

# 1 Project 3: Monte Carlo Option Pricing Terminal

## 1.1 Objective

The goal of this project is to implement a professional-grade simulation tool to price various financial derivatives using the Monte Carlo method. The tool handles both standard European options and complex path-dependent “exotic” options.

## 1.2 Mathematical Framework

The underlying asset price  $S_t$  is assumed to follow a Geometric Brownian Motion (GBM), discretized as follows:

$$S_t = S_{t-1} \exp \left( \left( \mu - \frac{\sigma^2}{2} \right) dt + \sigma \sqrt{dt} W \right).$$

where:

- $\mu$ : expected annual return (drift),
- $\sigma$ : annual volatility,
- $W$ : random variable following a standard normal distribution  $\mathcal{N}(0, 1)$ .

## 1.3 Supported Instruments

- **Vanilla call**: standard right to buy the asset.
- **Tunnel**: a strategy involving a cap and a floor to limit the gain/loss variance.
- **Himalaya**: a path-dependent option based on the average performance of the asset over multiple dates.
- **Napoleon**: an option paying a coupon adjusted by the worst periodic performance.

## 1.4 Technical Implementation

- **Language**: Python 3.14.0.
- **Engine**: vectorized **NumPy** operations for high-speed simulations.
- **Interface**: **Streamlit** web application for real-time adjustment.
- **Outputs**:
  - estimated option price,

- statistical error (99% confidence level),
- convergence graph: visualizing the stabilization of the price as the number of simulations increases.

## 2 Project 4: PDE Numerical Solver (Finite Difference)

### 2.1 Objective

This project focuses on the numerical resolution of Partial Differential Equations (PDEs) applied to finance. By using finite difference methods, we compute the price of financial instruments on a grid without relying on random simulations.

### 2.2 Numerical Scheme

We implement the **Crank–Nicolson** scheme ( $\theta = 0.5$ ), which offers a high degree of stability and second-order temporal accuracy. The resulting tridiagonal system is solved efficiently at each time using the **Thomas algorithm**.

### 2.3 Implemented Models

The tool reproduces four classic financial models provided in the course annexes:

- **Black–Scholes**: for European equity options.
- **Cox–Ingersoll–Ross (CIR)**: for mean-reverting interest rates with non-negativity constraints.
- **Vasicek**: for mean-reverting interest rates.
- **Merton**: for firm value and credit risk modeling.

### 2.4 Key Parameters (Annex Compliance)

The solver is calibrated using the mandatory parameters from the project specifications:

- **CIR**:  $\kappa = 0.8$ ,  $\theta = 0.10$ ,  $\sigma = 0.5$ ,  $\lambda = 0.05$ .
- **Vasicek**:  $a = 0.95$ ,  $b = 0.10$ ,  $\sigma = 0.2$ ,  $\lambda = 0.05$ .
- **Black–Scholes**:  $K = 100$ ,  $\sigma = 0.20$ ,  $r = 0.08$ .

## 2.5 Technical Stack

- **Language:** Python 3.14.0.
- **Architecture:** modular design with separate classes for models and solvers.
- **Visualization:** price curves at  $t = 0$  compared to terminal payoffs at  $T$ .

## 3 Getting Started: Running the Pricing Terminal

To ensure the application runs smoothly, follow these steps to set up your environment and launch the dashboard.

### 3.1 Prerequisites

Ensure you have **Python 3.8** or higher installed on your machine. You can check this by running:

```
python --version
```

### 3.2 Creating a Virtual Environment

It is highly recommended to use a virtual environment to avoid version conflicts between libraries.

- **Windows:**

```
python -m venv your_env  
.\your_env\Scripts\activate
```

- **macOS / Linux:**

```
python3 -m venv your_env  
source your_env/bin/activate
```

### 3.3 Installing Dependencies

Once the environment is activated, install the required packages (NumPy, SciPy, Matplotlib and Streamlit):

```
pip install -r requirements.txt
```

### 3.4 Project Structure

```
/your-project-folder
|-- app.py           # The UI (Streamlit)
|-- models.py        # Financial models (GBM, CIR, Vasicek, BS, Merton)
|-- instruments.py    # Options payoffs
|-- solvers.py        # Monte Carlo & Thomas Algorithm
```

### 3.5 Launching the Application

To start the professional web interface, run the following command in your terminal:

```
streamlit run app.py
```

A new tab will automatically open in your web browser (`localhost`).

### 3.6 Stopping the App

To stop the server, you can press `Ctrl + C` directly in the terminal where `app.py` has been launched.