GRANDE ÉCOLE D'ACTUARIAT ET DE GESTION DES RISQUES
INSTITUT DE SCIENCE FINANCIÈRE ET D'ASSURANCES

Université Claude Bernard Lyon 1

Institut de Science Financière et d'Assurances

M1 Actuariat
Années 2022/2023
TER

Stochastic modeling and barrier options pricing

Group :
Mr Théo Jalabert
Ms Sabrina Khorsi
Ms Lou Simoneau-Friggi

Supervising teacher :
Ms Anne Eyraud

April 25, 2023

Contents

Abstract

Over the last ten years, the financial products available on the markets have become particularly complex: after the first generation of derivatives introduced in the 1970s, based on the famous work of Black and Scholes, the second generation of exotic products appeared in the early 1990s. Among them, and this is the subject of this report, are barrier options, whose main characteristic is the dependence of the option's terminal flow on the full path of the underlying asset's price.

In this report, we study the different pricing methods for barrier options. We will use the Heston model with jumps, otherwise known as the Bates model, which is a stochastic volatility model used in activating derivatives. We present the analytical solutions of this model before looking at the implementation of the pricer in Python thanks to the Monte-Carlo method using variance reduction methods such as the antithetical variables method. We then calibrate our model with S&P500 data and analyse the time to price and parameter settings. Finally, we study the pricing of barrier options under the binomial tree model.

Résumé

Au cours des dix dernières années, les produits financiers disponibles sur les marchés se sont particulièrement complexifiés : après les produits dérivés de première génération introduits dans les années 1970, et sur lesquels s'appuient les célèbres travaux de Black et Scholes, les produits exotiques de deuxième génération ont fait leur apparition au début des années 1990. Parmi eux, et c'est ce dont fait l'objet ce rapport, les options barrières dont la caractéristique principale réside dans la dépendance du flux terminal de l'option en la trajectoire complète du cours de l'actif sous-jacent.

Dans ce rapport, nous étudions les différentes méthodes de pricing pour les options à barrières. Nous utiliserons le modèles de Heston avec sauts autrement appelé modèle de Bates, notamment utilisé en dérivés actions, qui est un modèle à volatilité stochastique. Nous présentons les solutions analytiques de ce modèle avant de s'intéresser à l'implémentation du pricer en Python grâce à la méthode de Monte-Carlo en utilisant des méthodes de réduction de variance comme celle des variables antithétiques. Ensuite, nous calibrons notre modèle grâce aux données de l'index S&P500 puis analysons les temps de fixations des prix et des paramètres. Pour finir, nous avons étudié le pricing d'option barrières sous le modèle d'arbres binomiaux.

Key-words

Finance , pricing , options , Monte-Carlo method , stochastic modeling , barrier option , underlying asset , Heston model , Bates model , variance reduction , antithetical variables , calibration , S&P500 , Euler-Maruyama discretisation, binomial tree method

Introduction

A barrier option is a financial derivative product on an underlying asset (St) where the payment at maturity depends on whether or not the underlying asset's price crosses one or more pre-defined barriers. Barrier options come in three forms:

- Knock-in barrier options: the option can only be exercised if the underlying asset's price crosses a certain "barrier". There are two types:
  - Up and in option: the option is activated when the underlying asset's price goes above the barrier.
  - Down and in option: the option is activated when the underlying asset's price goes below the barrier.
- Knock-out barrier options: the option can no longer be exercised if the underlying asset's price crosses a certain "barrier". There are two types:
  - Up and out option: the option is deactivated when the underlying asset's price goes above the barrier.
  - Down and out option: the option is deactivated when the underlying asset's price goes below the barrier.
- There are also options that combine a knock-in and a knock-out option. The option then has two barriers, one that is activating and one that is deactivating.

A distinction is made between European barrier options and American barrier options. In the case of European barriers, observation is said to be "in fine" or "at maturity only." The underlying asset level is observed only on the last day to determine whether the underlying asset price has reached a barrier. In the case of American barriers, observation is said to be "continuous." The underlying asset level is observed at every moment to determine whether a barrier has been breached.

Barrier options have several advantages. First, they allow investors to protect themselves against unfavourable price variations. By setting levels for the underlying asset price at which the option is activated or deactivated, investors effectively protect themselves against potential important losses. Barrier options also offer the opportunity for short-term profits. They allow investors to speculate on short-term price variations by adopting positions based on pre-defined barriers. If market forecasts come true, barrier options offer opportunities for quick profits. Additionally, barrier options allow for great flexibility in designing investment strategies. They offer investors the ability to customize barriers based on their needs and goals. It is also possible to combine barrier options with other financial products to meet investor desires efficiently.

It is important to note that because there is a greater uncertainty with their exercise, barrier options are cheaper than vanilla options with the same characteristics, making them attractive.

However, to compensate for the risk taken on a deactivating barrier option, the investor can opt for a barrier option with a rebate. The investor will then be refunded a

financial amount, often a portion of the premium, if the underlying asset price reaches the deactivating barrier. In exchange, the premium paid by the investor is higher than if there were no rebate.

Barrier options, which have an additional condition that must be met before the option can be exercised, are in practice harder to price than vanilla options. The presence of a "barrier" makes the valuation of the option more complex because it requires a deeper analysis of possible variations in the price of the underlying asset. Moreover, multiple definitions of barriers (simple, double, continuous, discrete) require the use of more advanced mathematical techniques, such as stochastic differential equations, to accurately model the variations of the underlying asset.

The attractiveness of barrier options and the greater complexity of pricing a barrier option justify the importance of studying barrier option pricing.

As part of our option theory course, we are already studying stochastic modeling and vanilla option pricing. The main objective of our research is to understand the pricing mechanism of barrier options and to carry out an application of it.

In the remainder of our research paper, we will first understand the principles of stochastic modeling for barrier options and present a stochastic model specific to barrier option pricing. We will then focus on the principles of barrier option pricing, particularly the Monte Carlo method used to price barrier options, which we will apply in the stochastic model mentioned above. To go further, we will apply the binomial tree method to price both European and American barrier options.

I Literature review

I.1 Presentation of the main concepts related to barrier option pricing

As introduced in the previous section, a barrier option is a financial instrument that activates or deactivates depending on whether or not an underlying asset reaches a pre-determined price level, called a barrier. Barrier options differ from traditional options because of their dependence on the barrier, making their valuation more complex.

I.2 Analysis of existing methods for pricing barrier options

Several methods have been developed to price barrier options, each with its own advantages and disadvantages. The main methods include analytical, numerical, and simulation-based methods.

To begin with, analytical methods are based on mathematical equations derived to calculate the option price. The most famous method is the Black-Scholes formula, which is used to price European options. The Black-Scholes approach to modeling option prices is based on the use of stochastic differential calculus. This model assumes that the movement of the stock price follows a process called geometric Brownian motion, where price changes can reach infinite values. The return on the stock, on the other hand, follows a process called the generalized Wiener process, a stochastic process in continuous time representing the random evolution of a variable over time. However, analytical methods cannot be used directly to evaluate barrier options precisely because of their dependence on the barrier.

Numerical methods are used to solve complex mathematical equations that cannot be solved analytically. Two of the most common numerical methods are the finite difference method and the Monte Carlo method. The first method approximates the derivatives of a function using finite differences. In the case of option pricing, it is used to discretize space and time into a grid of points and solve the Black-Scholes equation, which describes the evolution of the option price as a function of time and the price of the underlying asset. This allows for the rapid calculation of option prices for a large number of parameter combinations, but requires fine discretization to obtain accurate results. The Monte Carlo method consists of simulating a large number of random paths for an underlying asset and calculating the option value from the average of these paths. This second method is often used in conjunction with stochastic modeling of the underlying asset.

Simulation-based methods are used to directly simulate the path of the underlying asset using mathematical models. The Brownian bridge method is often used to value barrier options.

In summary, there are different methods for pricing barrier options, each with its own advantages and disadvantages. The valuation of barrier options is a complex subject because of the dependence on the barrier. Traditional analytical methods cannot be used

directly to price barrier options, which has necessitated the development of specific numerical methods, while numerical and simulation-based methods are used to solve complex mathematical equations that cannot be solved analytically. The Monte-Carlo method stands out as the most commonly used numerical method in barrier option pricing.

In the following section, we will present the basic principles of stochastic modeling in finance and their application to barrier options.

II Stochastic modeling of barrier options

II.1 Presentation of the basic principles of stochastic modeling in finance

Stochastic modeling in finance is an essential approach for studying uncertain movements in financial asset prices. Stochastic models are used to describe and analyze asset price fluctuations over time by taking into account the random nature of price movements. Common stochastic models used in finance include the geometric Brownian motion, the Merton jump model and the Heston stochastic volatility model.

Barrier options are exotic options whose life and value depend not only on the value of the underlying asset, but also on the occurrence of a certain price event in the underlying asset. Specifically, the barrier is a predetermined price level that, once reached, activates or deactivates the option.

To model barrier options, it is crucial to take into account the stochastic nature of the underlying asset price, as well as the existence of the barrier. Commonly used stochastic models for barrier options include the geometric Brownian motion with jumps, the stochastic volatility model with jumps, and regime-switching models.

II.2 Application of stochastic modeling to barrier options

The payoff of an up-and-out barrier put option on the underlying asset price S_t with exercise date T , strike price K and barrier level (or call level) B is

$$C = (K - S_T)_+ \mathbb{1}_{\left\{\max\limits_{0\leqslant t\leqslant T} S_t < B\right\}} = \begin{cases} (K - S_T)_+ & \text{if } \max\limits_{0\leqslant t\leqslant T} S_t < B \\ 0 & \text{if } \max\limits_{0\leqslant t\leqslant T} S_t \geqslant B. \end{cases}$$

This option is also called a *Callable Bear Contract*, or a Bear CBBC with no residual value, or a turbo warrant with no rebate, in which the call level B usually satisfies $B \leqslant K$

The table above shows the payoff expressions for barrier call and barrier put options according to the different barrier types. This table is very useful to quickly understand the influence of barriers on payoffs, we will use it during the implementation in Python.

Option type	CBBC	Barrier type	Payoff
Barrier call	Bull	down-and-out (knock-out)	$(S_T - K)_+ \mathbb{1}_{\left\{\min\limits_{0\leqslant t\leqslant T} S_t > B\right\}}$
		down-and-in (knock-in)	$(S_T - K)_+ \mathbb{1}_{\left\{\min\limits_{0\leqslant t\leqslant T} S_t < B\right\}}$
		up-and-out (knock-out)	$(S_T - K)_+ \mathbb{1}_{\left\{\max\limits_{0\leqslant t\leqslant T} S_t < B\right\}}$
		up-and-in (knock-in)	$(S_T - K)_+ \mathbb{1}_{\left\{\max\limits_{0\leqslant t\leqslant T} S_t > B\right\}}$
Barrier put		down-and-out (knock out)	$(K - S_T)_+ \mathbb{1}_{\left\{\min\limits_{0\leqslant t\leqslant T} S_t > B\right\}}$
		down-and-in (knock-in)	$(K - S_T)_+ \mathbb{1}_{\left\{\min\limits_{0\leqslant t\leqslant T} S_t < B\right\}}$
	Bear	up-and-out (knock-out)	$(K - S_T)_+ \mathbb{1}_{\left\{\max\limits_{0\leqslant t\leqslant T} S_t < B\right\}}$
		up-and-in (knock in)	$(K - S_T)_+ \mathbb{1}_{\left\{\max\limits_{0\leqslant t\leqslant T} S_t > B\right\}}$

Table 1: Barrier option types.

II.3 The Heston model

Market observation shows the need to model volatility as a random variable. Indeed, one need only recall the existence of stock market crashes to be convinced of this.

In the Heston model, a mean-reverting parameter of volatility appears. It is economically reasonable that there should be such a parameter insofar as we can assume that, even in the long term, the volatility of a share will remain of a certain order of magnitude.

II.3.A Mathematical characteristics

The Heston (1993) model assumes that the stock price, S, and its variance, v, verify the following SDE system:

$$\begin{cases} dS_t = \mu_t S_t dt + \sqrt{v_t} S_t dW_1 \\ dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t} dW_2 \end{cases} \tag{II.1}$$

with

$$\langle dW_1, dW_2 \rangle = \rho dt$$

where μ_t is the instantaneous (deterministic) trend of the stock, κ, the speed of reversion to the mean, θ the variance at infinity, σ the volatility of the volatility and ρ the correlation of the two Brownians W_1 and W_2.

In the remainder of this section, we have largely adopted the approach of Xu Jianqiang [1]. In the Black-Scholes case, the only source of uncertainty comes from the stock price, which can be hedged with the stock. In the Heston case, the uncertainty arising from the stochastic nature of the volatility must also be hedged to create a risk-free portfolio. Let us imagine a portfolio Π containing the option whose price we are trying to determine

noted $V(S,v,t)$, the quantity $-\Delta$ of stock and the quantity $-\Delta_1$ of another asset, of value V_1 depending on the volatility. We thus have:

$$\Pi = V - \Delta S - \Delta_1 V_1 \tag{II.2}$$

We obtain, with the third formula of Itô:

$$\begin{aligned} d\Pi = &\left\{\frac{\partial V}{\partial t} + \frac{1}{2}vS_t^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma v S_t\frac{\partial^2 V}{\partial v \partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2}\right\}dt \\ -\Delta_1&\left\{\frac{\partial V_1}{\partial t} + \frac{1}{2}vS_t^2\frac{\partial^2 V_1}{\partial S^2} + \rho\sigma v S_t\frac{\partial^2 V_1}{\partial v \partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V_1}{\partial v^2}\right\}dt \\ &+\left\{\frac{\partial V}{\partial S} - \Delta_1\frac{\partial V_1}{\partial S} - \Delta\right\}dS + \left\{\frac{\partial V}{\partial v} - \Delta_1\frac{\partial V_1}{\partial v}\right\}dv \end{aligned} \tag{II.3}$$

For the portfolio to be risk-free, it is necessary to eliminate the terms in dS and dv, from which the quantities:

$$\begin{cases} \Delta = \frac{\partial V}{\partial S} - \frac{\partial V/\partial v}{\partial V_1/\partial v}\frac{\partial V_1}{\partial S} \\ \Delta_1 = \frac{\partial V/\partial v}{\partial V_1/\partial v} \end{cases} \tag{II.4}$$

Since the return on a risk-free portfolio must be equal to the risk-free rate r (assumed constant) - otherwise there would be an arbitrage opportunity, we have:

$$\begin{aligned} d\Pi &= \left\{\frac{\partial V}{\partial t} + \frac{1}{2}vS_t^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma v S_t\frac{\partial^2 V}{\partial v \partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2}\right\}dt \\ &\quad - \Delta_1\left\{\frac{\partial V_1}{\partial t} + \frac{1}{2}vS_t^2\frac{\partial^2 V_1}{\partial S^2} + \rho\sigma v S_t\frac{\partial^2 V_1}{\partial v \partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V_1}{\partial v^2}\right\}dt \\ &= r\Pi dt \\ &= r(V - \Delta S - \Delta_1 V_1)dt \end{aligned}$$

Using (II.4), the last equation can be rewritten:

$$\begin{aligned} &\frac{1}{\partial V/\partial v}\left\{\frac{\partial V}{\partial t} + \frac{1}{2}vS_t^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma v S_t\frac{\partial^2 V}{\partial v \partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} + rS\frac{\partial V}{\partial S} - rV\right\} = \\ &\frac{1}{\partial V_1/\partial v}\left\{\frac{\partial V_1}{\partial t} + \frac{1}{2}vS_t^2\frac{\partial^2 V_1}{\partial S^2} + \rho\sigma v S_t\frac{\partial^2 V_1}{\partial v \partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V_1}{\partial v^2} + rS\frac{\partial V_1}{\partial S} - rV_1\right\} \end{aligned} \tag{II.5}$$

The left-hand member is a function only of V while the right-hand member is a function only of V_1. This forces each of the two members to be equal to a function f of the independent variables S, v and t. We can thus write that:

$$\frac{\partial V}{\partial t} + \frac{1}{2}vS_t^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma v S_t\frac{\partial^2 V}{\partial v \partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} + rS\frac{\partial V}{\partial S} - rV = \frac{\partial V}{\partial v}f \tag{II.6}$$

Or

$$\frac{\partial V}{\partial t} + \frac{1}{2}vS_t^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma v S_t\frac{\partial^2 V}{\partial v \partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} + rS\frac{\partial V}{\partial S} - rV = \frac{\partial V}{\partial v}\left[\kappa(\theta - v) - \Lambda(S,v,t)\sigma\sqrt{v}\right] \tag{II.7}$$

where $\Lambda(S, v, t)$ is the market price of volatility.
In his paper, Heston chooses $\Lambda(S, v, t) = \frac{\lambda\sqrt{v}}{\sigma}$, so that:

$$\frac{\partial V}{\partial t} + \frac{1}{2}vS_t^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma v S_t\frac{\partial^2 V}{\partial v \partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} + rS\frac{\partial V}{\partial S} - rV - \frac{\partial V}{\partial v}\Big[\kappa(\theta - v) - \lambda v\Big] = 0 \tag{II.8}$$

II.3.B Expression of analytical solutions for vanilla products

The mathematical expressions we will see in this subsection will be used to calibrate the Heston model.

We now seek to solve II.8 for a European call of strike K. We assume that the solution we are looking for can be written as:

$$C(S, v, t) = SP_1 - Ke^{-r(T-t)}P_2 \tag{II.9}$$

Let $x = ln(S)$ and $\tau = T - t$. II.9 can therefore be rewritten as:

$$C(X, v, \tau) = e^X P_1 - Ke^{-r\tau}P_2$$

where

– P_1 is the delta of the European call option

– P_2 is the conditional risk neutral probability that the asset price will be greater than K at the maturity.

Injecting II.9 into II.8, we obtain for $j \in \{1, 2\}$:

$$\frac{\partial P_j}{\partial t} + \frac{1}{2}v\frac{\partial^2 P_j}{\partial x^2} + \rho\sigma v\frac{\partial^2 P_j}{\partial x \partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 P_j}{\partial v^2} + (r + u_j v)\frac{\partial P_j}{\partial x} + (a - b_j v)\frac{\partial P_j}{\partial v} = 0 \tag{II.10}$$

where

$$u_1 = \frac{1}{2},\ u_2 = -\frac{1}{2},\ a = \kappa\theta,\ b_1 = \kappa + \lambda - \rho\sigma,\ b_2 = \kappa + \lambda$$

On the other hand, since it is a call, we have the following boundary conditions:

$$P_j(x, v, T) = \mathbb{1}_{\{x \geqslant ln(K)\}} \tag{II.11}$$

The terms P_j can be interpreted as probabilities: The corresponding characteristic functions φ_1 and φ_2 satisfy the PDE:

$$\frac{\partial \varphi_j}{\partial t} + \frac{1}{2}v\frac{\partial^2 \varphi_j}{\partial x^2} + \rho\sigma v\frac{\partial^2 \varphi_j}{\partial x \partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 \varphi_j}{\partial v^2} + (r + u_j v)\frac{\partial \varphi_j}{\partial x} + (a - b_j v)\frac{\partial \varphi_j}{\partial v} = 0 \tag{II.12}$$

with the terminal condition

$$\varphi_j(x, v, t, \Phi) = e^{i\Phi x}$$

We are therefore looking for a solution in the form:

$$\varphi_j(x, v, t, \Phi) = exp(C(\tau, \Phi) + D(\tau, \Phi)v + i\Phi x) \tag{II.13}$$

Injecting into II.12, we get:

$$-\frac{\Phi^2}{2} + \rho\sigma\Phi i D + \frac{1}{2}\sigma^2 D^2 + u_j \Phi i - b_j D + \frac{\partial D}{\partial t} = 0 \tag{II.14}$$

$$r\Phi i + aD + \frac{\partial C}{\partial t} = 0 \tag{II.15}$$

with $C(0,\Phi) = D(0,\Phi) = 0$. We can solve these ODEs:

$$C(\tau,\Phi) = r\Phi i\tau + \frac{a}{\sigma^2}\left[(b_j - \rho\sigma\Phi i + d)\tau - 2ln(\frac{1-ge^{d\tau}}{1-g})\right] \tag{II.16}$$

$$D(\tau,\Phi) = \frac{b_j - \rho\sigma\Phi i + d}{\sigma^2}\left[\frac{1-e^{d\tau}}{1-ge^{d\tau}}\right] \tag{II.17}$$

with

$$g = \tfrac{b_j - \rho\sigma\Phi i + d}{b_j - \rho\sigma\Phi i - d} \text{ et } d = \sqrt{(\rho\sigma\Phi i - b_j)^2 - \sigma^2(2u_j\Phi i - \Phi^2)}$$

Finally, by inverse Fourier transformation, we obtain (cf [2]):

$$P_j = \frac{1}{2} + \frac{1}{\pi}\int_0^{+\infty} Re\left[\frac{e^{-i\Phi ln(K)}\varphi_j}{i\Phi}\right] d\Phi$$

II.3.C Improvement: The Heston model with jumps or Bates model

A common improvement of the Heston stochastic volatility model is to add jumps to the dynamics of the underlying, through the Merton model. The combination of these two models is called the Heston model with jumps or Bates model, and is written:

$$\begin{cases} dS_t = rS_t dt + \sqrt{v_t}S_t dW_1 + (e^J - 1)dN_t \\ dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t}dW_2 \end{cases} \tag{II.18}$$

with

$$\langle dW_1, dW_2\rangle = \rho dt$$

and N_t is a Poisson process with parameter λ and J follows a Gaussian distribution with mean μ_{jump} and variance δ_{jump}. In this model, there are now not five but eight parameters.

More generally, there are a large number of refinements that can be made: consider dividends, a random risk-free rate etc. Figure 1 gives an overview of some possible refinements. However, it should be kept in mind that the more parameters are introduced, the more difficult it will be to calibrate them correctly.

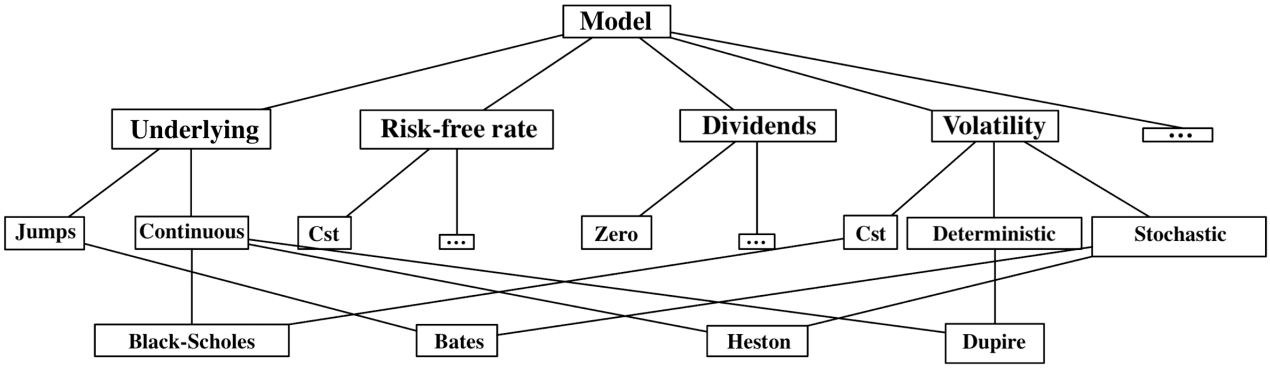

Figure 1: Structural tree of equity derivative models

In the rest of our work and in order to price barrier options we will use the Heston model with jumps. We will not deal with possible dividend payments, random risk free rate etc.

This choice stems from the fact that in practice the Heston model with jumps is widely used for barrier option pricing as it captures the key characteristics of barrier options by taking into account volatility variations and price jumps. Correlated Brownian movements allow to model the joint evolution of the volatility and the price of the underlying asset, while the Poisson process introduces price jumps which can be particularly relevant for the behaviour of barrier options.

III Pricing of barrier options with the Monte-Carlo method

Several methods have been developed for pricing barrier options. Among the most common methods, we find:

The analytical method: This approach relies on the exact solution of the PDE (Partial Differential Equation) associated with the barrier option. However, analytical solutions are often difficult to obtain due to the complexity of the PDE and the conditional nature of the option.

The finite difference method: This method consists of discretising the PDE and solving the resulting system of equations using a numerical algorithm. The finite difference method is widely used for pricing barrier options, but it can be costly in terms of computation time and memory, especially for complex stochastic models.

Finite element method: Similar to the finite difference method, this approach involves discretising the PDE and solving the resulting system of equations. The finite element method is also suitable for pricing barrier options, but it can be costly in terms of computation time and memory.

Monte Carlo method: This method is based on the simulation of price paths of underlying assets and the estimation of the option value from these paths. The Monte Carlo method is very flexible and can be applied to complex stochastic models. However, it can also be expensive in terms of computational time and require a large number of simulations to obtain accurate estimates.

In the following sections, we have opted for the Monte Carlo method for pricing barrier options and will explore its application in the stochastic Heston model with jumps. We have chosen to use the Python programming language to implement our barrier option pricing tool. Indeed, this language is widely used in the field of quantitative finance.

III.1 Barrier option pricing in Python

Since barrier options are sensitive to the trajectory of the underlying asset, we need to simulate price paths taking into account stochastic volatility and price jumps. To do so, we use the Euler-Maruyama method to discretise the stochastic differential equations of the Heston model with jumps.

To implement this method in Python, we proceed as follows:

- Initialize the parameters of the Heston model with jumps, such as expected return, long-run volatility, rate of reversion to the mean, volatility of volatility, correlation between Brownian movements and parameters of the jump process.
- Generate price trajectories using the *Euler-Maruyama method* to discretise the stochastic differential equations. For each trajectory, simulate correlated Brownian motion and the Poisson process, and update prices and volatility according to

these realisations. For each price path, check whether the barrier is reached and, if so, calculate the value of the barrier option using the appropriate payoff formula (call or put, knock-in or knock-out).

- Estimate the barrier option price by averaging the option values obtained for all simulated paths and then discounting this average at the risk-free interest rate.
- In order to improve the accuracy of the estimation, we propose to use variance reduction techniques, such as preferential sampling and antithetical variable. Preferential sampling involves simulating price paths from a modified distribution that gives more weight to paths that are likely to cross the barrier. The antithetical variable involves generating pairs of opposing paths and using the average of these pairs to estimate the option price, thereby reducing the estimation error.

By applying these techniques to the Monte Carlo method, we obtained more accurate estimates of the barrier option price in the stochastic Heston model with jumps. This demonstrates that our improved Monte Carlo approach is effective for pricing barrier options in complex stochastic models, while offering advantages in terms of variance reduction and computational efficiency.

III.2 Step-by-step implementation

In this section we will detail the steps mentioned above.

III.2.A Initialization of parameters

For the initialization of the parameters we have chosen:

$$\begin{cases} \kappa = 1.5738 \\ \theta = 0.552 \\ \sigma = 0.3 \\ \rho = -0.5711 \\ v_0 = 0.33 \\ \lambda = 0.575 \\ \mu_{jump} = -0.06 \\ \delta_{jump} = 0.1 \end{cases}$$

We chose these values because we wanted the initial values to be well calibrated. We therefore used market data that was similar to our simulation thanks to the open-source reports on the website BlackRock.

In section III.4.D, we will discuss the efficiency of this calibration and the influence of the number of iterations.

III.2.B The Euler-Maruyama method for discretization

Then we had to generate correlated Brownian motions and a Poisson process. For this we proceeded as follows:

```
W1 = np.random.normal(size=time_steps) * sqrt_dt
W2 = np.random.normal(size=time_steps) * sqrt_dt
```

```
Z = rho * W1 + np.sqrt(1 - rho**2) * W2
N = np.random.poisson(lamb * dt, size=time_steps)
```

After generating the correlated Brownian motion and the Poisson process, we decided to use the Euler-Maruyama method to discretise the stochastic differential equations.

Here are some of the reasons why we chose this method:

- **Simplicity**: the Euler-Maruyama method is a relatively simple method and easy to implement. It is based on Euler's method for ordinary differential equations (ODEs) and consists of discretising time into regular intervals and then approximating the evolution of the stochastic process using a sum of increments.
- **Convergence**: under certain conditions, the Euler-Maruyama method offers convergence in probability to the exact solution of the SDE when the time step tends to zero. This convergence is generally of order 0.5, which means that the approximation error decreases at a rate proportional to the square root of the time step.
- **Flexibility**: The Euler-Maruyama method can be applied to a wide variety of stochastic models, including stochastic volatility models and jump models, which is exactly the case here. It can also be combined with other numerical techniques, such as Monte Carlo methods, to estimate prices of barrier options and other derivatives.
- **Compatibility**: The Euler-Maruyama method is compatible with various boundary conditions, which makes it suitable for pricing barrier options.

In order to have an objective look at our approach, here is the main limitation we attribute to the Euler-Maruyama method: it may give less accurate results for relatively large time steps and very irregular stochastic differential equations. In such cases, more advanced discretisation methods, such as the Milstein method or higher-order schemes, can be used to obtain better accuracy. We have therefore adapted the time steps of our model accordingly to make the discretisation more reliable.

With the Euler-Maruyama method, the discretization formulas for our S_t and v_t processes are:

$$\begin{cases} S_{t+1} = S_t e^{(r-\frac{1}{2}v_t)dt} + \sqrt{v_t} W_2^t + \mu_{jump} N_t - \frac{1}{2}\delta_{jump}^2 N_t \\ v_{t+1} = max(0, v_t + \kappa(\theta - v_t)dt + \sigma\sqrt{v_t} Z_t \end{cases}$$

where $Z = \rho W_1 + \sqrt{1-\rho^2} W_2$

We based our implementation on the Heston model with jumps or Bates model that we saw in II.3.C, taking equation (II.18), J is a random variable that follows a Gaussian distribution with a mean μ_{jump} and a standard deviation δ_{jump}. The variable N_t is a Poisson counting process that models the arrival of jumps.

In our configuration, μ_{jump} is the log mean of the jumps, and δ_{jump} is the log standard deviation of the jumps. When a jump occurs (i.e. when $dN_t = 1$), the jump factor is e^J, and the price change of the underlying is $(e^J - 1)S_t$. Thus, μ_{jump} and δ_{jump} determine

the size and variability of jumps in the Heston model with jumps.

Here is the code python for the Euler-Maruyama discretisation:

```
St = np.zeros(time_steps + 1)
vt = np.zeros(time_steps + 1)
St[0] = S0
vt[0] = v0
for t in range(time_steps):
    St[t + 1] = St[t] * np.exp((r - 0.5 * vt[t]) * dt +
                np.sqrt(vt[t]) * W1[t] + mu_jump * N[t] -
                0.5 * delta_jump ** 2 * N[t])
    vt[t + 1] = np.maximum(0, vt[t] + kappa * (theta - vt[t]) * dt +
                sigma * np.sqrt(vt[t]) * Z[t])
```

III.2.C Barrier option price estimate

In this section, we will use the table on types of barrier options that we made earlier.

In order to determine the final price of the barrier option we need to determine whether a disabling barrier has been breached (in which case the option will be void) or whether an activating barrier has been breached (in which case the option is still running). We also calculate the final payoff, which we add to the table containing all the simulated option prices.

Here is the corresponding python code:

```
option_values,option_value = [], 0

if option_type == 'call':
    if barrier_type == 'up-and-out':
        if np.any(St >= barrier):
            option_value = 0
            barrier_crossed.append(True)
        else:
            option_value = np.exp(-r * T) * max(St[-1] - K, 0)
            barrier_crossed.append(False)
    elif barrier_type == 'up-and-in':
        if np.any(St >= barrier):
            option_value = np.exp(-r * T) * max(St[-1] - K, 0)
            barrier_crossed.append(False)
        else:
            option_value = 0
            barrier_crossed.append(True)
elif option_type == 'put':
    if barrier_type == 'down-and-out':
        if np.any(St <= barrier):
            option_value = 0
            barrier_crossed.append(True)
        else:
```

```
                option_value = np.exp(-r * T) * max(K - St[-1], 0)
                barrier_crossed.append(False)
            elif barrier_type == 'down-and-in':
                if np.any(St <= barrier):
                    option_value = np.exp(-r * T) * max(K - St[-1], 0)
                    barrier_crossed.append(False)
                else:
                    option_value = 0
                    barrier_crossed.append(True)

        option_values.append(option_value)
```

You may notice that we have introduced a variable to determine whether the deactivating barrier has been breached or whether the activating barrier has not been breached, which in both cases makes the option value zero.

This variable will be particularly useful when we display the prices of the assets by differentiating them.

III.2.D Variance reduction with antithetical variables

We have already mentioned the many advantages of the Monte Carlo method which led us to choose this method. However, it should be kept in mind that this method is often time-consuming and also requires attention to variance reduction.

The most common methods for dealing with variance reduction are:

- antithetical variables
- control variables
- Latin square
- stratification
- Preferential sampling.

In particular, the idea behind the antithetic variables method is to take advantage of the symmetry of the underlying distributions to reduce the variance of the estimates.

There are several reasons for using this method in our work:

- **Exploiting symmetry**: Antithetical variables take advantage of the symmetry of the underlying distributions. In the Heston model with jumps, the distribution of returns is symmetrical, which allows this method to be applied.
- **Variance reduction**: The use of antithetic variables reduces the variance of the estimates. This results in more accurate option price estimates and faster convergence of the Monte Carlo simulation.
- **Ease of implementation**: The antithetical method is relatively simple to implement in a Monte Carlo simulation. It usually requires only minor code modifications to generate pairs of antithetical paths and calculate the corresponding average prices.

- **Efficiency**: The antithetical method is often more efficient than other variance reduction techniques, as it directly exploits the symmetric structure of the underlying distributions. As a result, it can provide more accurate estimates with fewer simulations.

The idea of antithetical control is very simple. It is based on the symmetry property of Brownian motion: $W \sim -W$ in distribution, therefore:

$$\mathbb{E}\left[\frac{F(W_t) + F(-W_t)}{2}\right] = \mathbb{E}[F(W_t)].$$

The variance is reduced if:

$$Var\left[\frac{F(W_t) + F(-W_t)}{2}\right] < \frac{Var[F(W_t)]}{2},$$

which is the same as saying

$$Cov(F(W_t), F(-W_t)) < 0.$$

This method is not only very simple but can reduce the computational time as the random generator is used in half the time. $N/2$ random numbers are generated according to the $\mathcal{N}(0,1)$ distribution and the other $N/2$ are obtained by reversing the sign of the first two, N being the Monte Carlo sample size.

Figure 2 shows the effect of using the antithetical variables technique on the variance of the valuation of a vanilla option.

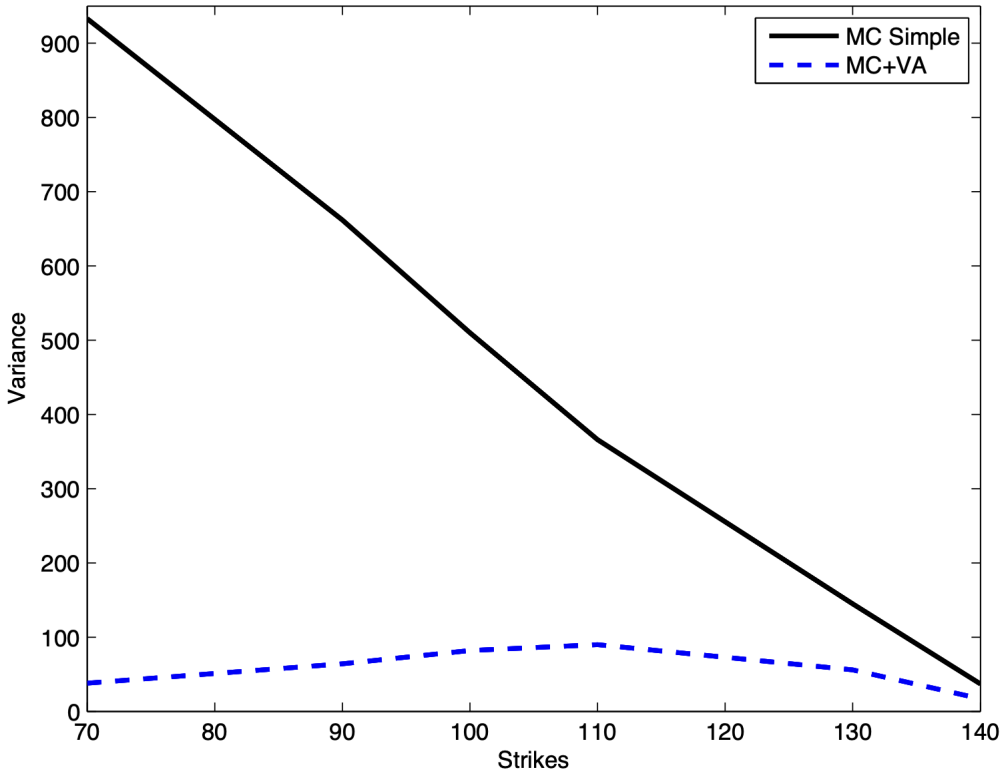

Figure 2: Effect of the antithetical variables technique on the variance of the Monte Carlo valuation of a vanilla option

The reduction in variance is very clear and this is confirmed by the shape of the simulated trajectories (see Figure 3).

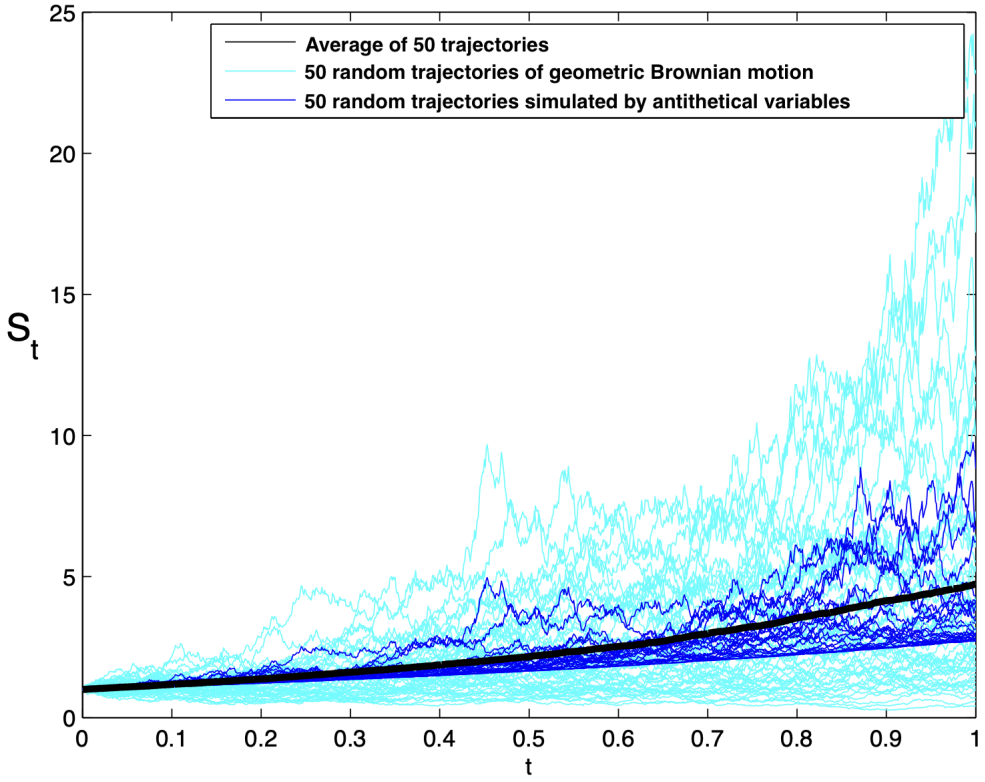

Figure 3: Effect of the antithetical variable technique on Monte Carlo simulated trajectories

III.3 Results of the implementation

We will now present our results for the pricer of barrier options with the Monte-Carlo method.

III.3.A Example of a CALL

Let's take the example of a CALL barrier option of strike $K = \$80$ with an up-and-out barrier of \$120. The price of the underlying asset at date $t = 0$ is $S_0 = \$100$.

Figures 4 to 6 are the graphical results of the modeling to price the CALL barrier option. For the sake of visibility we have chosen to present the graphs for 30 simulations. In reality, in order for the estimated price to be accurate, we can simulate up to 50,000 trajectories as shown in figure 7.

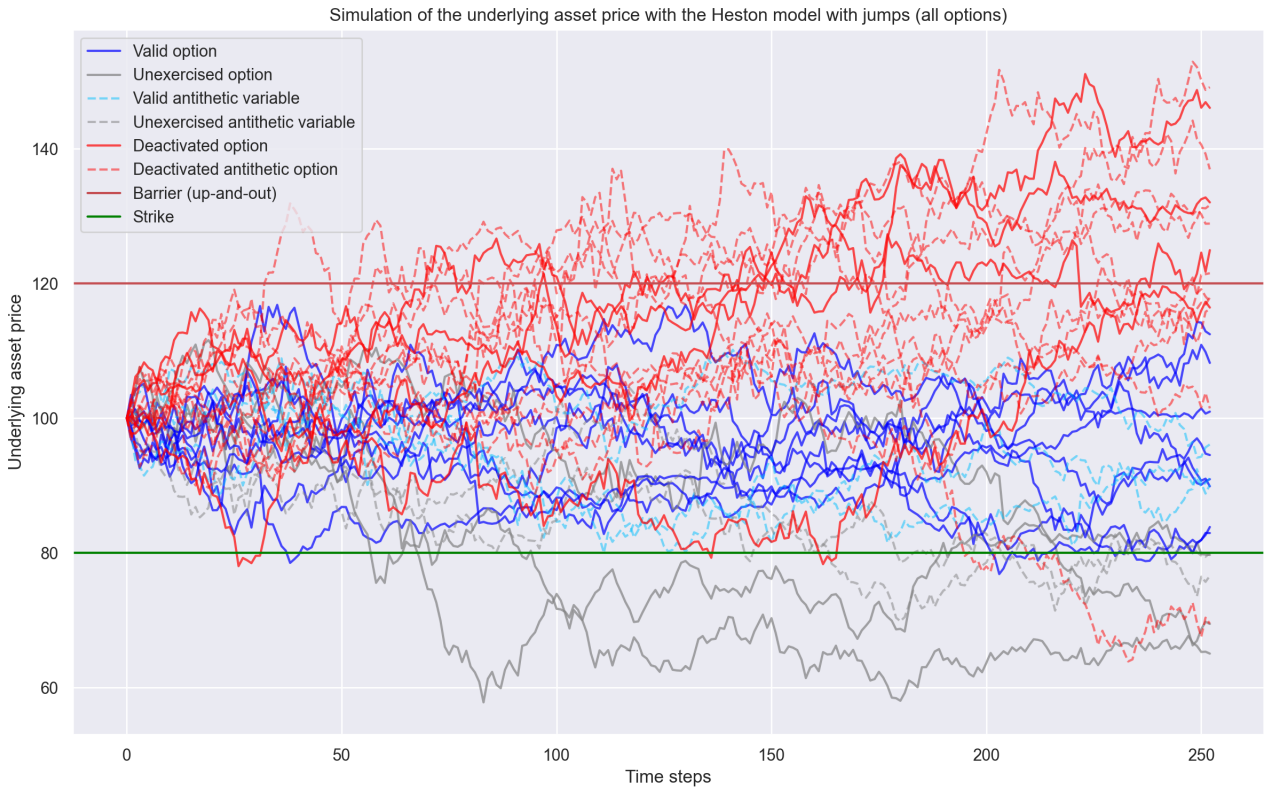

Figure 4: Simulation of the underlying asset price with the Heston model with jumps (all options)

Here we can see the different cases in which the option can end up at maturity.

It can be (as seen by the red paths and in figure 5) deactivated. The option is deactivated as soon as it has crossed a deactivating barrier. The dashed paths represent the antithetical paths used for variance reduction (see III.2.D).

When the option holder is in a favourable position at expiration, he can exercise his option (this case is represented by the blue paths in figure 6). Here, the options valid at expiration are those which, during the exercise, have not crossed a disabling barrier and which, at expiration, benefit from the fact that the price of the underlying is higher than the strike value.

Finally, the grey paths represent options that are not exercised at expiration because the price of the underlying is below the strike.

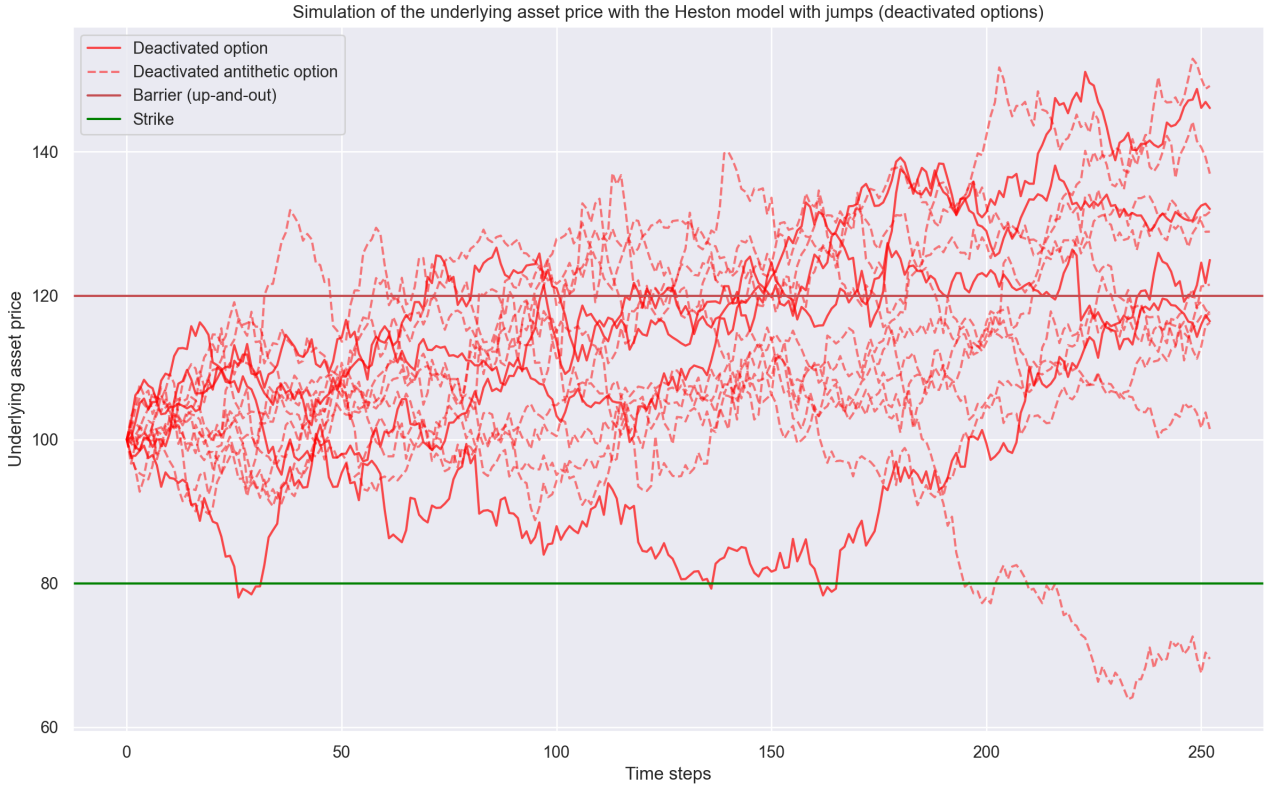

Figure 5: Simulation of the underlying asset price with the Heston model with jumps (deactivated options)

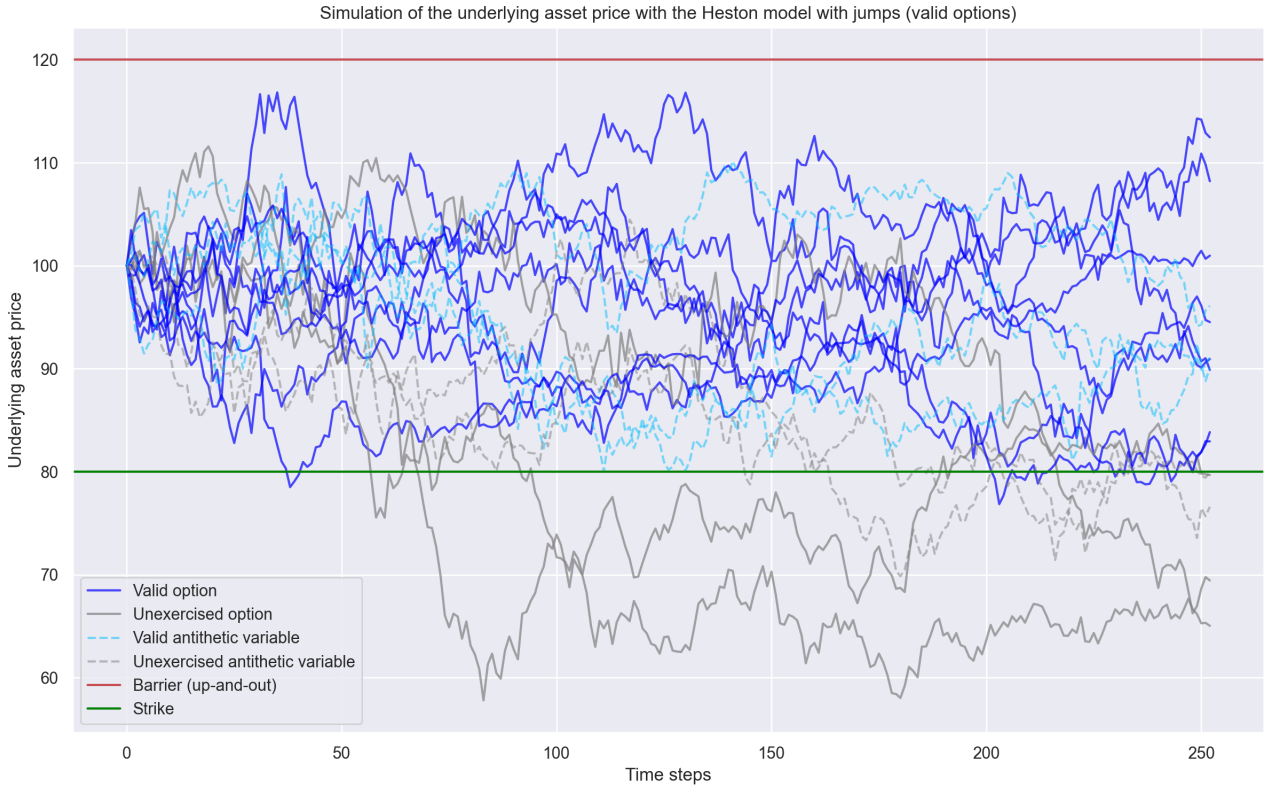

Figure 6: Simulation of the underlying asset price with the Heston model with jumps (valid options)

Figures 7 shows the influence of the number of simulations on the option price. We notice that the price of our option is fixed after 40000 iterations. We can finally see that in the case of a CALL barrier option with a strike of \$80 and an up-and-out barrier of \$120, knowing that $S_0 = \$100$. The estimated price of the barrier option by our pricer is \$6.134.

Figure 7: Influence of the number of simulations on the CALL option price

```
Running script: "/Users/theojalabert/Desktop/TER.py"
Estimated barrier option price : 6.13409832001982
```

III.3.B Example of a PUT

We have seen the results of our pricer in the case of a CALL. Let's now take the case of a PUT barrier option of strike $K = \$110$ with an down-and-out barrier of \$70. The price of the underlying asset at date $t = 0$ is $S_0 = \$100$.

With figure 8 we can see the different cases in which the option can end up at maturity.

It can be (as seen by the red paths and in figure 9) deactivated. The option is deactivated as soon as it has crossed a deactivating barrier. The dashed paths represent the antithetical paths used for variance reduction (see III.2.D).

When the option holder is in a favourable position at expiration, he can exercise his option (this case is represented by the blue paths in figure 10). Here, the options valid at expiration are those which, during the exercise, have not crossed a disabling barrier and which, at expiration, benefit from the fact that the price of the underlying is lower than the strike value.

Finally, the grey paths represent options that are not exercised at expiration because the price of the underlying is above the strike.

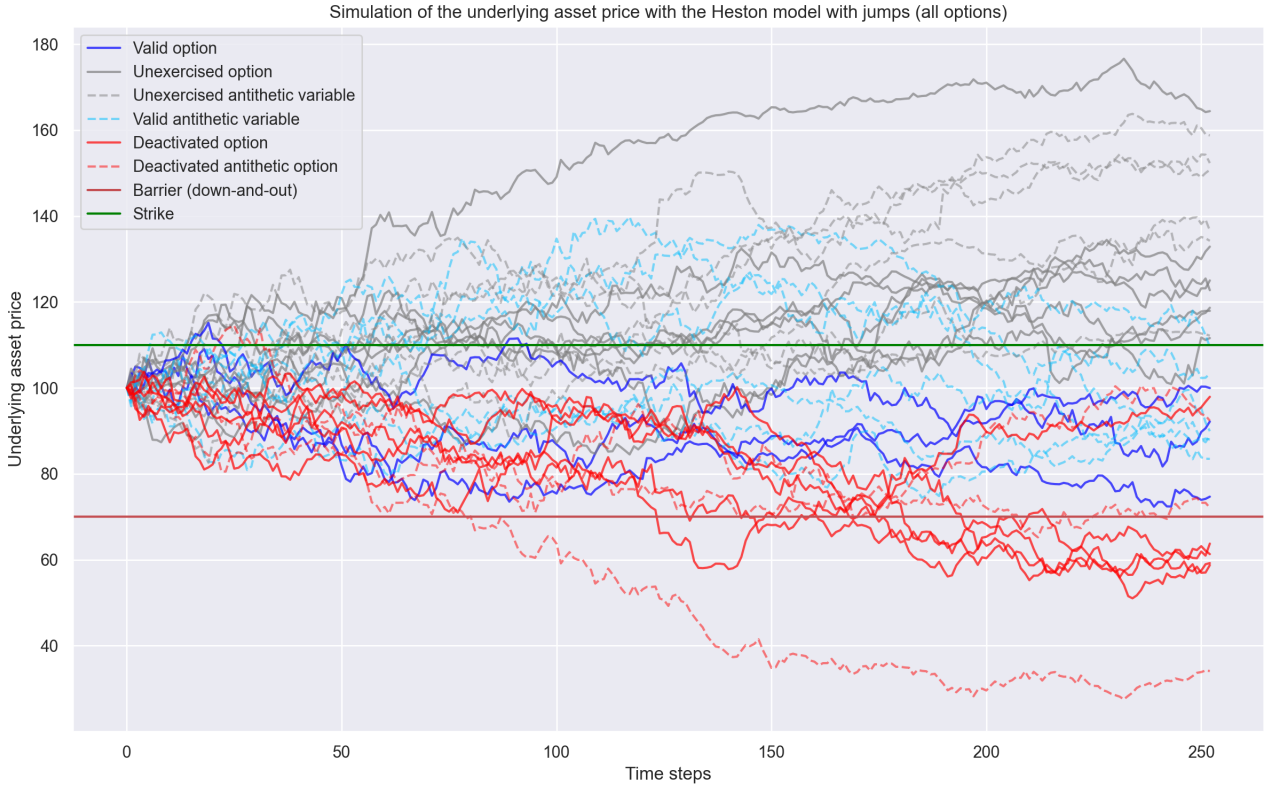

Figure 8: Simulation of the underlying asset price with the Heston model with jumps (all options)

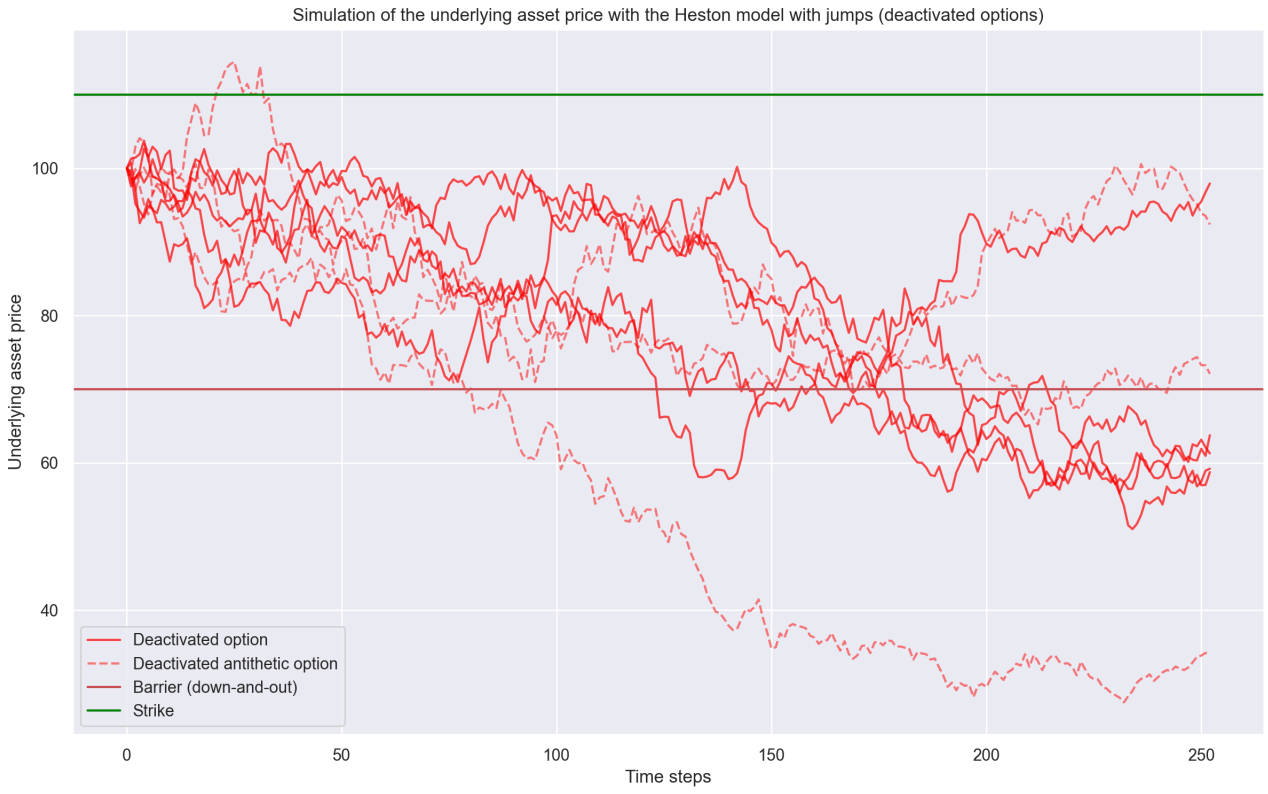

Figure 9: Simulation of the underlying asset price with the Heston model with jumps (deactivated options)

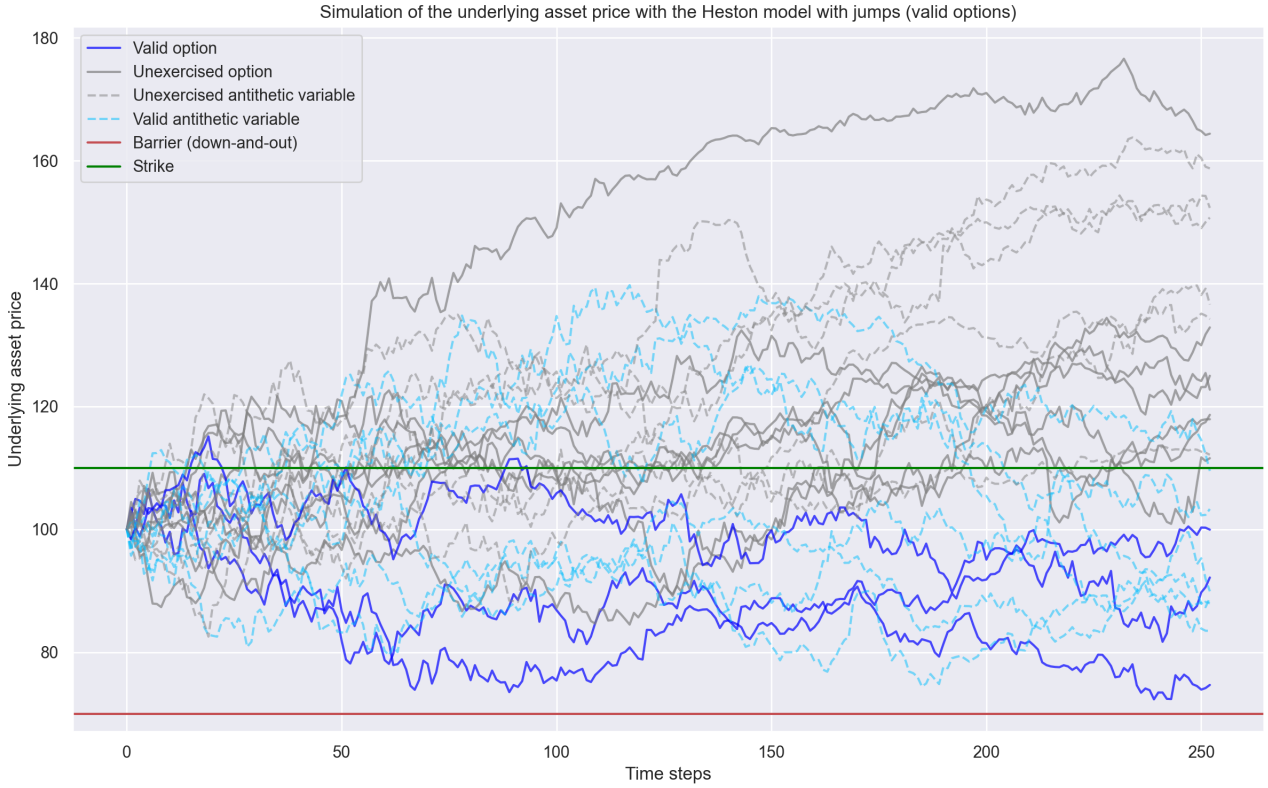

Figure 10: Simulation of the underlying asset price with the Heston model with jumps (valid options)

Finally, as in the case of CALL, we have analysed the influence of the number of simulations on the PUT option price (see figure 11). We notice that the price of our option is fixed after 35000 iterations. We can finally see that in the case of a PUT barrier option with a strike of \$110 and an down-and-out barrier of \$70, knowing that $S_0 = \$100$. The estimated price of the barrier option by our pricer is \$6.163.

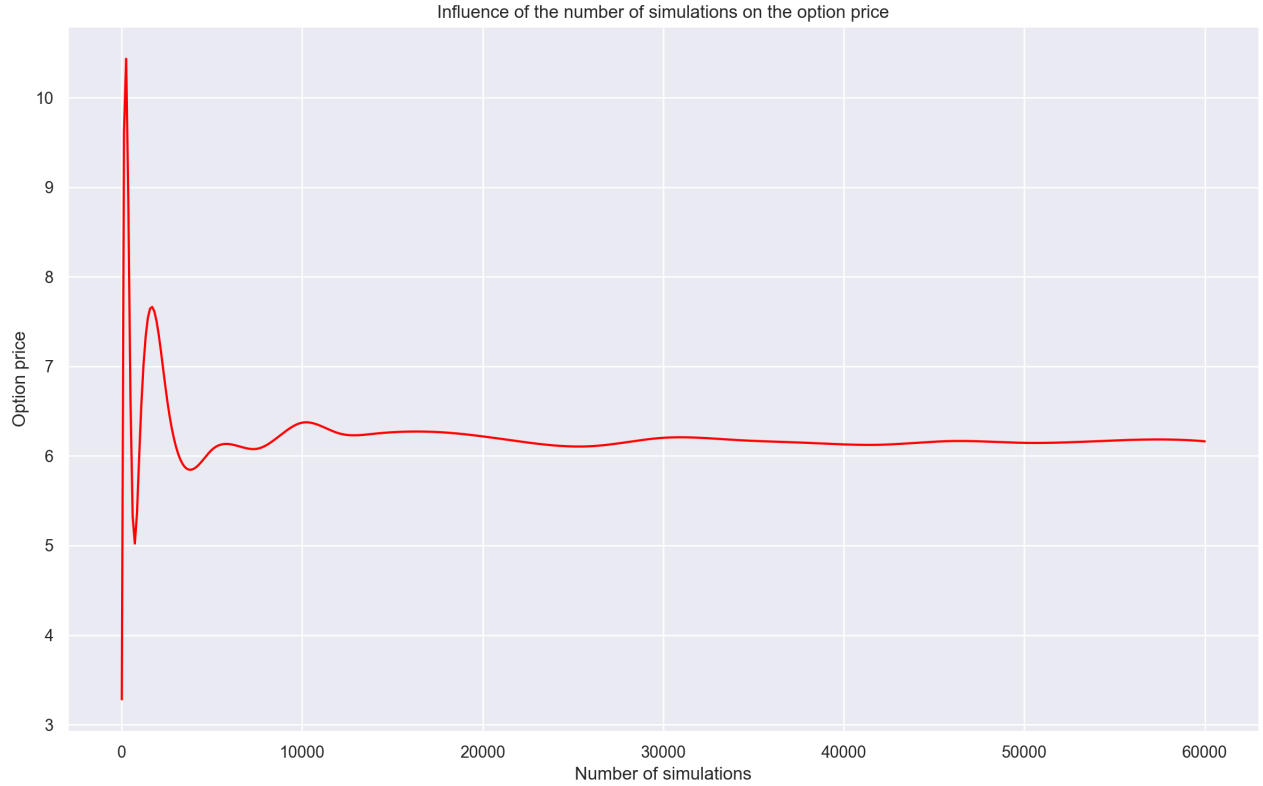

Figure 11: Influence of the number of simulations on the PUT option price

```
Running script: "/Users/theojalabert/Desktop/TER.py"
Estimated barrier option price : 6.162786425219842
```

Now that we have built our barrier option pricer, we need to determine if the estimated prices are true to the market. To do this, we will calibrate our model in the following section.

III.4 Model calibration on S&P500 data

In this section we will look at the calibration of our model. But what motivates us to calibrate our model?

In practice, we actually don't use the pricer to compute prices. Instead, we attempt to determine the parameters for the Heston model based on the observed market prices.

These parameters are employed in pricing complex/exotic options. For highly complex options, it's challenging to find a closed-form formula for pricing exotic options.

In such cases, we use plain vanilla options to determine the parameters of the Heston model and apply the parameters obtained from the market to calculate the price of exotic options using Monte Carlo methods.

We will therefore, as announced, use the analytical solutions for the vanilla products seen in II.3.B.

III.4.A Implementation of the characteristic function.

Injecting equations II.16 and II.17 into II.13 we obtain the following long form for the characteristic function:

$$\varphi(X_0, K, v_0, \tau; \Phi) = e^{r\Phi i\tau} S^{i\Phi} \left[\frac{1-ge^{d\tau}}{1-g}\right]^{\frac{-2a}{\sigma^2}} exp\left[\frac{a\tau}{\sigma^2}(b_2 - \rho\sigma\Phi i + d) + \frac{v_0}{\sigma^2}(b_2 - \rho\sigma\Phi i + d)\left[\frac{1-e^{d\tau}}{1-ge^{d\tau}}\right]\right]$$

where d and g no longer change with b_1, b_2 or u_1, u_2

- $d = \sqrt{(\rho\sigma\Phi i - b)^2 + \sigma^2(\Phi i + \Phi^2)}$
- $g = \frac{b - \rho\sigma\Phi i + d}{b - \rho\sigma\Phi i - d}$
- $a = \kappa\theta$
- $b = \kappa + \lambda$

Here is the python implementation of the characteristic function:

```
def heston_charfunc(S0, r, tau ,params, phi):

    kappa, theta, sigma, rho, lamb, v0 = params

    # Constants
    a = kappa*theta
    b = kappa+lamb

    # Common terms w.r.t phi
    rspi = rho*sigma*phi*1j

    # Define d parameter given phi and b
    d = np.sqrt( (rho*sigma*phi*1j - b)**2 + (phi*1j+phi**2)*sigma**2 )

    # Define g parameter given phi, b and d
    g = (b-rspi+d)/(b-rspi-d)

    # Calculate characteristic function by components
    exp1 = np.exp(r*phi*1j*tau)
    term2 = S0**(phi*1j) * ( (1-g*np.exp(d*tau))/(1-g) )**(-2*a/sigma**2)
    exp2 =  np.exp(a*tau*(b-rspi+d)/sigma**2 +
            v0*(b-rspi+d)*( (1-np.exp(d*tau))/(1-g*np.exp(d*tau)) )/sigma**2)

    return exp1*term2*exp2
```

III.4.B Perform numerical integration on the integrand and compute the option price.

First, we need to define the integrand $\int_0^{+\infty} Re\left[\frac{e^{-i\Phi ln(K)}\varphi_j}{i\Phi}\right] d\Phi$ as a function. To achieve this, we proceeded as follows:

We rewrote the integrand in this way:

$$\int_0^{+\infty} Re\Big[\frac{e^{-i\Phi ln(K)}\varphi_j}{i\Phi}\Big]d\Phi = \int_0^{+\infty} Re\Big[e^{r\tau}\frac{\varphi(\Phi - i)}{i\Phi K^{i\Phi}} - K\frac{\varphi(\Phi)}{i\Phi K^{i\Phi}}\Big]d\Phi$$

Then, we wrote the following code:

```
def integrand(S0, r, tau, params, phi):
    args = (S0, r, tau, params, phi)

    numerator = np.exp(r*tau)*heston_charfunc(args, phi-1j) -
                K*heston_charfunc(args, phi)
    denominator = 1j*phi*K**(1j*phi)

    return numerator/denominator
```

Using the equations from the section II.3.B, we know that:

$$C(S_0, K, v_0, \tau) = \frac{1}{2}(S_0 - Ke^{-r\tau}) + \frac{1}{\pi}\int_0^{+\infty} Re\Big[e^{r\tau}\frac{\varphi(\Phi - i)}{i\Phi K^{i\Phi}} - K\frac{\varphi(\Phi)}{i\Phi K^{i\Phi}}\Big]d\Phi$$

So using the principle of rectangular integration, we calculate the option price with the following code:

```
def heston_price_rec(S0, K, r, tau, params):
    args = (S0, r, tau, params)

    P, umax, N = 0, 100, 10000
    dphi=umax/N # dphi is width
    for i in range(1,N):
        # Rectangular integration
        phi = dphi * (2*i + 1)/2 # Midpoint to calculate height
        numerator = np.exp(r*tau)*heston_charfunc(args, phi-1j) -
                    K * heston_charfunc(args, phi)
        denominator = 1j*phi*K**(1j*phi)

        P += dphi * numerator/denominator

    return np.real((S0 - K*np.exp(-r*tau))/2 + P/np.pi)
```

Then we use the scipy package to gain precision:

```
def heston_price(S0, K, r, tau, params):
    args = (S0, r, tau, params)

    real_integral, err = np.real( quad(integrand, 0, 100, args=args) )

    return (S0 - K*np.exp(-r*tau))/2 + real_integral/np.pi
```

III.4.C Calibration with market option prices for S&P500 Index

We will now deal with calibration. As the calibration of such a model is a rather complex task, we were able to draw on the work of Emmerick Johnaton, a quantitative analyst. We had several choices for this part of the assignment, and we chose to test the calibration of our model on the S&P500 Index. To do so, we used the API of the website EOD Historical Data.

An API (Application programming interface) is a tool that allows our machine to exchange with the website in order to collect data.

We chose to use the API because it allows us to have reliable data that is up-to-date and true to the market.

Once the data was collected, we corrected it because, like all data sets, it can contain holes, biases etc...

The parameters to be determined by calibration with market prices are:
$\Theta = (v_0, \kappa, \theta, \sigma, \rho, \lambda)$

We decided to do a least squared error fit in the following way.

Let $(\tau_i)_{1 \geqslant i \geqslant M}$ be some times to maturities. The purpose of the calibration is to minimize the least squared error

$$SqErr(\Theta) = \sum_{i=1}^{N} \sum_{j=1}^{M} \omega_{ij} \Big[C_{MP}(K_i, \tau_j) - C_{SV}(S_\tau, K_i, \tau_j, r_j, \Theta) \Big]^2 + Penalty(\Theta, \Theta_0)$$

where $C_{MP}(K_i, \tau_j)$ represents the market price for a call with strike K_i and maturity τ_j. On the other hand, C_{SV} denotes the price calculated with our Heston model and the parameters Θ.

The penalty function, $Penalty(\Theta, \Theta_0) = \|\Theta - \Theta_0\|^2$, is the distance to the initial parameter vector.

We therefore have $\hat{\Theta} = \underset{\Theta \in \Omega_\Theta}{\arg\min} SqErr(\Theta)$

Here we accept that the set Ω_Θ of possible combinations of the parameters is compact and is such that there is at least one solution for each parameter. The proof of this property, because of the time it would take and its distance from the initial subject, is left to the reader if he wishes to convince himself of it.

The code below solves this minimisation problem:

```
# This is the calibration function
# heston_price_rec(S0, K, r, tau, params)
# Parameters are kappa, theta, sigma rho, v0, lambda

# Define variables to be used in optimization
S0 = resp['lastTradePrice']
K = volSurfaceLong['strike'].to_numpy('float')
r = volSurfaceLong['rate'].to_numpy('float')
tau = volSurfaceLong['maturity'].to_numpy('float')
P = volSurfaceLong['price'].to_numpy('float')
```

```
params = {"kappa": {"x0": 1.5, "lbub": [1e-3,5]},
          "theta": {"x0": 0.5, "lbub": [1e-3,0.1]},
          "sigma": {"x0": 0.3, "lbub": [1e-2,1]},
          "rho": {"x0": -0.6, "lbub": [-1,0]},
          "v0": {"x0": 0.3, "lbub": [1e-3,0.1]},
          "lamb": {"x0": 0.5, "lbub": [-1,1]}}

x0 = [param["x0"] for key, param in params.items()]
bnds = [param["lbub"] for key, param in params.items()]

def SqErr(params):
    kappa, theta, sigma, rho, v0, lamb = params

    err = np.sum( (P-heston_price_rec(S0, K, r, tau, params))**2 /len(P) )

    # Zero penalty term - no good guesses for parameters
    pen = 0

    return err + pen

result = minimize(SqErr, x0, tol = 1e-3, method='SLSQP',
            options={'maxiter': 1e4 }, bounds=bnds)

kappa, theta, sigma, rho, v0, lamb = [param for param in result.params]
```

Finally, we wanted to make sense of our calibration and we wanted to determine whether or not our model was correct. We therefore wanted to see the market prices compared to the prices under the Heston model and for this purpose, we used the following code:

```
heston_prices = heston_price_rec(S0, K, r, tau, params)
volSurfaceLong['heston_price'] = heston_prices

import plotly.graph_objects as go
from plotly.graph_objs import Surface
from plotly.offline import iplot, init_notebook_mode
init_notebook_mode()

fig = go.Figure(data=[go.Mesh3d(x=volSurfaceLong.maturity, y=volSurfaceLong.strike,
            z=volSurfaceLong.price, color='mediumblue', opacity=0.55)])
fig.add_scatter3d(x=volSurfaceLong.maturity, y=volSurfaceLong.strike,
            z=volSurfaceLong.heston_price, mode='markers')

fig.update_layout(
    title_text='Market Prices (Mesh) vs Calibrated Heston Prices (Markers)',
    scene = dict(xaxis_title='TIME (Years)',
```

```
                    yaxis_title='STRIKES (Pts)',
                    zaxis_title='INDEX OPTION PRICE (Pts)'),
    height,width=800,800)

fig.show()
```

Figure 12 below shows the market prices (visible with the blue mesh) and the prices simulated by our Heston model (visible with the red markers).

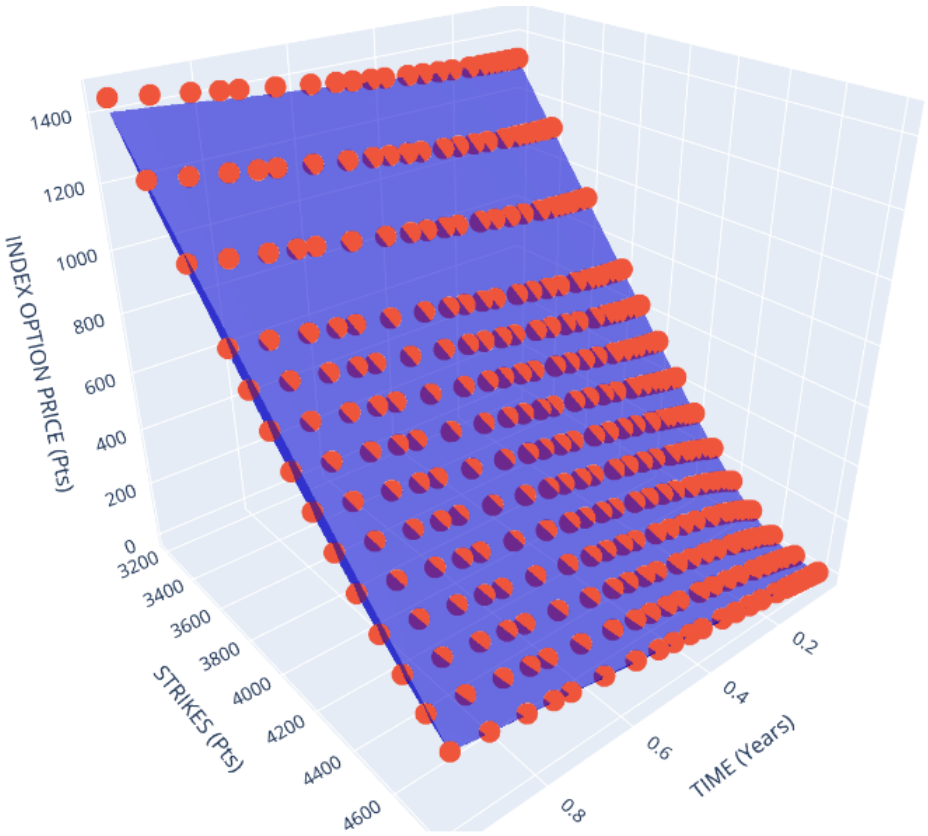

Figure 12: Market Prices (Mesh) and Calibrated Heston Prices (Markers)

III.4.D Influence of iterations

Figures 13 to 17 show the evolution of the different values of the parameters to be calibrated during the optimisation. We see that after 8000 iterations, the parameters ρ, κ, σ and v_0 reach their optimal values. On the other hand, we see that even after 20000 iterations, the parameter θ is not fixed. Nevertheless, the influence of its evolution on the value of the function to be minimized is minimal after 6000 iterations.

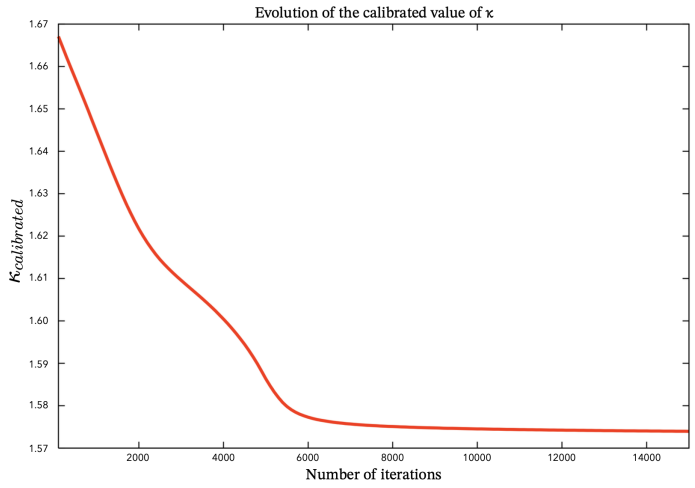

Figure 13: Evolution of the calibrated value of κ

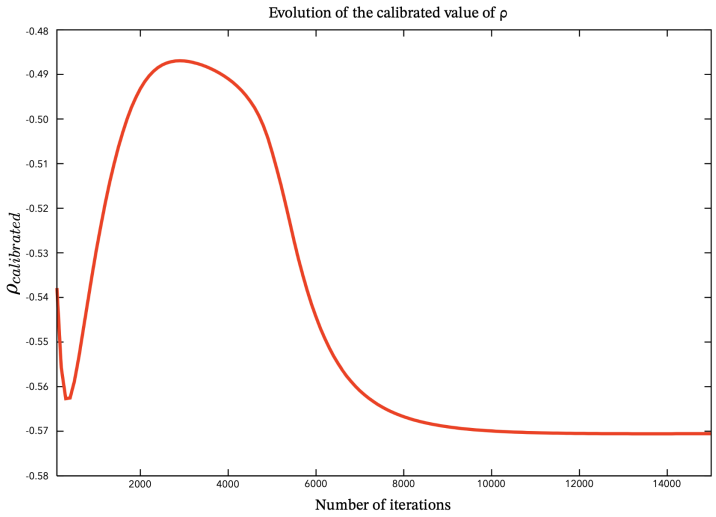

Figure 14: Evolution of the calibrated value of ρ

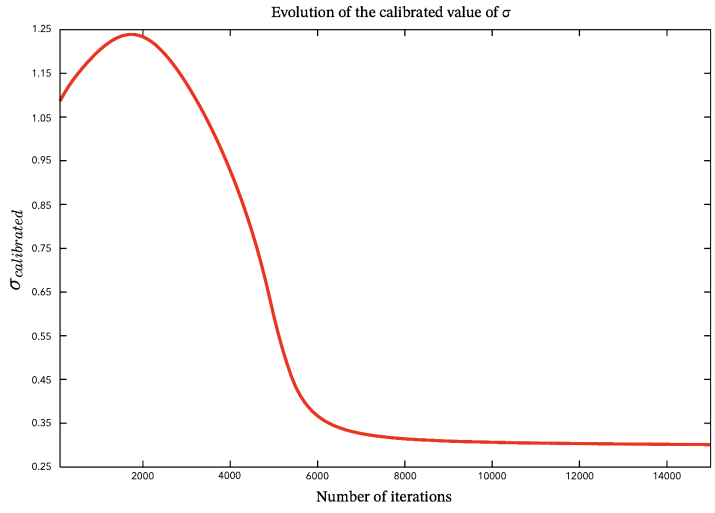

Figure 15: Evolution of the calibrated value of σ

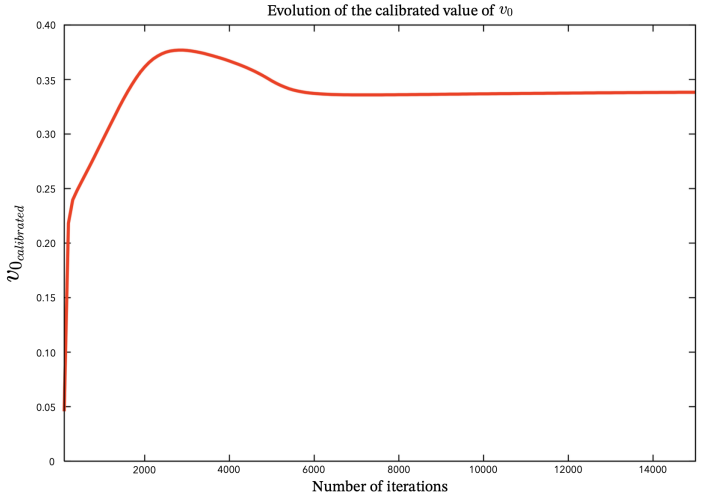

Figure 16: Evolution of the calibrated value of v_0

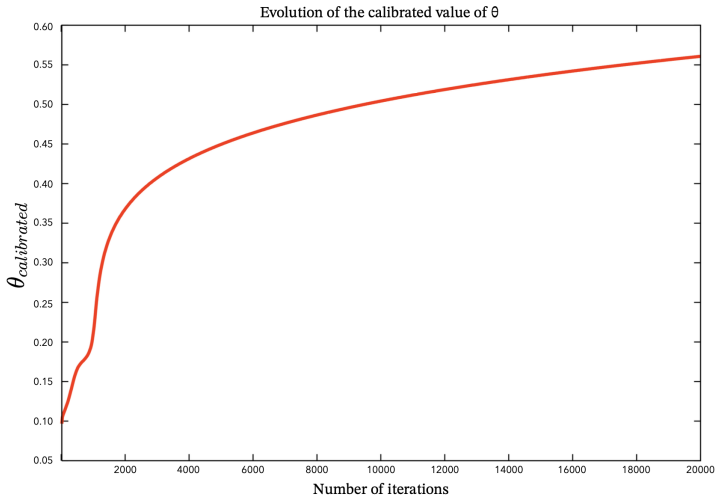

Figure 17: Evolution of the calibrated value of θ

IV Pricing of barrier options with the binomial tree method

The binomial model is a discrete method for pricing financial options. In 1979, Cox, Ross and Rubinstein (CRR) introduced the binomial method for pricing options using the different paths that the underlying asset can take. This model, also called CRR model, is an adaptation of the 1973 Black-Scholes model that has become the standard for pricing European-style options.

IV.1 Principle of the model

The binomial model is based on several key assumptions:

- A so-called "perfect" and friction-less market environment: securities can be liquid, traded freely, can be split infinitely, short sales are not limited,...
- A world where investors are risk-neutral, i.e. the expected return on risky assets is identical to the risk-free rate of return.

The binomial model also assumes that the underlying asset (for example a stock) can only take one of two possible values in each period (A or B). To represent these periods, the model uses a binary tree that illustrates the different possible price developments.

The underlying is valued through a series of "steps" that correspond to the time interval between the valuation date and the expiration date of the option. The nodes of the tree represent the possible prices of the underlying at specific points in time.

To determine the value of the option, one must go back from the option expiration date to the valuation date using the previous values of the underlying asset. The option value is equal to the value of the first node of the binomial model.

By applying this method iteratively, the possible final values of the underlying at the expiration of the option can be increased. The binomial model is particularly useful for pricing American options.

IV.2 Advantages and disadvantages

The binomial model is a simple but very effective concept for modeling complex and random phenomena in finance. The simplicity of the binomial model meets all the criteria of a robust concept capable of modeling all phenomena. Although the binomial model shows some weaknesses regarding the speed of convergence, its flexibility and extensibility make it an essential concept in finance. Different models have been derived from the binomial model (or CRR model) such as the flexible binomial model or the generalized CRR model. These models have all been studied in terms of convergence and have been improved. The binomial model also performs well in modeling complex options such as Asian options or barrier options, due to its simple properties that offer both quality and accuracy.

IV.3 Binomial tree method for European and American options

European and American options are types of financial options that differ in their exercise terms. A European option can only be exercised at the expiration of the option, while an American option can be exercised at any time before expiration. The binomial tree model is an option pricing method that uses a tree structure to model the price movement of the underlying asset and the calculation of the option value at each stage of the tree.

In the binomial tree model for European options, it is assumed that the option can only be exercised at the option's expiration. Thus, the option value at each node of the tree is calculated by taking the weighted average of the two child nodes, where each child node represents the price of the underlying asset at the next stage of the tree. This weighted average is then discounted at the risk-free interest rate to obtain the option value at the current node.

In the binomial tree model for American options, on the other hand, each node in the tree must be valued to determine whether early exercise of the option would be beneficial. Thus, the option value at each node of the tree is calculated by taking the maximum of the difference between the option strike price and the price of the underlying asset, or the weighted average of the two child nodes discounted at the risk-free interest rate. This maximum value is then discounted at the risk-free interest rate to obtain the value of the option at the current node.

The main difference between the binomial tree model for European options and for American options is the way in which the option values are calculated at each node of the tree, depending on the option's exercise conditions.

IV.4 How to apply the binomial tree method to barrier options

The binomial tree method for barrier options works by introducing special nodes into the tree to represent times when the barrier is breached. The tree nodes that represent the crossing of the barrier are called barrier nodes. It is actually quite rare for the barrier to correspond exactly to nodes in the tree, so the "barrier nodes" in the tree often correspond to frames of the actual barrier.

There are some major differences in how vanilla options and barrier options are priced using the binomial tree method.

The first difference between pricing a vanilla option and pricing a barrier option with the binomial tree method is in the calculation of the payoff. For a vanilla option, the payoff is based on the difference between the strike price and the underlying asset price at expiration. For a barrier option, the payoff is based on whether the underlying asset price has breached a pre-determined barrier level during the option's lifetime.

The second difference lies in the construction of the binomial tree. For a vanilla option, the tree is built by assuming that the underlying asset can only move up or down with fixed probabilities, and that these probabilities remain constant over time. For a

barrier option, the tree is constructed in a similar way, but the probabilities of moving up or down may depend on whether the underlying asset price has already crossed the barrier.

The third difference between the two types of options is in the pricing algorithm. For a vanilla option, the pricing algorithm involves working backwards through the binomial tree to calculate the option price at each node, using the probabilities of moving up or down and the discounted future values of the option. For a barrier option, the pricing algorithm is more complex, as the probabilities of moving up or down may depend on whether the barrier has been crossed, and there may be additional rules governing when the option is activated or terminated.

And finally, the calculation of the final payoff is different for each type of option. For a vanilla option, the final payoff is simply the difference between the strike price and the underlying asset price at expiration. For a barrier option, the final payoff may be more complicated, as it depends on whether the barrier was breached or not, and whether the option was activated or terminated.

IV.5 Construction of the binomial tree

While using python, we will consider our binomial tree as a network with nodes (i, j): i represents the time steps and j represents the number of ordered price outcome, from the bottom to the the top of the tree.

The underlying asset's value $S_{i,j}$ is represented by each node (i, j) and is calculated in the following way :

$$S_{i,j} = S_0 u^j d^{i-j}$$

$C_{i,j}$ represents the contract price at each node (i, j), meaning the current price of the option at the time and $C_{N,j}$ is the final payoff function.

For an up-and-out barrier call option, if $T = t_N$ then at the final nodes,

$$C_{N,j} = (S_{N,j} - K)_+ \mathbb{1}_{S_{N,j}<H}$$

And at each node (i, j) :

- if $S_{N,j} > H$ then $C_{i,j} = 0$
- if $S_{N,j} < H$ then $C_{i,j} = e^{-r\Delta T} q_{ij} C_{i+j,j+1} + (1 - q_{ij}) C_{i+1,j-1}$

We will use for loops to iterate through the nodes j at each time step i, and therefore building our binomial tree.

When trying to price a barrier option with a binomial tree method, we quickly had to decide how to define the parameters of the model, and especially the up and down factors, u and d. We looked at several methods such as the Jarrow Rudd (JR) method or even the Equal Probabilities method, but in the end, we choose to define u and d (and therefore p) with the Cox, Ross and Rubinstein (CRR) method.

For the CRR method, we choose the up and down factors to have the same size but opposite. The factors are defined in the following way :

$$u = e^{\sigma\sqrt{\Delta t}}$$

$$d = e^{-\sigma\sqrt{\Delta t}}$$

The CCR method ensures that the tree is recombining, i.e., if the price of the underlying asset decreases and then increases, the price of the option will be the same as if the price of the underlying asset increases and then decreases. In either case, the branches merge and recombine. This property speeds up the calculation of the option price.

Now let's take a look at how both European and American options behave when parameters S_0 and K vary.

We consider an up and out barrier option with only S_0 and K varying. The other parameters (T, H, r, σ) remain constant.

Given that we are looking at a up and out barrier option with $H = 120$, we do notice that in both cases, European and American, the option never gets deactivated in this particular example.

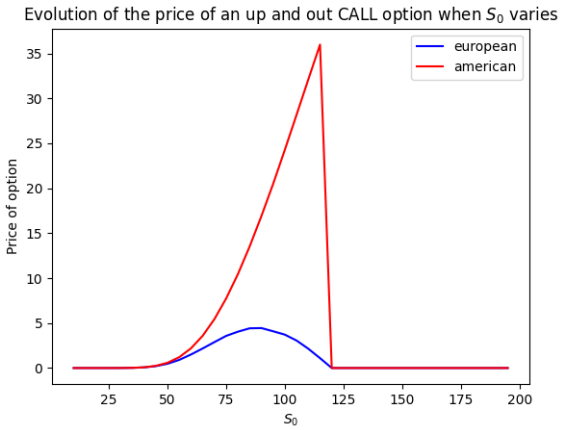

Figure 18: Evolution of the price of an up and out CALL option when S_0 varies

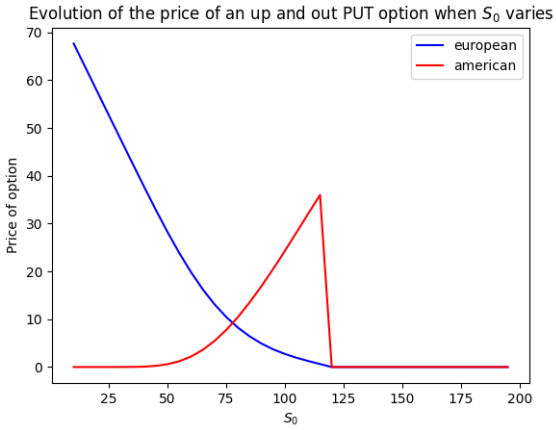

Figure 19: Evolution of the price of an up and out PUT option when S_0 varies

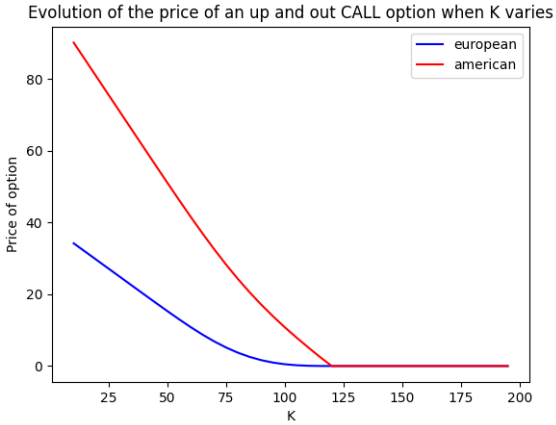

Figure 20: Evolution of the price of an up and out CALL option when K varies

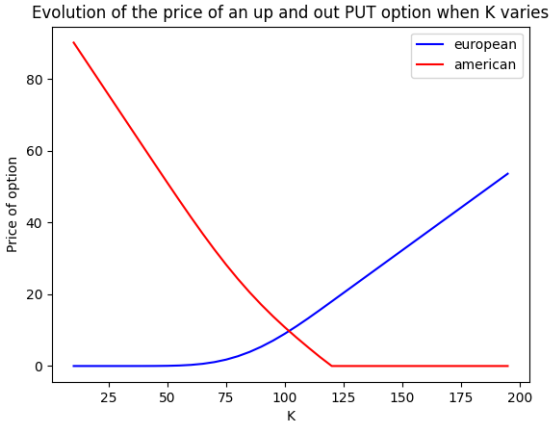

Figure 21: Evolution of the price of an up and out PUT option when K varies

But if we take a look at a down and in barrier option with $H = 20$, we begin to see a real difference between the price of a European and an American barrier option.

We can actually notice that with the parameters that we chose, the European option barely gets activated while the American option crossed the barrier every time.

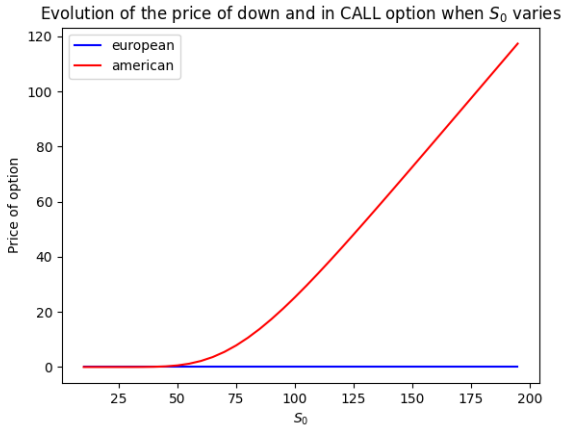

Figure 22: Evolution of the price of a down and in CALL option when S_0 varies

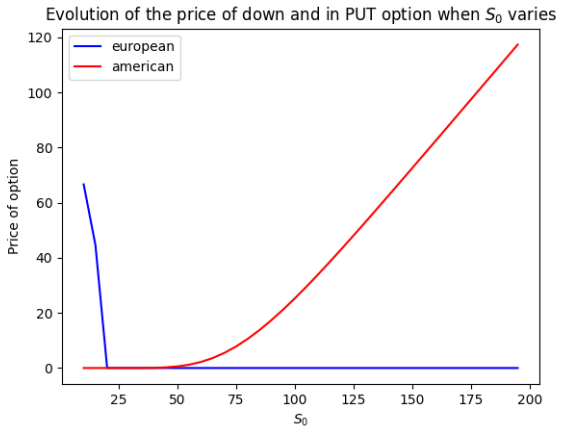

Figure 23: Evolution of the price of a down and in PUT option when S_0 varies

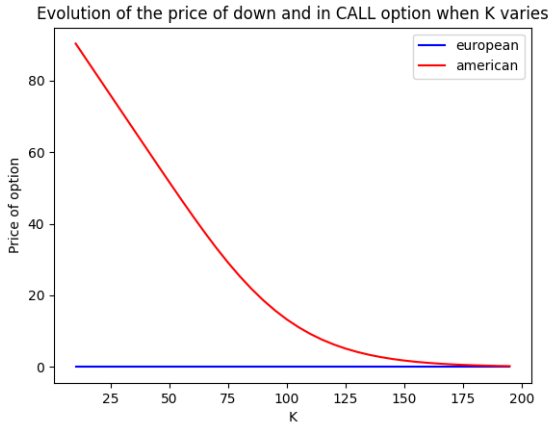

Figure 24: Evolution of the price of a down and in CALL option when K varies

Figure 25: Evolution of the price of a down and in PUT option when K varies

Conclusion

In conclusion, the aim of this paper has been to present the main pricing methods and models of barrier options, focusing on the stochastic models most commonly used in finance. We have examined in detail the Heston model as well as the Heston model with jumps, otherwise known as the Bates model, using the Monte Carlo method and the binomial tree method for pricing barrier options. We implemented these different methods using the Python programming language (code available in the Appendix page 43).

The results of our study showed that the Monte-Carlo method was the most accurate for pricing barrier options, although the limitations of this method should be taken into account and the variance reduction and error minimisation techniques should be considered. On the other hand, the binomial tree method was faster and easier to implement. This work also allowed us to see that model calibration, although a particularly complex task, was an indispensable process for obtaining accurate barrier option prices, as was the choice of parameters, which is also a key point in option pricing.

Finally, this work has highlighted the importance of stochastic modeling in finance and has shown how the methods presented can be applied to the pricing of barrier options. This TER provides a better understanding of stochastic models and barrier option pricing methods and gives insight into different implementation methods. Finally, this TER has, above all, allowed us to apply all the concepts of stochastic modeling and option theory that our professor Ms Eyraud Anne has taught us during our semester.

References and Bibliography

[1] Xu Jianqiang. "Pricing and hedging options under stochastic volatility". PhD thesis. University of British Columbia, 2003.

[2] Kendall Maurice George Stuart Alan. *Kendall's advanced theory of statistics*. A Hodder Arnold Publication, 1998.

[3] El-Karoui Nicole. *Couverture des risques dans les marchés financiers*. École Polytechnique, 2003.

[4] Bergeron Maxime Fung Nicholas Hull John Poulos Zissis. *Variational Autoencoders : A Hands-Off Approach to Volatility*. 2021.

[5] Eyraud Anne. *Théorie des options*. Support de cours, 2023.

[6] Ma Haoxiang. "Calibration of Heston Pricing Model - using for the valuation of employee stock option". MA thesis. Institut de Science Financière et d'Assurances, 2012.

[7] Heston Steven L. "A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options". In: *The Review of Financial Studies* 6 (1993).

[8] Ksouri Najed. "Approximation Numerical Methods for vanilla and asian options for Heston stochastic volatility model pricing". MA thesis. INRIA, 2012.

[9] Bencheikh Oumaima Jourdain Benjamin. *Convergence in total variation of the Euler-Maruyama scheme applied to diffusion processes with measurable drift coefficient and additive noise*. 2020.

[10] Rivalland Johann Tirolien Thibaut. *Options Barrières : Couverture statique et risque de modèle*. ENSAE, 2003.

[11] Dudley Evan. "Évaluation d'options à barrière discrète avec volatilité stochastique". MA thesis. HEC, 1999.

[12] Augros Jean-Claude Moreno Michaël. *Évaluation partiellement séquentielle des options à barrière*. ISFA, 1999.

[13] Zayani Sami. "Lien entre le modèle binomial et les équations de Black-Shcoles". MA thesis. Université du Québec à Montréal, 2016.

[14] "Option à barrière activante, désactivante... Définition Explications". In: *Finance de Marché* (2013).

[15] QuantPy. *Barrier Option Pricing with Binomial Trees*. Youtube. 2022. URL: `https://www.youtube.com/watch?v=WxrRi9lNnqY`.

[16] QuantPy. *How to Choose Binomial Parameters - Binomial Option Pricing*. Youtube. 2022. URL: `https://www.youtube.com/watch?v=nWslah9tHLk`.

Appendices

Group Organization

It is obvious that each of us is more comfortable with this or that part of the subject (theory, pure mathematics, programming in Python, interpretations, explanations, writing, LaTeX indentation...). However, we have tried to participate as much as possible in the research and development of the TER. Our work was organized throughout the semester, with regular work points and task sharing, at school or via Microsoft Teams. Moreover, we often asked our teacher Anne Eyraud (between classes or by mail) when we encountered a problem or when we needed advice or guidance.

Several notions seen in the theory of options class were indispensable throughout our work. Our work was regularly updated directly on Overleaf which allowed the three of us to have a direct visibility on our work and to easily see each other's progress.

It is always interesting to do a research work in group in order to work on our organization and our communication, but also our autonomy. Our choice of subject has certainly allowed us to be more involved and interested. Finally, this work also allowed us to learn a lot about option pricing which will undoubtedly be useful in our future studies or professional life.

Python Code

```
## TER Théo JALABERT - Sabrina KHORSI - Lou SIMONEAU-FRIGGI

import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns

def heston_jump_monte_carlo_barrier(S0, K, r, T, params, barrier, option_type,
    mc_paths, time_steps, barrier_type, antithetic=True, seed=None):

    # Initialization of the parameters of the Heston model with jumps
    kappa, theta, sigma, rho, v0, lamb, mu_jump, delta_jump = params

    # Initialization of the variables and the random number generator
    np.random.seed(seed)
    dt = T / time_steps
    sqrt_dt = np.sqrt(dt)

    # Simulation of price paths
    option_values = []
    all_St = []
    all_St_antithetic = []
    barrier_crossed = []
```

```
barrier_crossed_antithetic = []
option_off = []
option_antithetic_off = []

for _ in range(mc_paths // (2 if antithetic else 1)):
    # Generating correlated Brownian motion and a Poisson process
    W1 = np.random.normal(size=time_steps) * sqrt_dt
    W2 = np.random.normal(size=time_steps) * sqrt_dt
    Z = rho * W1 + np.sqrt(1 - rho ** 2) * W2
    N = np.random.poisson(lamb * dt, size=time_steps)

    # Discretising stochastic differential equations with Euler-Maruyama method
    St = np.zeros(time_steps + 1)
    vt = np.zeros(time_steps + 1)
    St[0] = S0
    vt[0] = v0
    for t in range(time_steps):
        St[t + 1] = St[t] * np.exp((r - 0.5 * vt[t]) * dt +
                    np.sqrt(vt[t]) * W1[t] + mu_jump * N[t] -
                    0.5 * delta_jump ** 2 * N[t])
        vt[t + 1] = np.maximum(0, vt[t] + kappa * (theta - vt[t]) * dt +
                    sigma * np.sqrt(vt[t]) * Z[t])

    option_value = 0
    option_value_antithetic = 0

    # Check if the barrier is crossed and calculate the value of the option
    if option_type == 'call':
        if barrier_type == 'up-and-out':
            if np.any(St >= barrier):
                option_value = 0
                barrier_crossed.append(True)
            else:
                option_value = np.exp(-r * T) * max(St[-1] - K, 0)
                barrier_crossed.append(False)
        elif barrier_type == 'up-and-in':
            if np.any(St >= barrier):
                option_value = np.exp(-r * T) * max(St[-1] - K, 0)
                barrier_crossed.append(False)
            else:
                option_value = 0
                barrier_crossed.append(True)
    elif option_type == 'put':
        if barrier_type == 'down-and-out':
            if np.any(St <= barrier):
                option_value = 0
                barrier_crossed.append(True)
```

```
            else:
                option_value = np.exp(-r * T) * max(K - St[-1], 0)
                barrier_crossed.append(False)
        elif barrier_type == 'down-and-in':
            if np.any(St <= barrier):
                option_value = np.exp(-r * T) * max(K - St[-1], 0)
                barrier_crossed.append(False)
            else:
                option_value = 0
                barrier_crossed.append(True)

    option_values.append(option_value)

    # Add paths to the list
    all_St.append(St)
    option_off.append(option_value == 0)

    # Using the antithetical variable to reduce variance
    if antithetic:
        St_antithetic = np.zeros(time_steps + 1)
        vt_antithetic = np.zeros(time_steps + 1)
        St_antithetic[0] = S0
        vt_antithetic[0] = v0

        for t in range(time_steps):
            St_antithetic[t + 1] = St_antithetic[t] *
                np.exp((r - 0.5 * vt_antithetic[t]) * dt -
                np.sqrt(vt_antithetic[t]) * W1[t] -
                mu_jump * N[t] + 0.5 * delta_jump ** 2 * N[t])
            vt_antithetic[t + 1] = np.maximum(0, vt_antithetic[t] +
                kappa * (theta - vt_antithetic[t]) * dt -
                sigma * np.sqrt(vt_antithetic[t]) * Z[t])

        # Check if the barrier is crossed and calculate the value of the barrier
        # option for the antithetical path
        if option_type == 'call':
            if barrier_type == 'up-and-out':
                if np.any(St_antithetic >= barrier):
                    option_value_antithetic = 0
                    barrier_crossed_antithetic.append(True)
                else:
                    option_value_antithetic = np.exp(-r * T) *
                        max(St_antithetic[-1] - K, 0)
                    barrier_crossed_antithetic.append(False)
            elif barrier_type == 'up-and-in':
                if np.any(St_antithetic >= barrier):
                    option_value_antithetic = np.exp(-r * T) *
```

```
                            max(St_antithetic[-1] - K, 0)
                        barrier_crossed_antithetic.append(False)
                    else:
                        option_value_antithetic = 0
                        barrier_crossed_antithetic.append(True)
            elif option_type == 'put':
                if barrier_type == 'down-and-out':
                    if np.any(St_antithetic <= barrier):
                        option_value_antithetic = 0
                        barrier_crossed_antithetic.append(True)
                    else:
                        option_value_antithetic = np.exp(-r * T) *
                            max(K - St_antithetic[-1], 0)
                        barrier_crossed_antithetic.append(False)
                elif barrier_type == 'down-and-in':
                    if np.any(St_antithetic <= barrier):
                        option_value_antithetic = np.exp(-r * T) *
                            max(K - St_antithetic[-1], 0)
                        barrier_crossed_antithetic.append(False)
                    else:
                        option_value_antithetic = 0
                        barrier_crossed_antithetic.append(True)

            option_values.append(option_value_antithetic)

            # Add antithetical paths to the list
            all_St_antithetic.append(St_antithetic)
            option_antithetic_off.append(option_value_antithetic == 0)

    # Estimating barrier option prices and price paths
    option_price = np.mean(option_values)

    return option_price, all_St, all_St_antithetic, barrier_crossed,
        barrier_crossed_antithetic, option_off, option_antithetic_off


def plot_stock_prices(all_St, barrier_crossed, option_off=None,
        all_St_antithetic=None, barrier_crossed_antithetic=None,
        option_antithetic_off=None,K=None):
    sns.set(style="darkgrid")

    # Graph of options still valid or void at expiration
    plt.figure(1)

    first_valid = True
    first_disabled = True
    for i, St in enumerate(all_St):
```

```
        if not barrier_crossed[i] and not option_off[i]:
            label = "Valid option" if first_valid else None
            plt.plot(St, color="blue", alpha=0.7, label=label)
            first_valid = False
        elif not barrier_crossed[i]:
            label = "Unexercised option" if first_disabled else None
            plt.plot(St, color="gray", alpha=0.7, label=label)
            first_disabled = False


    # The legends of the antithetical variable types are fixed
    first_valid_antithetic = True
    first_disabled_antithetic = True

    if all_St_antithetic is not None and barrier_crossed_antithetic is not None:
        for i, St_antithetic in enumerate(all_St_antithetic):

            if not barrier_crossed_antithetic[i] and not option_antithetic_off[i]:
                label = "Valid antithetic variable" if first_valid_antithetic
                        else None
                plt.plot(St_antithetic, color="deepskyblue", alpha=0.5,
                        linestyle="--", label=label)
                first_valid_antithetic = False

            elif not barrier_crossed_antithetic[i]:
                label = "Unexercised antithetic variable"
                    if first_disabled_antithetic else None
                plt.plot(St_antithetic, color="gray", alpha=0.5, linestyle="--",
                    label=label)
                first_disabled_antithetic = False

    plt.axhline(y=barrier,color="r",linestyle="-",label=f"Barrier ({barrier_type})")
    plt.axhline(y=K, color="green", linestyle="-", label="Strike")
    plt.xlabel("Time steps")
    plt.ylabel("Underlying asset price")
    plt.title("Simulation of the underlying asset price with the Heston model
        with jumps (valid options)")
    plt.legend()
    plt.show()


    # Graph of deactivated options
    plt.figure(2)

    first_off_barrier = True
    first_off = True
    for i, St in enumerate(all_St):
```

```
        if barrier_crossed[i]:
            label = "Deactivated option" if first_off_barrier else None
            plt.plot(St, color="red", alpha=0.7, label=label)
            first_off_barrier = False

    first_antithetic_off_barrier = True
    first_antithetic_off = True

    if all_St_antithetic is not None and barrier_crossed_antithetic is not None:
        for i, St_antithetic in enumerate(all_St_antithetic):
            if barrier_crossed_antithetic[i]:
                label = "Deactivated antithetic option"
                    if first_antithetic_off_barrier else None
                plt.plot(St_antithetic, color="red", alpha=0.5, linestyle="--",
                    label=label)
                first_antithetic_off_barrier = False


    plt.axhline(y=barrier,color="r",linestyle="-",label=f"Barrier ({barrier_type})")
    plt.axhline(y=K, color="green", linestyle="-", label="Strike")
    plt.xlabel("Time steps")
    plt.ylabel("Underlying asset price")
    plt.title("Simulation of the underlying asset price with the Heston model
        with jumps (deactivated options)")
    plt.legend()
    plt.show()


    plt.figure(3)

    # Options with uncrossed barrier
    first_valid = True
    first_disabled = True
    for i, St in enumerate(all_St):
        if not barrier_crossed[i] and not option_off[i]:
            label = "Valid option" if first_valid else None
            plt.plot(St, color="blue", alpha=0.7, label=label)
            first_valid = False
        elif not barrier_crossed[i]:
            label = "Unexercised option" if first_disabled else None
            plt.plot(St, color="gray", alpha=0.7, label=label)
            first_disabled = False

    # The legends of the antithetical variable types are fixed
    first_valid_antithetic = True
    first_disabled_antithetic = True
```

```
if all_St_antithetic is not None and barrier_crossed_antithetic is not None:
    for i, St_antithetic in enumerate(all_St_antithetic):

        if not barrier_crossed_antithetic[i] and not option_antithetic_off[i]:
            label = "Valid antithetic variable" if first_valid_antithetic
                    else None
            plt.plot(St_antithetic, color="deepskyblue", alpha=0.5,
                linestyle="--", label=label)
            first_valid_antithetic = False

        elif not barrier_crossed_antithetic[i]:
            label = "Unexercised antithetic variable"
                    if first_disabled_antithetic else None
            plt.plot(St_antithetic, color="gray", alpha=0.5, linestyle="--",
                    label=label)
            first_disabled_antithetic = False

# Deactivated options
first_off_barrier = True
first_off = True
for i, St in enumerate(all_St):
    if barrier_crossed[i]:
        label = "Deactivated option" if first_off_barrier else None
        plt.plot(St, color="red", alpha=0.7, label=label)
        first_off_barrier = False

first_antithetic_off_barrier = True
first_antithetic_off = True

if all_St_antithetic is not None and barrier_crossed_antithetic is not None:
    for i, St_antithetic in enumerate(all_St_antithetic):
        if barrier_crossed_antithetic[i]:
            label = "Deactivated antithetic option"
                    if first_antithetic_off_barrier else None
            plt.plot(St_antithetic, color="red", alpha=0.5, linestyle="--",
                    label=label)
            first_antithetic_off_barrier = False

plt.axhline(y=barrier,color="r",linestyle="-",label=f"Barrier ({barrier_type})")
plt.axhline(y=K, color="green", linestyle="-", label="Strike")
plt.xlabel("Time steps")
plt.ylabel("Underlying asset price")
plt.title("Simulation of the underlying asset price with the Heston model
        with jumps (all options)")
plt.legend()
plt.show()
```

```
# Parameters to test model
S0 = 100.  # Initial asset price
K = 80.    # Strike
r = 0.03  # Risk free rate
T = 1.     # Time to maturity (in years)

# Parameters of the Heston model with jumps
kappa = 1.5768 # Rate of mean reversion of variance process
theta = 0.0398 # Long-term mean variance
sigma = 0.3 # Volatility of volatility
rho = -0.5711 # Correlation between variance and stock process
v0 = 0.1 # Initial variance
lamb = 0.575 # Risk premium of variance
mu_jump = -0.06
delta_jump = 0.1
params = (kappa, theta, sigma, rho, v0, lamb, mu_jump, delta_jump)



# Barrier options parameters
barrier = 70                    # Barrier level
option_type = 'put'             # Option type (call or put)
barrier_type = 'down-and-out'   # Barrier type (up-and-out, up-and-in,
                                # down-and-out, down-and-in)

# barrier = 95
# option_type = 'put'
# barrier_type = 'down-and-in'

# barrier = 120
# option_type = 'call'
# barrier_type = 'up-and-out'

# barrier = 110
# option_type = 'call'
# barrier_type = 'up-and-in'

# Monte-Carlo simulation parameters
mc_paths = 30               # Number of simulations
time_steps = 252            # Number of time steps in the simulation

# Estimation of the barrier option price and extraction of the price trajectories
option_price,all_St, all_St_antithetic, barrier_crossed, barrier_crossed_antithetic,
    option_off, option_antithetic_off = heston_jump_monte_carlo_barrier(S0, K, r, T,
    params, barrier, option_type, mc_paths, time_steps, barrier_type=barrier_type)
print(f"Estimated barrier option price : {option_price:.2f}")
```

```
# Plot charts for all price paths of the underlying asset
plot_stock_prices(all_St, barrier_crossed, option_off, all_St_antithetic,
barrier_crossed_antithetic, option_antithetic_off,K)



## Analysis of pricing as a function of the number of iterations (curve)

from scipy.interpolate import interp1d
import numpy as np

# Analysis of pricing according to the number of iterations
iterations_list = [10, 50, 100, 500, 1000, 2500, 5000, 7500, 10000, 12000, 14000,
    18000, 22000, 26000, 30000, 34000, 38000, 42000, 46000, 50000, 54000, 60000]
# List of different iteration values to be tested
option_prices = []  # List to capture prices of options

for iters in iterations_list:
    option_price = heston_jump_monte_carlo_barrier(S0, K, r, T, params, barrier,
        option_type, iters, time_steps, barrier_type=barrier_type)[0]
    option_prices.append(option_price)

# An interpolation function is created with the iteration and price data
interpolation_function = interp1d(iterations_list, option_prices, kind='cubic')

# A new set of iteration points is created for a smoother curve
smooth_iterations = np.linspace(min(iterations_list), max(iterations_list), 500)


# The interpolated prices for the new iterations are calculated
smooth_option_prices = interpolation_function(smooth_iterations)

# The graph of the pricing time versus the number of simulations is displayed
plt.title("Influence of the number of simulations on the option price")
plt.plot(smooth_iterations, smooth_option_prices, linestyle='-',color = 'red')
#plt.scatter(smooth_iterations, smooth_option_prices, marker='o', color='r')
plt.xlabel("Number of simulations")
plt.ylabel("Option price")
plt.grid(True)
plt.show()


## Analysis of the influence of parameters on the price

def analyze_parameter_influence(S0, K, r, T, params, barrier, option_type, mc_paths,
        time_steps, barrier_type):
    param_names = ['kappa', 'theta', 'sigma', 'rho', 'v0', 'lamb', 'mu_jump',
```

```
        'delta_jump']
    parameters_to_study = [ [-0.7, -0.4, -0.1], [0.03, 0.04, 0.05], [1.0, 1.5, 2.0],
        [0.05, 0.1, 0.15], [0.2, 0.3, 0.4]]
    results = {}

    for i in range(5):

    for parameter_name in param_names:
        temp_params = params
        volatilities = []

        for delta in parameter_values:
            temp_params[parameter_name] += delta
            all_S = heston_jump_monte_carlo_barrier(S0, K, r, T, temp_params,
                barrier, option_type, mc_paths, time_steps, barrier_type)[1]
            sample_volatility = calculate_sample_volatility(all_S)
            volatilities.append(sample_volatility)

        results[parameter_name] = volatilities

    return results

def plot_parameter_influence(results):
    fig, ax = plt.subplots()

    for parameter_name, volatilities in results.items():
        parameter_values = np.linspace(-0.5, 0.5, num=11)
        ax.plot(parameter_values, volatilities, label=parameter_name)

    ax.set_xlabel('Variation of parameters')
    ax.set_ylabel('Historical volatility')
    ax.set_title('Influence of parameters on historical volatility')
    ax.legend()
    plt.show()

# Parameters of the Heston model with jumps
kappa = 1.5768 # Rate of mean reversion of variance process
theta = 0.0398 # Long-term mean variance
sigma = 0.3 # Volatility of volatility
rho = -0.5711 # Correlation between variance and stock process
v0 = 0.1 # Initial variance
lamb = 0.575 # Risk premium of variance
mu_jump = -0.06
delta_jump = 0.1
params = {kappa, theta, sigma, rho, v0, lamb, mu_jump, delta_jump}

# Additional parameters for the barrier option and simulation
```

```
S0 = 100
K = 100
r = 0.03
T = 1
barrier = 120
option_type = 'call'
mc_paths = 1000
time_steps = 252
barrier_type = 'up-and-out'

# Analysis of the influence of the parameters and display of the results
results = analyze_parameter_influence(S0, K, r, T, params, barrier, option_type,
    mc_paths, time_steps, barrier_type)
plot_parameter_influence(results)
```

```
### Calibration under the Heston model

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.integrate import quad
from scipy.optimize import minimize
from datetime import datetime as dt
from eod import EodHistoricalData
from nelson_siegel_svensson import NelsonSiegelSvenssonCurve
from nelson_siegel_svensson.calibrate import calibrate_nss_ols
import csv

# with open('/Users/theojalabert/Desktop/TER/S&P500_lastyear.csv', 'r') as data:
#     reader = csv.reader(data)
#     data_SNP500 = []
#     for ligne in reader:
#         data_SNP500.append(ligne)
#
# data.close()

# data_SNP500[0] = ['date','lastTradePrice','volume','open','top','bottom']
# for i in range(1,len(data_SNP500)):
#     data_SNP500[i][1],data_SNP500[i][3],data_SNP500[i][4],data_SNP500[i][5] =
#         float(data_SNP500[i][1]),float(data_SNP500[i][3]),
#         float(data_SNP500[i][4]),float(data_SNP500[i][5])
```

```
def heston_charfunc(phi, S0, v0, kappa, theta, sigma, rho, lambd, tau, r):

    # constants
    a = kappa*theta
    b = kappa+lambd

    # common terms w.r.t phi
    rspi = rho*sigma*phi*1j

    # define d parameter given phi and b
    d = np.sqrt( (rho*sigma*phi*1j - b)**2 + (phi*1j+phi**2)*sigma**2 )

    # define g parameter given phi, b and d
    g = (b-rspi+d)/(b-rspi-d)

    # calculate characteristic function by components
    exp1 = np.exp(r*phi*1j*tau)
    term2 = S0**(phi*1j) * ( (1-g*np.exp(d*tau))/(1-g) )**(-2*a/sigma**2)
    exp2 = np.exp(a*tau*(b-rspi+d)/sigma**2 + v0*(b-rspi+d)*( (1-np.exp(d*tau))/
            (1-g*np.exp(d*tau)) )/sigma**2)
    return exp1*term2*exp2

def integrand(phi, S0, v0, kappa, theta, sigma, rho, lambd, tau, r):
    args = (S0, v0, kappa, theta, sigma, rho, lambd, tau, r)
    numerator = np.exp(r*tau)*heston_charfunc(phi-1j,*args) -
            K*heston_charfunc(phi,*args)
    denominator = 1j*phi*K**(1j*phi)
    return numerator/denominator

def heston_price_rec(S0, K, v0, kappa, theta, sigma, rho, lambd, tau, r):
    args = (S0, v0, kappa, theta, sigma, rho, lambd, tau, r)

    P, umax, N = 0, 100, 10000
    dphi=umax/N # dphi is width
    for i in range(1,N):
        # rectangular integration
        phi = dphi * (2*i + 1)/2 # midpoint to calculate height
        numerator = np.exp(r*tau)*heston_charfunc(phi-1j,*args) -
                K * heston_charfunc(phi,*args)
        denominator = 1j*phi*K**(1j*phi)

        P += dphi * numerator/denominator

    return np.real((S0 - K*np.exp(-r*tau))/2 + P/np.pi)

def heston_price(S0, K, v0, kappa, theta, sigma, rho, lambd, tau, r):
    args = (S0, v0, kappa, theta, sigma, rho, lambd, tau, r)
```

```
    real_integral, err = np.real( quad(integrand, 0, 100, args=args) )

    return (S0 - K*np.exp(-r*tau))/2 + real_integral/np.pi

# Parameters to test model
S0 = 100. # initial asset price
K = 100. # strike
v0 = 0.1 # initial variance
r = 0.03 # risk free rate
kappa = 1.5768 # rate of mean reversion of variance process
theta = 0.0398 # long-term mean variance
sigma = 0.3 # volatility of volatility
lambd = 0.575 # risk premium of variance
rho = -0.5711 # correlation between variance and stock process
tau = 1. # time to maturity
print(heston_price( S0, K, v0, kappa, theta, sigma, rho, lambd, tau, r ))

yield_maturities = np.array([1/12, 2/12, 3/12, 6/12, 1, 2, 3, 5, 7, 10, 20, 30])
yeilds = np.array([0.15,0.27,0.50,0.93,1.52,2.13,2.32,
            2.34,2.37,2.32,2.65,2.52]).astype(float)/100


#NSS model calibrate
curve_fit, status = calibrate_nss_ols(yield_maturities,yeilds)
curve_fit

# load the key from the environment variables
api_key = "64369c2fd0ad32.39402157"

# creation of the client instance
client = EodHistoricalData(api_key)
resp = client.get_stock_options('GSPC.INDX')
resp
market_prices = {}
S0 = resp['lastTradePrice']
for i in resp['data']:
    market_prices[i['expirationDate']] = {}
    market_prices[i['expirationDate']]['strike'] = [name['strike']
            for name in i['options']['CALL']]# if name['volume'] is not None]
    market_prices[i['expirationDate']]['price'] = [(name['bid']+name['ask'])/2
            for name in i['options']['CALL']]# if name['volume'] is not None]
all_strikes = [v['strike'] for i,v in market_prices.items()]
common_strikes = set.intersection(*map(set,all_strikes))
print('Number of common strikes:', len(common_strikes))
common_strikes = sorted(common_strikes)
prices = []
```

```
maturities = []
for date, v in market_prices.items():
    maturities.append((dt.strptime(date, '%Y-%m-%d') - dt.today()).days/365.25)
    price = [v['price'][i] for i,x in enumerate(v['strike']) if x in common_strikes]
    prices.append(price)
price_arr = np.array(prices, dtype=object)
np.shape(price_arr)
volSurface = pd.DataFrame(price_arr, index = maturities, columns = common_strikes)
volSurface = volSurface.iloc[(volSurface.index > 0.04) & (volSurface.index < 1),
        (volSurface.columns > 3000) & (volSurface.columns < 5000)]
volSurface


# Convert our vol surface to dataframe for each option price with parameters
volSurfaceLong = volSurface.melt(ignore_index=False).reset_index()
volSurfaceLong.columns = ['maturity', 'strike', 'price']

# Calculate the risk free rate for each maturity using the fitted yield curve
volSurfaceLong['rate'] = volSurfaceLong['maturity'].apply(curve_fit)

# This is the calibration function
# heston_price(S0, K, v0, kappa, theta, sigma, rho, lambd, tau, r)
# Parameters are v0, kappa, theta, sigma, rho, lambd
# Define variables to be used in optimization
S0 = resp['lastTradePrice']
r = volSurfaceLong['rate'].to_numpy('float')
K = volSurfaceLong['strike'].to_numpy('float')
tau = volSurfaceLong['maturity'].to_numpy('float')
P = volSurfaceLong['price'].to_numpy('float')
params = {"v0": {"x0": 0.1, "lbub": [1e-3,0.1]},
          "kappa": {"x0": 3, "lbub": [1e-3,5]},
          "theta": {"x0": 0.05, "lbub": [1e-3,0.1]},
          "sigma": {"x0": 0.3, "lbub": [1e-2,1]},
          "rho": {"x0": -0.8, "lbub": [-1,0]},
          "lambd": {"x0": 0.03, "lbub": [-1,1]},
          }
x0 = [param["x0"] for key, param in params.items()]
bnds = [param["lbub"] for key, param in params.items()]
def SqErr(x):
    v0, kappa, theta, sigma, rho, lambd = [param for param in x]

    # Attempted to use scipy integrate quad module as constrained to single
    #       floats not arrays
    # err = np.sum([ (P_i-heston_price(S0, K_i, v0, kappa, theta, sigma, rho, lambd,
    #       tau_i, r_i))**2 /len(P) \
    #               for P_i, K_i, tau_i, r_i in zip(marketPrices, K, tau, r)])
```

```
    # Decided to use rectangular integration function in the end
    err = np.sum( (P-heston_price_rec(S0, K, v0, kappa, theta, sigma, rho, lambd,
            tau, r))**2 /len(P) )

    # Zero penalty term - no good guesses for parameters
    pen = 0 #np.sum( [(x_i-x0_i)**2 for x_i, x0_i in zip(x, x0)] )

    return err + pen
result = minimize(SqErr, x0, tol = 1e-3, method='SLSQP', options={'maxiter': 1e4 },
        bounds=bnds)
v0, kappa, theta, sigma, rho, lambd = [param for param in result.x]
v0, kappa, theta, sigma, rho, lambd

heston_prices = heston_price_rec(S0, K, v0, kappa, theta, sigma, rho, lambd, tau, r)
volSurfaceLong['heston_price'] = heston_prices

import plotly.graph_objects as go
from plotly.graph_objs import Surface
from plotly.offline import iplot, init_notebook_mode
init_notebook_mode()
fig = go.Figure(data=[go.Mesh3d(x=volSurfaceLong.maturity, y=volSurfaceLong.strike,
        z=volSurfaceLong.price, color='mediumblue', opacity=0.55)])
fig.add_scatter3d(x=volSurfaceLong.maturity, y=volSurfaceLong.strike,
        z=volSurfaceLong.heston_price, mode='markers')
fig.update_layout(
    title_text='Market Prices (Mesh) vs Calibrated Heston Prices (Markers)',
    scene = dict(xaxis_title='TIME (Years)',
                 yaxis_title='STRIKES (Pts)',
                 zaxis_title='INDEX OPTION PRICE (Pts)'),
    height=800,
    width=800
)
fig.show()


## Pricing with binomial tree
import numpy as np

# Parameters initialization
S0 = 100      # initial stock price
K = 80        # strike price
T = 1       # time to maturity in years
H = 120       # up-and-out barrier price/value
r = 0.06      # annual risk-free rate
N = 1000       # number of time steps
sigma=0.3     # volatility
opttype = 'C' # Option Type 'C' or 'P'
```

```
def binomial_tree_european_up_and_out(K,T,S0,H,r,N,sigma,opttype):
    #precompute values
    dt = T/N
    u = np.exp(sigma*np.sqrt(dt))
    d = 1/u
    q = (np.exp(r*dt) - d)/(u-d)
    disc = np.exp(-r*dt)

    # initialise asset prices at maturity
    S = np.zeros(N+1)
    for j in range(0,N+1):
        S[j] = S0 * u**j * d**(N-j)

    # option payoff
    C = np.zeros(N+1)
    for j in range(0,N+1):
        if opttype == 'C':
            C[j] = max(0, S[j] - K)
        else:
            C[j] = max(0, K - S[j])

    # check terminal condition payoff
    for j in range(0, N+1):
        S = S0 * u**j * d**(N-j)
        if S >= H:
            C[j] = 0

    # backward recursion through the tree
    for i in np.arange(N-1,-1,-1):
        for j in range(0,i+1):
            S = S0 * u**j * d**(i-j)
            if S >= H:
                C[j] = 0
            else:
                C[j] = disc * (q*C[j+1]+(1-q)*C[j])
    return C[0]


def binomial_tree_american_up_and_out(K,T,S0,H,r,N,sigma,opttype):
    # precompute values
    dt = T/N
    u = np.exp(sigma*np.sqrt(dt))
    d = 1/u
    q = (np.exp(r*dt) - d)/(u-d)
    disc = np.exp(-r*dt)
```

```
    # initialise asset prices at maturity
    S = np.zeros(N+1)
    for j in range(0,N+1):
        S[j] = S0 * u**j * d**(N-j)

    # option payoff
    C = np.zeros(N+1)
    for j in range(0,N+1):
        if opttype == 'C':
            C[j] = max(0, S[j] - K)
        else:
            C[j] = max(0, K - S[j])

    # check terminal condition payoff
    for j in range(0, N+1):
        S = S0 * u**j * d**(N-j)
        if S >= H:
            C[j] = 0

    # backward recursion through the tree
    for i in np.arange(N-1,-1,-1):
        for j in range(0,i+1):
            S = S0 * u**j * d**(i-j)
            if S >= H: # Check if the barrier is crossed
                C[j] = 0
            else: # 2 possible cases ofr the american one
                # option value obtained with risk neutral valuation equation
                equation_value = disc * (q*C[j+1]+(1-q)*C[j])
                value = 0 # value of the option if exercised early

                if opttype=='C':
                   value = max(0,S-K)
                else:
                    value = max(0,K-S)

                # the value we consider in the abbreviation is the maximum of the 2
                C[j] = max(value, equation_value)
    return C[0]

## Graphics

#Parameters Initialisation

K = 80          # strike price
T = 1           # time to maturity in years
H = 20          # up-and-out barrier price/value
```

```
r = 0.03       # annual risk-free rate
N = 1000       # number of time steps
sigma=0.3      # volatility
opttype = 'C' # Option Type 'C' or 'P'

AbsiS0=[]
Ord1S0=[]
Ord2S0=[]
def value_S0():
    for S0 in range (10,200,5):
        Absi.append(S0)
        Ord1.append(arbre_binomial_european_up_and_out(K,T,S0,H,r,N,sigma,opttype))
        Ord2.append(arbre_binomial_american_up_and_out(K,T,S0,H,r,N,sigma,opttype))
    return (Absi,Ord1,Ord2)

x= AbsiS0
y1=Ord1S0
y2=Ord2S0
# Graphics creation
plt.plot(x, y1,'blue', label='european')
plt.plot(x, y2,'red', label='american')

# Embellishment of the figure
plt.title("Evolution of the price of down and in CALL option when $S_0$ varies")
plt.xlabel('$S_0$')
plt.ylabel('Price of option')
plt.legend()

plt.show()

## K Graphic

# Parameters initialisation
S0 = 100       # initial stock price
T = 1          # time to maturity in years
H = 20         # up-and-out barrier price/value
r = 0.03       # annual risk-free rate
N = 1000       # number of time steps
sigma=0.3      # volatility
opttype = 'C' # Option Type 'C' or 'P'

AbsiK=[]
Ord1K=[]
Ord2K=[]
def value_K():
    for K in range (10,200,5):
        Absi.append(K)
```

```
        Ord1.append(arbre_binomial_european_up_and_out(K,T,S0,H,r,N,sigma,opttype))
        Ord2.append(arbre_binomial_american_up_and_out(K,T,S0,H,r,N,sigma,opttype))
    return (Absi,Ord1,Ord2)

x= AbsiK
y1=Ord1K
y2=Ord2K
# Graphics creation
plt.plot(x, y1,'blue', label='european')
plt.plot(x, y2,'red', label='american')

# Embellishment of the figure
plt.title("Evolution of the price of down and in CALL option when K varies")
plt.xlabel('K')
plt.ylabel('Price of option')
plt.legend()

plt.show()
```

End