Comparison of option pricing techniques using the Heston model

Anita Mezzetti
Master in Financial Engineering
EPFL
anita.mezzetti@epfl.ch

Abstract

The Heston model, thanks to its closed-form solution, allows to easily price European call options. This model-based solution is our benchmark to compare the performances of different pricers. We use the Monte Carlo method, also applying a variance reduction technique. Next, we focus on ML models by proposing to combine classification and regression, because this allows to obtain good results through relatively simple methods. In conclusion, we present a solution to improve the results in the ATM region, which is the hardest one to predict. This combination idea, being model-independent and particularly flexible, aims to be a good answer in many situations, not only for the Heston model.

1 Introduction

It is well known that an option is a security which gives its owner the right, but no obligation, to trade a certain number of shares of a specified common stock at a fixed price at a specific time or at any time, depending on the kind of option. The fixed priced is called *strike price* and the date within which exercising the option is the *maturity date*. For instance, a *call* option gives the right to buy the shares and a *put* option the right to sell them [1].

The option price, or option premium, is the price that the buyer of the option contract pays to become the owner of that option and exercise all its connected rights. An option pricing model is a formula or a mathematical framework for determining the correct price for an option. These theoretical models take into account the strike price, the maturity date, the price of the underlying asset and the standard deviation of the underlying asset returns. Of course, finding a good option pricing model is a complex task, because the price of the underlying or other factors may change over the life of the contract. Hence, the majority of option pricing models lay on certain restrictive assumptions that may affect their accuracy.

Historically, option pricing theory lived a revolutionary change in 1973, when Fisher, Black and Merton presented the first equilibrium pricing model for simple calls and puts [2]. Since then, great strides have been made. For instance, consider the risk-neutral evaluation for options introduced by Cox and Ross [1]. Nonetheless, option pricing is still a challenge. Besides the increasing complexity of the products, obtaining fair prices requires more complex models of the underlying asset behaviour. The Back-Scholes model, after almost 50 years, is still extensively used but, despite its success, the model fails to capture empirical phenomena, like a leptokurtic distribution of returns or the existence of a volatility smile [3].

Many financial engineering models have tried to relax the Black-Scholes (BS) model restrictions, as the Heston model, which we cover in section 2. However, even if they are more correct, they are much more computational expensive and they require parameters calibration. Some do not even present a closed form solution and they are all still restricted in terms of economical assumptions. Therefore, a data driven approach, which does not follow parametric models, is justified [3].

Machine Learning algorithms are particularly suitable if we consider a functional mapping between the contract terms, the inputs, and the option price, the output. These techniques do not rely on restrictive assumptions or specific parameters, so they are flexible and robust to specification errors of classical models. Already in

1994, Hutchinson et al. [4] proved that Neural Networks (NN) are an excellent way to approximate the option pricing function, being more accurate and computationally more efficient than previous models. Therefore since the 90s, researchers has been focusing on different algorithms, depending on the derivative to price and the most recent available techniques. A particularly interesting line of research is the one which tries to adapt intuitions from other fields, like Physics or Biology, to option pricing. For instance, in 2000 Grace [5] suggested that Genetic algorithms also fit option prices or, in 2020, Stamatopoulos et al. [6] proposed to use quantum computers for option pricing.

In this report, we first analyse the Heston parametric model, which has a closed form option pricing solution. The underlying stock and volatility dynamics given by the Heston model are used to generate many stock paths and, consequently, many option prices. These generated data are given as input for the training of Machine Learning models, which we cover in the second part of this work.

2 Heston model

In 1993, Heston [7] proposed a stochastic volatility model that, still now, is the most well known and popular among all the stochastic volatility models. According to it, the price od the underlying S_t is governed by

$$\frac{dS_t}{S_t} = \mu dt + \sqrt{V_t} dW_t^1 \tag{1}$$

$$dV_t = \kappa(\theta - V_t)dt + \sigma\sqrt{V_t} dW_t^2; \tag{2}$$

where dW_t^1 and dW_t^2 are Brownian motions with correlation ρ given by $dW_t^1 dW_t^2 = \rho dt$. Specifically, we can determine dW_S from the following equation

$$dW_S = \rho dW_V + \sqrt{1-\rho^2} dW_i, \tag{3}$$

where W_i is an independent Brownian Motion.

Regarding the parameters, μ is the instantaneous expected rate of return of the underlying asset, V_t is the variance at time t, θ is the long time variance, κ is the variance mean-reversion speed and σ is the volatility of the variance process.

The Heston model generalises the BS model by assuming a stochastic volatility. In particular, V_t is a mean reverting Cox-Ingersoll-Ross (CIR) process and follows a square-root diffusion. This change is essential because the BS model, which assumes constant volatility, does not reflect real data. Note that if $2\kappa\theta > \sigma^2$ the variance of a CIR process is always non-zero and the deterministic part is asymptotically stable if $\kappa > 0$ [8].

Among the drawbacks of this model, we have that it is hard to find proper parameters to calibrate it and the produced prices are sensitive to these parameters. Therefore, the result strongly depends on the calibration. Moreover, the standard Heston model usually fails to create a short term skew as strong as the one given by the market [8].

However, Heston's dynamic shows many desirable and rare properties. First, the volatility follows a mean-reverting process, which is consistent with the financial markets. Second, it introduces correlation shocks between asset returns and volatility. This allows the model to capture the dependence between the underlying and its volatility, which is a well-known feature of real financial data. Moreover, Heston's setting takes into account the non-lognormal distribution of the assets returns, the leverage effect and the BS volatility surfaces, built using this model, look like the empirical ones. Therefore, the Heston model is a versatile tool that encapsulates many of the specific characteristics of financial markets [9, 8].

2.1 Option Pricing Closed Formula

Supposing that the market is arbitrage free, a European call, with maturity T and strike price K, has a payoff equal to $max(0, S_T - K) = (S_T - K)^+$ and its present value, which corresponds to its price, is

$$C(S, \tau) = e^{-r\tau}\mathbb{E}[(S_T - K)^+], \tag{4}$$

where τ represents the time to maturity. We suppose the interest rate is constant. An option pricing model should give an explicit formula to calculate $C(S, \tau)$.

Assuming the stock price follows a dynamic equal to $dS_t = \mu S_t dt + \sigma S_t dW_t$ and using the Ito lemma, Black and Scholes [2] proved that a call European option must follow the well known Black-Scholes PDE:

$$rS_t \frac{dC_{\text{BS}}}{dS} + \frac{dC_{\text{BS}}}{dt} + \frac{1}{2}\sigma^2 S^2 n \frac{d^2 C_{\text{BS}}}{dS^2} - rC_{\text{BS}} = 0. \tag{5}$$

Providing some boundary conditions, they solved this PDE and they found out that, if the underlying stock follows the dynamic of their model, the call price is

$$C_{\text{BS}}(S,\tau) = S_t N(d1) - e^{r\tau} K N(d2), \tag{6}$$

where $d_1 = \frac{log(\frac{S_t}{K}) + (r + \frac{\sigma^2}{2})\tau}{\sigma\sqrt{\tau}}$ and $d_2 = d_1 - \sigma\sqrt{\tau}$. More details can be found in their paper [2].

For our work, their findings are essential because Heston, borrowing from the Black-Scholes model, suggested that the solution to his model should have a similar form:

$$C_H(S,\tau) = SP_1 - Ke^{-r\tau}P_2. \tag{7}$$

The first term represents the present value of the spot asset upon optimal exercise and the second one is the present value of the strike price payment. It was convenient to continue the calculations in terms of the logarithm of the stock price:

$$x = \ln(S). \tag{8}$$

Matching his idea with the Black and Scholes research, Heston found that that P_1 and P_2 must satisfy the PDEs

$$\frac{1}{2}v\frac{d^2P_j}{dx^2} + \rho\sigma v\frac{d^2P_j}{dxdv} + \frac{1}{2}\sigma^2 v\frac{d^2P_j}{dv^2} + (r + u_j v)\frac{dP_j}{dx} + (a_j - b_j v)\frac{dP_j}{dv} + \frac{dP_j}{dt} = 0, \tag{9}$$

where $u_1 = 1/2$, $u_2 = -1/2$, $a = \kappa\theta$, $b_1 = \kappa + \lambda - \rho\sigma$, $b_2 = \kappa + \lambda$ for j = 1,2. The value λ represents the price of volatility risk and, for simplicity, without loss of generality, we suppose it is zero in this work.
As for the BS model, also in this case there are some boundary conditions that must be satisfied:

$$P_j(s,v,T;ln(K)) = \mathbf{1}_{x \geq ln(K)}. \tag{10}$$

Therefore, P_j can be interpreted as the conditional probability that the option expires in the money. These probabilities are not directly available in a closed form, but it can be proved (see the Appendix in [7]) that their characteristic functions, $f_j(x,v,T;\phi)$, satisfy the same equation Equation 9, subject to the terminal condition $f_j(x,v,T;\phi) = e^{i\phi x}$.

We can write the characteristic function solution:

$$f_j(x,v,t;\phi) = e^{C(\tau,\phi) + D(\tau,\phi)v + i\phi x}, \tag{11}$$

where

$$C(\tau;\phi) = r\phi i\tau + \frac{a}{\sigma^2}\{(b_j - \rho\sigma\phi i + d)\tau - 2\log\big[\frac{a - ge^{d\tau}}{1-g}\big]\}] \tag{12}$$

$$D(\tau;\phi) = \frac{b_j - \rho\sigma\phi i + d}{\sigma^2}\big[\frac{1 - e^{d\tau}}{1 - ge^{d\tau}}\big] \tag{13}$$

$$g = \frac{b_j - \rho\sigma\phi i + d}{b_j - \rho\sigma\phi i - d} \tag{14}$$

$$d = \sqrt{(\rho\sigma\phi i - b_j)^2 - \sigma^2(2u_j\phi i - \phi^2)}. \tag{15}$$

In conclusion, we can revert the characteristic function to get the probabilities:

$$P_j(s,v,T;ln(K)) = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty Re\big[\frac{e^{-i\phi ln(K)}f_j(x,v,t;\phi)}{i\phi}\big]d\phi. \tag{16}$$

Thanks to the fact that the characteristic function always exists and the integral converges [10], we arrive to a closed form solution substituting Equation 16 in 7.

The solution is not so easy and immediate as the Black-Scholes one, but this model is remarkably more flexible. This closed form precise solution is the benchmark by which we evaluate the performance of all the other methods.

3 Monte Carlo Methods

In general, for a simple Monte Carlo (MC) problem, we express the quantity we want to know as the expected value of a random variable X, such as $\mu = \mathbb{E}(X)$. Then, we independently and randomly sample $X_1, ..., X_n$ from its distribution and we take their empirical mean

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} X_i \tag{17}$$

as estimation of μ.

The law of large numbers ensures that this estimate converges to the correct value as the number of draws increases. However, usually, the situation is a bit more complex in the sense that X itself is a function of another random variable [11, 12]. In this case, Kolmogorov's law of large numbers implies

$$\mathbb{E}[f(X)] = \lim_{n \to \infty} \frac{1}{n} \sum_{k=1}^{n} f(X_k). \tag{18}$$

Therefore for a fixed n, the Monte Carlo methods provide unbiased estimations of $\mathbb{E}[f(X)]$ [11].

Regarding the error, the central limit theorem provides information about the likely magnitude of the error of the estimation. We discuss more in-depth the MC methods error in subsection 3.3.

3.1 Stock prices simulations

Given the parameters of a model and the initial conditions, we can simulate the dynamics of the stock price. For the Heston model, the parameters are $\mu, \kappa, \theta, \sigma$ and ρ, and the initial conditions are V_0 and S_0. From this, using the dynamics given by Equation 1 and 2, we can easily simulate the whole path of the stock until the maturity T. An example is given in Figure 1. Let us suppose that, starting from the same parameters and the same initial conditions, we compute many paths. Each is potentially different, due to the two Brownian Motions. Thus, we obtain different S_T values with different simulation runs. One may ask what is the distribution followed by S_T. Due to the Brownian Motion properties, it can be proved that S_T tends to be normally distributed increasing the number of simulations. In Figure 2a we plot the stock paths of the MC simulations, while Figure 2b shows the probability distribution of S_T, which, as anticipated, tends to be Gaussian. The stock prices simulations underlie the Monte Carlo methods.

3.2 Monte Carlo for Option Pricing

A fundamental implication of asset pricing theory is that, under uncertainty circumstances, the price of a derivative security can be represented as an expected value. Therefore, valuing derivatives, such as options, reduces to computing expectations. For this reason, MC is a particularly suitable method for option pricing [11].

For European options, only the end-point of the paths is relevant for the payoff. After creating an ensemble of random values for the value of the underlying assets at maturity $S_j(T)$, see subsection 3.1, we can easily find the value of the payoff at maturity for each path using Equation 4. Taking the average of the discounted payoffs of each simulated path, we obtain the price of the option today. For an American option however, the situation is somewhat more complex, but this is out of the scope of this work.

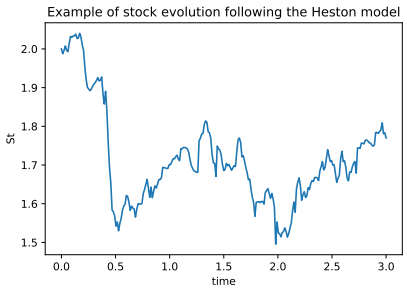
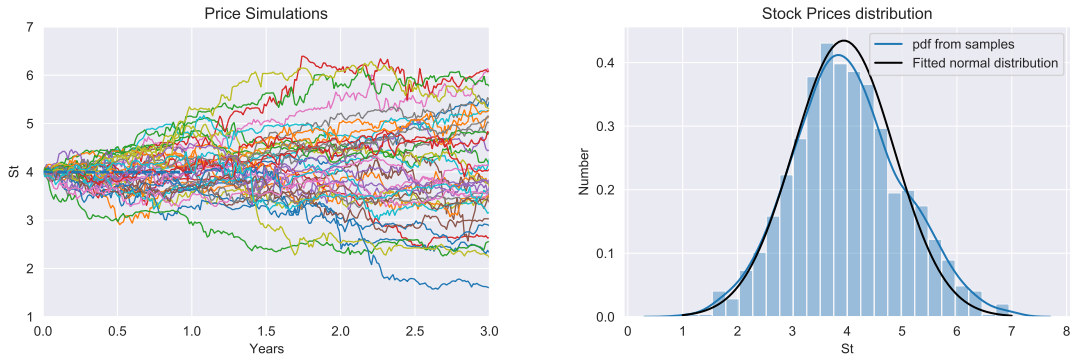


Figure 1: Example of stock evolution following the Heston model.



(a) Plot MC simulation.

(b) Plot distribution of stock prices at maturity

Figure 2: Monte Carlo simulation.

3.3 Variance reduction techniques

Assume $\mathbb{E}([(X)^2] < \infty$ and denote the standard deviation of X by $\sigma := \sqrt{\mathbb{E}[(f(X) - \mathbb{E}[f(X)])^2]}$ and the Monte Carlo error by

$$\epsilon_n := \mathbb{E}[(f(X)] - \frac{1}{n}\sum_{k=1}^{n} f(X_k). \tag{19}$$

The mean square error satisfies $\sqrt{\mathbb{E}[\epsilon_n(f,X)]} = \frac{\sigma}{\sqrt{n}}$. Therefore, the error in a direct Monte Carlo simulation goes as $\sigma/\sqrt{n}$. Note that the control of the error is only in expectation: it does not give any information on the error of the actually simulated paths.

The error is proportional to the standard deviation of the random variable $f(X)$. So, the efficiency of MC simulations can be improved by reducing $Var[f(X)]$ through a variance reduction technique. There are two main strategies for variance reduction: taking advantage of the features of the model to adjust the simulation outputs and reducing the volatility of simulation inputs [11]. More specifically, among the different available techniques, we discuss the antithetic variables method in the following section.

3.3.1 Antithetic variables

The method of antithetic variables was introduced by Hammersley and Morton in 1956 and it tries to reduce the variance by introducing negative dependence between pairs of replications. If X and Y are independent, the variance of their sum is the sum of their variances. However, if they are not, the contribution of the covariance must be added. Formally, $Var(X+Y) = Var(X) + Var(Y) + 2Cov(X,Y)$, where $Cov(X,Y) = \rho_{XY}\sigma_X\sigma_Y$. If the correlation ρ_{XY} is negative, the variance of $X+Y$ is smaller than the sum of the variances.

The antithetic variables method is based on the idea that estimating μ by the averaging of i.i.d. random variables using *pairs* of negatively correlated random variables–again with expectation μ–reduces the error. Therefore, if we approximate μ by averaging the pairs, we should get an estimator with smaller variance.

We can substitute the naive MC estimator given by Equation 17 with the antithetic estimator

$$\hat{\mu}_n^{av} = \frac{1}{2n}\sum_{i=1}^{n/2}\frac{X_i + \bar{X}_i}{2} = \frac{1}{2n}\sum_{i=1}^{n/2}\frac{f(u_i) + f(\bar{u}_i)}{2}. \tag{20}$$

$\bar{X}_i = f(\bar{u}_i)$, where $\bar{u}_i \sim \text{Uniform}(0,1)$ is negatively correlated with u_i. The most used way to introduce this negative correlation is to define $\bar{u}_i = 1 - u_i$.

To apply this method to the Heston model, we slightly modify this theory. Instead of starting from a uniform distribution and a function f, we only change the sign of the stochastic component in the volatility dynamic

(see Equation 2). Namely,

$$dV_t = \kappa(\theta - V_t)dt + \sigma\sqrt{V_t}dW_t^2 \tag{21}$$

$$d\bar{V}_t = \kappa(\theta - \bar{V}_t)dt - \sigma\sqrt{\bar{V}_t}dW_t^2. \tag{22}$$

In this way, the variance of this estimator is $\frac{\sigma^2+\rho_{XY}}{n}$, which is lower than the naive MC variance. We empirically show this result in subsubsection 5.0.2.

4 Machine Learning Models for Option Pricing

Classical option pricing models are built on an underlying process that reproduces the empirical relationship among option data (strike price, time to maturity, type), underlying stock data and the option premium, which is observable in the market. On the other hand, Machine Learning methods do not assume anything about the underlying process: they aim at estimating a function between the input data and premiums, by minimising a given cost function (usually the mean squared error between the model price and the observed price on the market) to reach good out-of-sample performance.

More specifically, regression links input variables to the output variables of a training set in order to predict outputs for new inputs. Apart from *prediction*, another goal of regression is *interpretation*, which consists in understanding the effect of inputs on the output. In order to do that, let use suppose to have a dataset with a certain number of features. One of these attributes, the call price in our case, is what we want to predict and we can call it y. While all the other features (or some of them) constitutes X, the set of properties needed to predict y. Mathematically, y is a function of X: $y_n = f(X_n)$ for each n (for each point). Our goal is to try some ML methods in order to find the best f.

4.1 Loss function

The loss function $\mathcal{L}$ qualifies how well the model explains the data or, from the opposite point of view, how *costly* the mistakes are. If $e_n := y_n - f_w(x_n)$, then

$$\mathcal{L}_w = \frac{1}{N}\sum_{n=1}^{N} \text{cost}(e_n). \tag{23}$$

The function *cost* depends on what loss function we decide to use. Usually, symmetric cost functions, which equally penalise positive and negative errors, are desirable. For instance in subsubsection 4.1.1, we describe the root mean squared error.

In general, our goal is to find the optimal w^* that minimises the cost function:

$$\min_w \mathcal{L}_w. \tag{24}$$

Therefore, our learning problem, which aims to find the optimal model to define the call prices, can be formulated as an optimization problem.

4.1.1 RMSE

The root mean squared error (RMSE) is the square root of the average of the squared differences among the real results and the predictions:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(C_j - \hat{C}_j)^2}. \tag{25}$$

where C_j is the *real* call price and $\hat{C}_j$ is the estimated call price. Essentially, it is the square root of the mean square error. A useful property of the RMSE is that, thanks to its definition, it can be interpreted as the distance in L_2 norm between two vectors.

The Heston model has an explicit numerical solution for pricing a call option; so C_j is the price calculated using this closed form solution. Therefore, in this work, the error of a regression technique is given by

$$\text{RMSE}_{\text{method}} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(C_{\text{closed-form},j} - \hat{C}_{\text{method},j})^2}. \tag{26}$$

4.2 Machine Learning Models

In this chapter, we describe some of the most important and interesting ML techniques employed in this work. Some are suitable only for classification or regression, while some others can be used in both cases.

4.2.1 Logistic Regression

In general, a classifier divides the input space into a collection of regions belonging to each class. In our case, we only have two classes which divide the options between the one which are executed and the ones which are not, according to their price. In other words, it is a binary classification problem [13].

Binary classification can be interpreted as a regression with 0 and 1 as possible targets. The function used to transform an input whose range can be $(-\infty, \infty)$ to $(0, 1)$ is the logistic function: $\sigma(z) := \frac{e^z}{1+e^z}$. However, the binary classification output allows only two labels $\{0, 1\}$, so we need to pass from $(0, 1)$ to $\{0, 1\}$. The idea is the following: using the training set, we learn a weight vector w and a scalar w_0. Given another feature vector x, we predict the probability of the two labels:

$$p(1|x, w) = \sigma(x^T w + w_0) \tag{27}$$

$$p(0|x, w) = 1 - \sigma(x^T w + w_0). \tag{28}$$

Therefore, we do not directly predict the labels, but their probability and thus this method is named *regression* [13].

4.2.2 Linear Regression

As the name suggest, Linear Regression is a method that supposes a linear relationship between inputs and outputs. The main advantages of this method is that it is easy to implement and interpret. The function f is described by two paramenters w_0 and w_1: $f(x) = w_0 + w_1 x$, where w and x are scalars, in case of only one feature in X, or vectors, in case of many features. However, it does not perform well in case of non-linear relationships between the inputs and the outputs.

4.2.3 XGBoost

Literally, XGBoost stands for eXtreme Gradient Boosting [14]. This algorithm is a decision-tree-based ensemble algorithm that uses a gradient boosting framework, designed for performance and speed. In general, boosting is a method of converting weak learners, as decision trees, into strong learners. More specifically, gradient boosting trains many models in a gradual, additive and sequential way in order to improve the loss function [15].

In this work, we decide to use this algorithm because, thanks to these advantages, it tends to perform very well compared to other tree-based methods.

4.2.4 Neural Networks

In order to understand the Neural Networks (NNs) method, let us first introduce the *perceptrons*, which can be seen as the *foundation* of any NN. A perceptron is a structure which takes several inputs and and return a binary output, as it is shown in Figure 3.In other words, it is a model that makes decisions by weighting the evidence. The output, 0 or 1, depends on a threshold value: if $\sum_j w_j x_j$ is less than the threshold, we have 0; otherwise we have. 1 Of course, by varying the weights and the threshold, we get different decision-making models.

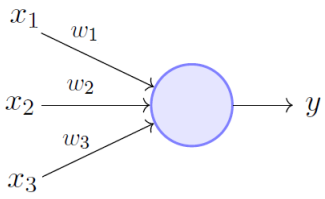


Figure 3: Perceptron model [16].

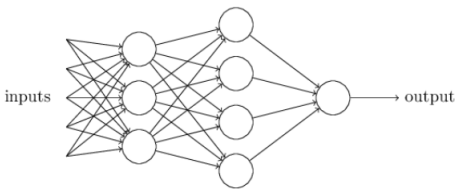


Figure 4: NN model [17].

If we suppose to have a sufficiently complex network of perceptrons, the model is likely to being able to evaluate harder situations. We can organise this network in columns of perceptrons, or *layers*, where each layer output is used as input of the next one. This system, shown in Figure 4, represent a NN, whose complexity can increase by expanding the number of layers or the number of perceptrons in each layer.

There are different kinds of layers: the input layer is responsible for receiving the inputs and the output layer is responsible for producing the final result. The hidden layers, between the other two, allow for the function of a NN to be broken down into specific transformations of the data.

$w_{ij}^{(l)}$ denotes the edge from node i in layer $l-1$ to node j in layer l. The output of node j in layer l is denoted by $x_j^{(l)}$ and is given by

$$x_j^{(l)} = \phi(\sum_i w_{ij}^{(l)} x_j^{(l-1)} + b_j^{(l)}), \tag{29}$$

where $b_j^{(l)}$ is the *bias term* and the function ϕ is the *activation function* [13].

Activation Functions The choice of the activation function is an essential block in a NN: the activation function of a node determines the output of that node, which, in turn, is the input for the next layer or the output of the model.

For each problem, we need to choose the right activation function for the output layer and all the other ones. In case of classification, the **Sigmoid** $\phi(x) = \frac{1}{1+e^{-x}}$ is a suitable choice as an output activation, while regression needs no activation function at all in the output layer. In general, they are non-linear functions; otherwise the NN is only a highly factorised Linear Regression.

A very popular choice is the rectified linear unit (**ReLU**), defined by $\phi(x) = \max(0, x)$. This function is always non negative, unbounded and, for positive x, its derivative is 1. However, for negative values of x, the derivative is zero. This is a problem, also called *vanishing gradients problem*. The main issue is that, if too many nodes evaluate to zero, then the gradients will evaluate to zero as well. Hence, the gradient descent updates will not be substantial.

In order to solve the 0-gradient issue, a solution is to add a very small slope α when it should be zero. This represents the Parametric ReLU (**PReLU**), defined by $f(x) = \max(\alpha x, x)$.

5 Code and Results

In this chapter, we focus on explaining the code and the obtained results. Note that the full code can be found on GitHub (click here).

5.0.1 Explicit solution, stock path generator and data generation

The first two sections of the code are dedicated to the implementation of all the needed functions to calculate the numerical solution of the Heston model, following Equation 16 , and the required methods to generate the stocks paths necessary for MC, see subsection 3.1. The function which calculates the numerical solution is called `call_price`, while `stock_price_generator` generates the simulation.

Then, we initialise the model parameters. Note that the choice of these parameters does not influence the results of this work, so they are chosen arbitrarily. In particular, we fix k at 2, while S_0 varies in a range from 0.5 to 4. This leads to the creation of a dataset, named `df1`, of two columns: the first one for S_0 and the second for the relative price calculated using Equation 4, where S_T is given by the stock evolution.

5.0.2 Monte Carlo methods

This section of the code is reserved to the MC methods. First, we create a function called `find_mc_price`, whose role is to return the MC price starting from a vector of simulations of S_T, by calculating the empirical mean as explained in section 3. Following the antithetic variables method presented in subsubsection 3.3.1, we finds also the MC prices using this technique. As anticipated above, the S_T distribution of the MC method with antithetic variables has a standard deviation which is lower than the MC method. However, the difference between the two standard deviations is not very high: around 0.02.
Figure 5 shows our results: the two solutions are quite similar and tend to reproduce correctly the numerical solution trend.

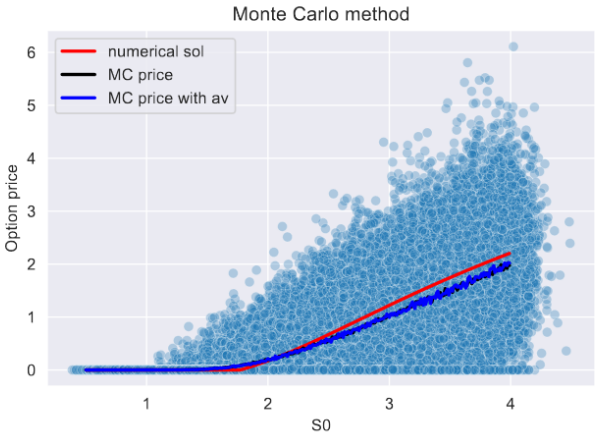


Figure 5: Plot MC and MC with Antithetic variables solutions.

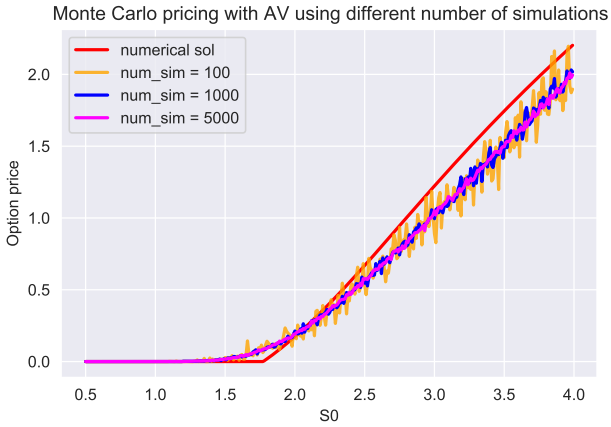


Figure 6: Comparison among different number of simulations.

Before analysing the error of these techniques, we try to compute also the MC method with antithetic variables for other two different numbers of simulations. The goal is to understand if, increasing the number of simulations, the error decreases. Figure 2a reveals that, as expected, the simulated prices are much more stable if we increase the number of simulations. So, we can conclude that an higher number of simulations helps to obtain less volatile results. However, increasing the number of simulations increases computation time and thus who wants to use the MC methods should balance these two aspects.

In terms of error, we use RMSE as explained in subsubsection 4.1.1. We apply the MC method and its variations to each of the initial stock prices included in the S_0 vector and we obtain the RMSEs contained in Table 1. All other things being equal, the antithetic variables allows the method to be 10,21% faster, but it slightly increases the RMSE (by 0.03 in this case). We believe that, considering that the main problem of MC is the computation time, it makes sense to use the antithetic variable technique.

	MC	MC with AV	MC with AV	MC with AV
Number of simulations	1000	500	1000	5000
RMSE	0.141	0.156	0.138	0.139

Table 1: RMSE for the MC methods.

5.0.3 Machine Learning models

In subsection 4.2 we explain that, to run a ML model, we need one or more input features, X, and an output feature, y. For this work, X includes only the initial stock price and y is the European call price. As data, we use the dataset `df1` previously generated, which is composed by 70000 rows. Please note that, for all the methods which require hyperparameters, but NNs, we perform a Grid Search in order to identify the optimal ones. For the sake of brevity, we report only the results obtained with the best hyperparameters.

Looking at Figure 11a, we notice that the distribution of the prices in the dataset is drastically unbalanced: many options have zero price. Usually, unbalanced datasets are a problem when we run classification techniques but, also for regression models, this imbalance may have negative consequences. For example, looking at the numerical solution for the Heston model in Figure 7, it is clear that the price is asymptotically linear when S_0 increases (ITM region). However, when the option is OTM, the price is zero. Therefore, the linear regression cannot give good results for the whole dataset, which includes both ITM, ATM and OTM options. This kind of behaviour requires some more complex techniques, which would probably take much longer. Nevertheless, if we first apply a classification method, which separates zero and non-zero option prices, then we can run the regression only for the non-zero option prices, increasing our possibility to capture the right trend with easier methods.

In terms of functions, let us call $c(\cdot)$ the classification function, which returns 0 if the option is classified as *non-executed* (price=0) and 1 if it is classified as *executed* (price>0). On the other hand, $r(\cdot)$ is the regression function: it is trained only on the options classified as *executed* and it returns the predicted price. In other words, the final price is zero if c returns zero and by the result of r in the other case. Mathematically, we compose these two functions:

$$r(c(s_0)) = r(s_0) \cdot c(s_0) = \begin{cases} 0 & \text{if } c(s_0) = 0 \\ r(s_0), & \text{if } c(s_0) = 1 \end{cases} \tag{30}$$

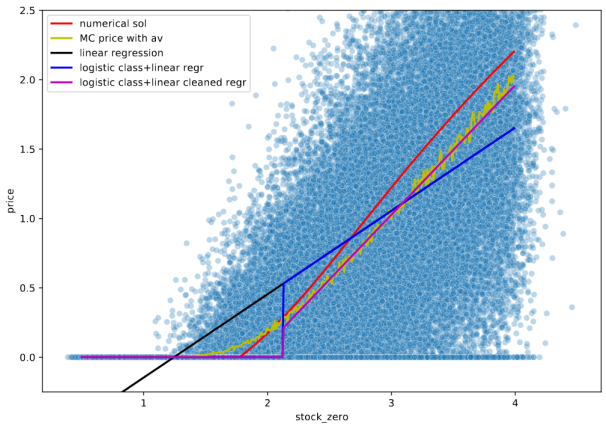


Figure 7: Benefits of combination of classification and regression.

for each s_0 in S0. This does not affect our capacity to predict also zero-value options (captured by the classification) and simplifies the regression part.

To give a graphical idea of the benefits of this combination strategy, we use Figure 7:

- *red line*: numerical solution;
- *red line*: Monte Carlo solution with antithetic variables;
- *black line*: Linear Regression trained on the whole dataset;
- *blue line*: classification combined with Linear Regression trained on the whole dataset;
- *magenta line*: classification combined with Linear Regression trained only on options classified as executed;

Not only Logistic Regression is able to correctly classify most of the zero price options, but Linear Regression significantly improves when it is trained only on the executed options: the magenta line is much more similar to the numerical solution compared to the blue line.

Considering all these benefits, we decide to combine classification and regression. Therefore, we implement some methods for both, in order to choose the best one in each case. After that, in subsubsection 5.0.6, we take care of the ATM region, which requires a deeper analysis.

5.0.4 Classification

For the classification part, we try Logistic Regression, KNNs and XGBoost classifier. For all these methods, we plot the confusion matrix in Figure 13 and the ROC curve in Figure 13. Remind that the more the area under the ROC curve is near to one, the better the model performs.

Their performances results are shown in Table 2. It seems like XGBoost is the best method in terms of precision. However, the main problem for this work are *false negatives*. Indeed, if an option is wrongly classified as executed (false positive), it is not a big problem, because it still passes through the regression part. On the other hand, if we wrongly classify an option as non executed, we set its price to zero when it is not. Therefore, more than precision and accuracy, we are interested in false negatives (shown in Figure 13) and in recall, which measures how many relevant options are selected. Therefore, the most suitable method is Logistic Regression. In terms of ROC curve, all the methods work quite well.

In this case, the classification is linearly separable, because we expect to have a unique threshold between zero and non-zero values. Hence, NNs are not a suitable to choice to capture this kind of behaviour and, as we have already seen, Logistic Regression is sufficient. Indeed, we expect NNs to overfit to underlie noise and lead to worse results. However, for sake of completeness, we empirically verify our hypothesis. Our NN has the following structure:

	LR	KNN	XGB	NNs
Precision	0.87	0.89	0.90	0.87
Accuracy	0.88	0.88	0.88	0.88
Recall	0.86	0.84	0.83	0.91

Table 2: Performances of classification methods.

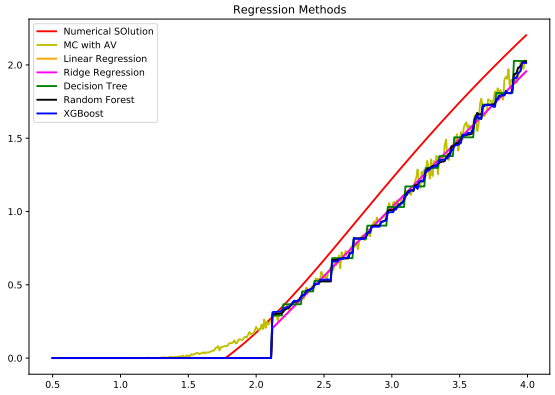


Figure 8: Comparison different regression methods

- Fully connected input layer with 20 nodes and ReLu activation function;
- Fully connected hidden layer with 10 nodes and ReLu activation function;
- Fully connected output layer with 1 node and sigmoid activation function.

Surprisingly, our NN performs quite well: not only precision and accuracy are similar to the other methods, but recall is even better. As we can see from Figure 14, our NN finds one threshold which is a bit more to the left than Logistic Regression. This lowers the number of false negatives. However, to filter values for the regression, we still use Logistic Regression because it has the best trade-off between recall and training-time (0.10 seconds vs 61.20 seconds).

5.0.5 Regression

After we exclude from the regression all the options classified as non executed by the Logistic Regression, the dataset is much more balanced (see Figure 11c). For this part, we try the Linear Regression, Ridge Regression, the Decision Tree approach, the Random Forest technique and the XGBoost regression.

Looking at the RMSE in Table 3, it is clear that all the models performs quite well. probably this is due to the fact that the trend tends to be linear, so it is a particularly easy behaviour to predict. This is what we mean when we say that, thanks to classification, we can use simple methods for the regression.

	LR	RR	DC	RF	XGB
RMSE	0.151	0.151	0.154	0.154	0.156

Table 3: RMSE regression methods.

We apply all these models to S0 and we plot the results in Figure 8. It is clear that, not only all the methods perform similarly, but they tend to be more similar to the MC simulation than the numerical solution. Probably, this is due to the fact that the MC methods, as the ML techniques, are based on generated paths, so they depend on the distribution of the simulations. On the other hand, the numerical price derives from a closed-form solution, which ignores the samples. We can conclude that the Linear Regression is the best technique. Considering the simple shape of the price curve, we decide not to implement an NN; the Linear Regression is sufficient.

5.0.6 ATM area

Another thing that is clear in Figure 8 is that the models fail to predict the price of the options in the area where $S_0 \sim k$, the *ATM area*. This is because any classification technique is more likely to misclassify the options in this area, which are harder to categorise than the other options ($S_0 << k$ or $S_0 >> k$). Therefore, the majority of the false negatives are likely to be in the ATM region. To overcome this problem, we decide to include in the regression all the options in this area, ignoring the classification results.

Let us suppose that the ATM area corresponds to the 10% of the whole S0 range. After Logistic Regression, we identify the average point where the classification makes the step from non-executed to executed and we include in the regression all the options around this point. We perform both Linear Regression and XGBoost method considering also the ATM area.

Figure 9 shows that for Linear Regression, including the ATM area imporves the predictions in the ATM area, but this does not negatively affect the ITM region. This can be seen also analysing the RMSE in Table 4: the

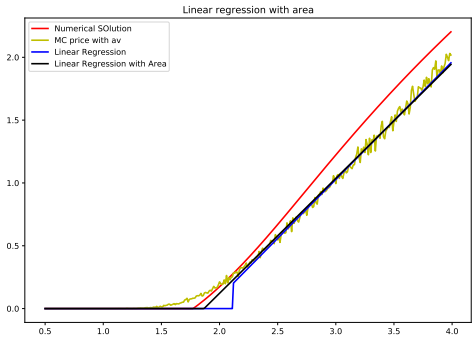


Figure 9: Linear Regression including the ATM area.

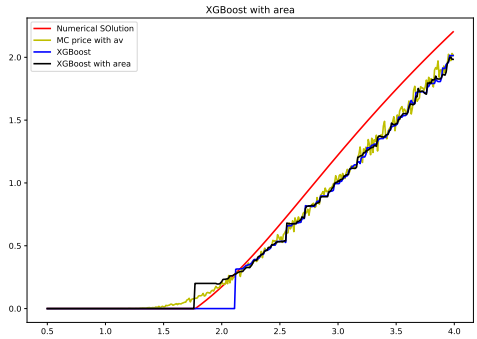


Figure 10: Decision Tree including the ATM area.

error decreasing drastically including also the ATM area. Note that, if we use as benchmark the MC price and not the numerical solution, the error is quite low. This proves that this simple combination of Logistic Regression and Linear Regression correctly reproduces the MC prices, which is much more time-consuming. In terms of RMSE, also the XGBoost technique performs quite well, but looking at Figure 10, we can conclude that Linear Regression is preferable.

Benchmark ATM area	Numerical sol without area	Numerical sol with area	MC with AV without area	MC with AV with area
Linear Regression	0.151	0.142	0.06	0.04
XGBoost	0.156	0.151	0.06	0.04

Table 4: RMSE with ATM area.

5.0.7 Features choice

What would have changed if we included more features, such as the initial volatility or the time to maturity? We try to run the ML techniques including more information and what we obtain is that the methods understands that the only influencing attribute is the initial price and they ignore the other features (showed in the code). Considering classification, for example, the balance between executed and non-executed options is almost the same after changing other features (see plots in the code).

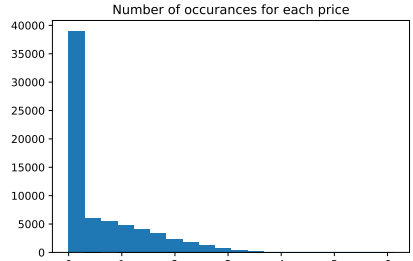
Another idea is to keep S_0 fixed and move k instead. Also in this case, we can follow the same steps. We verify that combining classification and regression still leads to good results which follows the numerical solution trend. Even if the curve is reflected, the ATM region is still the same, $S_0 \sim k$. Considering that the price curve shape is the same, for the sake of brevity, we only try Logistic Regression with Linear Regression.

6 Conclusion

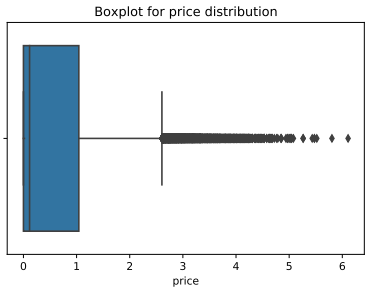
Monte Carlo remains a powerful and precise method fr option pricing. However, nowadays, there are many new techniques that allow for good results with considerably less efforts, as the one we propose in our work.

The trick of combining classification and regression is particularly suitable in this case, where we have a clear and unique threshold between zero and non-zero values and the behaviour for non-zero prices is relatively simple. Considering that the ML techniques are model-independent, this should be a good solutions for all the cases which show this kind of behaviour. In case the threshold is not unique and the prices are harder to predict, one should understand if this idea is still convenient. The positive aspect is that we can model the complexity of the predictor according to the needs. For example, a particularly complex price curve may need a NN for the regression part. While a behaviour which tends to be linear, as in our case, but with many thresholds, may benefit from a more complex approach for the classification. In any case, if the boundary between zero and non-zero values is not clear, probably it is better to solve the problem only as regression.

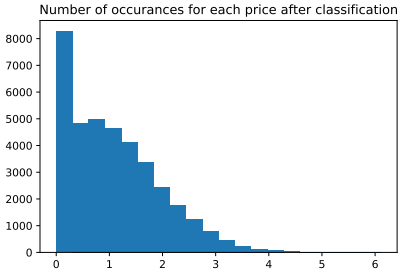
This work can be further extended. For instance, other techniques can be tried to solve the issue of the ATM area. One idea is to analyse separately this area, with a different regression methods than the ITM region, in order to chose the best technique for each area. It would be interesting to analyse also how to adapt this work to American vanillas or other kinds of options. The main problem is that there is not a closed-form solution for such options and thus we need to define another benchmark (the Monte Carlo solution for instance). Moreover, probably other parameters apart S_0/k would be influential and this must be considered.



(a) Distribution of call prices.



(b) Volatility call prices.



(c) Distribution of call prices after classification.

Figure 11: Call prices distributions

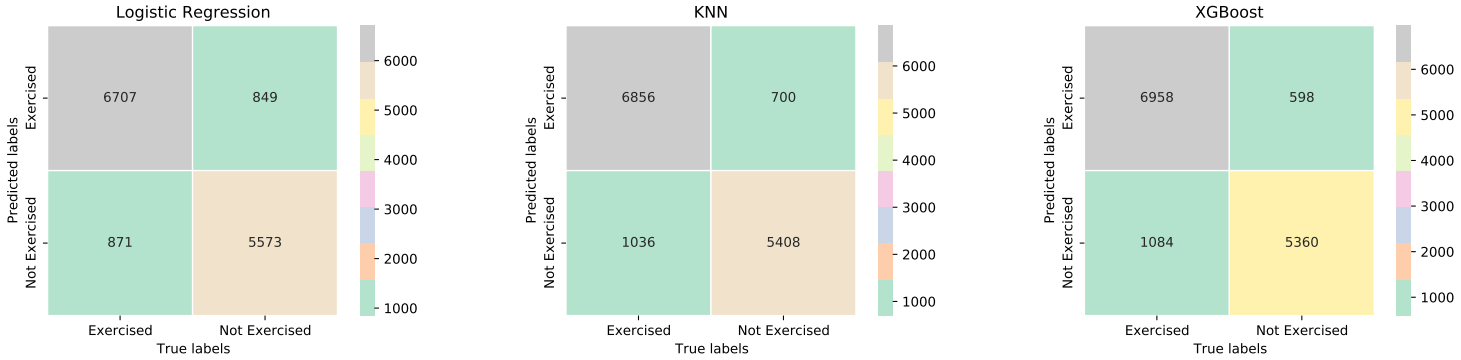


Figure 12: Classification confusion matrices.

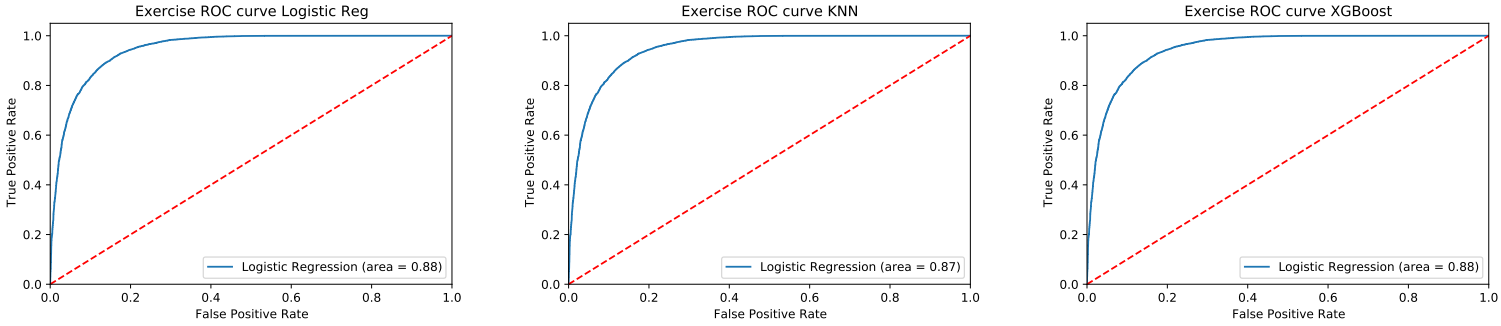


Figure 13: Classification ROC curve.

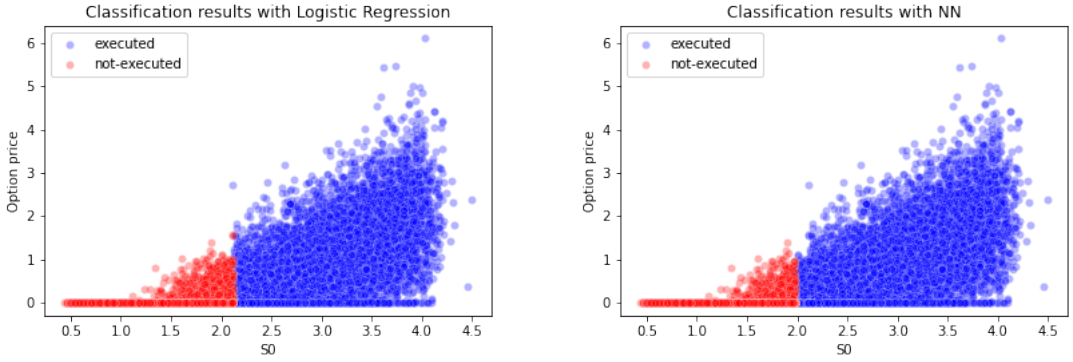


Figure 14: Division between executed and not-executed options classification.

References

[1] John C. Cox and Stephen A. Ross. "The valuation of options for alternative stochastic processes". In: *Journal of Financial Economics* 3.1 (1976), pp. 145–166. ISSN: 0304-405X. DOI: https://doi.org/10.1016/0304-405X(76)90023-4. URL: http://www.sciencedirect.com/science/article/pii/0304405X76900234.

[2] Fischer Black and Myron Scholes. "The Pricing of Options and Corporate Liabilities". In: *Journal of Political Economy* 81.3 (1973), pp. 637–654. URL: http://www.jstor.org/stable/1831029.

[3] Codruț-Florin Ivașcu. "Option pricing using Machine Learning". In: *Expert Systems with Applications* 163 (2021), p. 113799. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113799. URL: http://www.sciencedirect.com/science/article/pii/S0957417420306187.

[4] Hutchinson, Lo, and Poggio. "A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks". In: *The Journal of Finance* 49.3 (1994), pp. 851–889. DOI: https://doi.org/10.1111/j.1540-6261.1994.tb00081.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1994.tb00081.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1994.tb00081.x.

[5] Bruce K. Grace. "Black-Scholes option pricing via genetic algorithms". In: *Applied Economics Letters* 7.2 (2000), pp. 129–132. DOI: 10.1080/135048500351960. URL: https://doi.org/10.1080/135048500351960.

[6] Nikitas Stamatopoulos et al. "Option Pricing using Quantum Computers". In: *Quantum* 4 (July 2020), p. 291. ISSN: 2521-327X. DOI: 10.22331/q-2020-07-06-291. URL: http://dx.doi.org/10.22331/q-2020-07-06-291.

[7] Steven L. Heston. "A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options". In: *The Review of Financial Studies* 6.2 (Apr. 2015), pp. 327–343. ISSN: 0893-9454. DOI: 10.1093/rfs/6.2.327. URL: https://doi.org/10.1093/rfs/6.2.327.

[8] S. Mikhailov and Ulrich Nögel. "Heston ' s Stochastic Volatility Model Implementation , Calibration and Some". In: 2003.

[9] Ricardo Crisostomo. "An Analysis of the Heston Stochastic Volatility Model: Implementation and Calibration Using Matlab". In: *SSRN Electronic Journal* (Feb. 2015). DOI: 10.2139/ssrn.2527818.

[10] Maurice G. Kendall and Alan Stuart. "The Advanced Theory of Statistics. Volume 1". In: *Journal of the Royal Statistical Society Series C* (1977).

[11] Paul Glasserman. *Monte Carlo methods in financial engineering*. Springer, 2004. ISBN: 0387004513 9780387004518 1441918221 9781441918222.

[12] Art B. Owen. *Monte Carlo theory, methods and examples*. 2013.

[13] *Github: epfml/ML_course*. URL: https://github.com/epfml/ML%5C_course/tree/master/lectures.

[14] *XGBoost Documentation*. URL: https://xgboost.readthedocs.io/en/latest/.

[15] *Understanding Gradient Boosting Machines*. URL: https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab.

[16] *What is a perceptron?* URL: https://towardsdatascience.com/what-is-a-perceptron-210a50190c3b.

[17] Michael Nielsen. *Neural Networks and Deep Learning*. URL: http://neuralnetworksanddeeplearning.com/index.html.