A simulation and estimation of the errors of the delta trajectory process (core of the replication strategy) : `delta-trajectory`

Auxiliary functions

To get all these information with Monte-Carlo Method, we have to implement some auxiliary functions.
Let's begin with the simulation of a brownian path which is the path used in BSM-Model hypotheses. To get a brownian path, we implemented few functions such as follows :

- A standard normal distribution 'standardnormaldist', that creates a N-array of random variable following a N(0,1)
- A function 'BM' that takes a N-array from 'standardnormaldist' and transforms it into a brownian motion with a time step s and N values
- This normal brownian motion with the function 'underlying-trajectory' will be transformed into a geometric brownian motion with a drift 'r' and a diffusion 'sigma'

OOP Implementation Overview

- **Object-Oriented Design:**
  - `Option` Class:
    - Represents a generic European vanilla option.
    - Includes virtual methods (`payoff`, `Setdelta`, `price`) to be overridden by derived classes.
  - Derived Classes:
    - `Call` and `Put` classes derive from `Option`.
    - Override virtual methods with option type-specific implementations.
  - Polymorphism:
    - Achieved through virtual methods, allowing flexibility in using pointers or references to the base `Option` class.
    - Overloaded the << operator to enable consistent printing of option information across different option types.
  - Copy Constructor:
    - Implemented within the `Option` class for easy creation of a `Call` from a `Put` and vice versa, contributing to code flexibility and reuse.

Potential Improvements

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 (Asian option, Barrier Option, Lookback option). Introduce new classes for exotic options, implementing their specific payoff functions, and integrate these into the Monte Carlo simulation.

Graph: Our project may lack graphical representations.

Issues encountered

- Multiple definitions: Since we had multiple definitions of functions in 'princing.h', we encountered conflict during the linking phase.
- Theoretical Issue: We had little issues with the theory which made our values more imprecise.