

A MODEL OF *financial* ASSETS

ISFA LYON

 **Adrien Cortes**

ISFA Lyon

adrien.cortes@etu.univ-lyon1.fr

 **Arthur Bourdon**

ISFA Lyon

arthur.bourdon@etu.univ-lyon1.fr

 **Ilyes Grigah**

ISFA Lyon

Ilyes.Grigah@etu.univ-lyon1.fr

22 avril 2022

ABSTRACT

Numerous methods for modeling and calibrating financial assets have been developed in recent years. We propose here to compare the most recent methods, those using machine learning, to older methods, such as Heston models.

This work is an opportunity to take stock of the existing methods and to compare them.

Keywords QUANT GAN · WGAN-GP · RNN · CNN · LSTM · HESTON · STOCHASTIC VOLATILITY

Table des matières

1	Motivation for the study	2
1.1	From the old models..	2
1.2	Towards more recent models	3
2	Choice of the function $f_{\mathcal{W}}^3$	3
2.1	Reminder on neural networks and their interest	3
2.2	Vanishing and Exploding Gradient	5
2.3	The 5 blocks of our neural network	6
3	How to calibrate the function $f_{\mathcal{W}}^3$? Generative Adversial Networks	8
3.1	Introduction to GAN	9
3.2	Wasserstein GAN	10
3.3	Technical details for learning	11
4	Use of Heston model	13
4.1	Reminders of the Black-Scholes model and extension to the Heston model	13
4.2	Calibration of the Heston model	14
4.2.1	Calibration results	15

4.3	Model a price path of the underlying asset using the parameters of the Heston model	15
4.3.1	Transition from the real world to the risk-neutral world	15
4.3.2	Choice of the function f^4	16
5	Results analysis	17
6	Conclusion	22
A	Appendices on Heston model	23
A.1	Obtaining the formula for the price of a call in the Heston model	23
A.2	Transition from the real world to the risk-neutral world	26
A.3	Obtaining data on options and problems encountered	27
A.4	Constrained optimization of the cost function	27
B	Learning algorithm for GAN	28
C	An avenue to explore	28

We have chosen to work on a financial topic allowing us to make the link between the field of classical finance, where stochastic computation is widely used, and the more modern world of machine learning, which is growing rapidly.

For this work we did not use a ready-made library. We built each object by hand, and we detail it extensively throughout this paper. The construction of the neural network required more than 1000 lines of code and a total of 98h alone. The difficulty was to choose the right model. We tried multiple architectures and we will discuss them in the following.

Also a lot of time was spent on the Heston model and data recovery. The analysis of our results is also an important part of our contribution, as we propose to compare models that have not been compared before. We obtain quite qualitative results, and we refer to the last part to learn more.

We would like to thank our tutor, Mr. Stéphane LOISEL, whose advice has been precious so far. A part of our code is available on the online directory <https://github.com/c1adrien/TER.git>. It contains a part of our codes, the architecture of the neural network and a notebook to be able to run it and experiment these results.

We wish you an excellent reading !

1 Motivation for the study

1.1 From the old models..

The study of derivatives and, more broadly, of financial markets necessarily requires a modeling effort. Louis Bachelier [Bachelier, 1900] was the first to be interested in the models that could be used to simulate financial assets. The most widely used model today is that of Black & Scholes [Karoui et al., 1998], which proposes a remarkably robust modeling of financial assets. In order to fix the ideas, let us naively suppose that we wish to create a model for stocks, or at least their dynamics, without worrying about calibration.

Let us denote $(\Omega, \mathcal{F}, \mathbb{P})$ a filtered probability space. We want to simulate a discretized trajectory (S_1, \dots, S_n) of prices. We consider a sequence of independent and identically distributed variables $\varepsilon_i \sim N(0, 1)$.

It is assumed that there is a function $f^{original}$ such as :

$$f^{original} : \begin{cases} \mathbb{R}^n \longrightarrow \mathbb{R}^n \\ (\varepsilon_1, \dots, \varepsilon_n) \longrightarrow (S_1, \dots, S_n) \end{cases} .$$

All the rest of our work is a reflection on the best way to approximate $f^{original}$. A naive student would propose, inspired by [Bachelier, 1900], a first function f^1 allowing to simulate a trajectory of financial assets from these noises :

$$f_{\sigma}^1 : \begin{cases} \mathbb{R}^n \longrightarrow \mathbb{R}^n \\ (\varepsilon_1, \dots, \varepsilon_n) \longrightarrow (\sigma \varepsilon_1, \sigma (\varepsilon_1 + \varepsilon_2), \dots, \sigma (\varepsilon_1 + \dots + \varepsilon_n)) \end{cases}$$

where the only parameter to be calibrated here would be σ , which is intuitively the volatility. Apart from the fact that this model can give negative prices, and that we have not talked about the initial price of the asset, it seems very clear that very large assumptions about price dynamics have been made here. We do not want to discuss these assumptions and whether or not they are consistent with reality at this point, but simply point out that we have set a very narrow framework here.

With the arrival of Black & Scholes [Karoui et al., 1998], another student would have proposed a second function f^2 , solving the negative price problem we just gave, and allowing to get out a price process which is not necessarily a martingale (as it was the case with Bachelier) :

$$f_{\mu, \sigma}^2 : \begin{cases} \mathbb{R}^n \longrightarrow \mathbb{R}^n \\ (\varepsilon_1, \dots, \varepsilon_n) \longrightarrow (\exp(\sigma \varepsilon_1 + \mu - \frac{1}{2} \sigma^2), \dots, \exp(\sigma \sum_{i=1}^n \varepsilon_i + n(\mu - \frac{1}{2} \sigma^2))) \end{cases}$$

where μ is the trend parameter (the drift) and σ the volatility. Many authors have shown the limits of this model and have tried to improve it. One improvement is the Heston model that we will discuss later in this work.

1.2 Towards more recent models

Looking at the f^1 and f^2 functions, we think that it could be interesting to propose a more modern and flexible f^3 function, based on the emerging world of Deep-Learning :

$$f_{\mathcal{W}}^3 : \begin{cases} \mathbb{R}^n \longrightarrow \mathbb{R}^n \\ (\varepsilon_1, \dots, \varepsilon_n) \longrightarrow (r_1, \dots, r_n) \end{cases}$$

where (r_1, \dots, r_n) is a series of n returns of the asset we wish to simulate and \mathcal{W} is the set of parameters of the function. We model the returns because they are stationary and allow us to easily trace the price, which is not stationary.

We will now detail our choice for the function f^3 .

2 Choice of the function $f_{\mathcal{W}}^3$

First, we will give the form of the function $f_{\mathcal{W}}^3$. We will explain in a second time how we calibrated it. Keep in mind that the function f^3 is in fact a neural network.

We will approximate $f^{original}$ using a TCN (Temporal Convolutional Neural Networks). It is known for its ability to capture long term dependency structures and does not pose a learning problem.

We are going to define the different elements necessary to build the function $f_{\mathcal{W}}^3$. We started by building a succession of "block-temporals", which can be found in TCNs. Their characteristic is to take into account temporal dependencies. Empirically, we observed that the succession of several temporal blocks gave better results to capture temporal dependencies.

First of all, and to fix the ideas, it seems good to recall the basics concerning neural networks by introducing the feed forward neural network.

2.1 Reminder on neural networks and their interest

The feed forward neural network is the first type of neural network built. We use here the notations of [Mishachev, 2017] in order to quickly remind the reader this object.

Definition 2.1 (Activation function). A continuous and monotonic function $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ is called an activation function.

Remark. The most common activation functions are the sigmoid function, the hyperbolic tangent, the ReLU function, etc... We will come back to this in more detail in the definition and construction of our WGAN-GP.

Definition 2.2 (Multilayer Perceptron). Let $X \in \mathbb{R}^n$, $\sigma_1, \dots, \sigma_n$ be activation functions. Let $\Sigma(X)$ be defined by :

$$\Sigma(X) = \Sigma \left([x_1, \dots, x_n]^T \right) = [\sigma_1(x_1), \dots, \sigma_n(x_n)]^T$$

Let $n_1, n_2, \dots, n_k \in \mathbb{N}$, then the function $f : \mathbb{R}^n \rightarrow \mathbb{R}^{n_k}$ defined by :

$$f(X) = f(X; W) = \Sigma_k (W_k \cdot \Sigma_{k-1} (W_{k-1} \cdot \Sigma_{k-2} \cdots \Sigma_2 (W_2 \cdot \Sigma_1 (W_1 \cdot X)) \dots))$$

It is called a classical neural network or feed forward neural network with k layers. With $W_1 \in M_{n_1 \times n}(\mathbb{R}), \dots, W_i \in M_{n_i \times n_{i-1}}(\mathbb{R})$ are the weight matrices of the neural network.

Remark. We decide to write $\omega = (W_1, W_2, \dots, W_k)$ the parameters of the neural network. Thus, when we start the learning and we randomly initialize the weights \mathcal{W} of the neural network we note :

$$\exists \omega \in \mathcal{W} / f(X) = f_\omega(X)$$

The initial neural network works within the framework of a supervised learning. A loss function is defined to estimate how close the outputs produced are to the desired outputs and this then allows the neural network to learn from its errors.

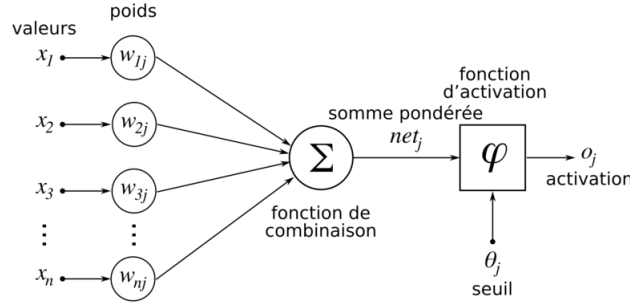


FIGURE 1 – Example of 1 neuron, a layer is composed of several neurons, that's why we used a matrix to designate the weights !

For this, we use the classical backpropagation algorithm. The texts [Mishachev, 2017, Parr and Howard, 2018] detail the calculation of the gradient. The idea is then to modify the weights W_i of the neural network using the error backpropagation algorithm based on the chain rule. The universal approximation theorem developed in [Hornik, 1991], shows that a neural network with 2 hidden layers can approximate with arbitrary accuracy any continuous function on a compact. (We just have to choose the right number of neurons in the hidden layer). This theorem is also called *the uniform approximation theorem*. It justifies the interest for neural networks. Thus if f is a continuous function on a compact, the gradient descent, or learning, consists in finding the best $w \in \mathcal{W}$ such that $f \simeq f_w$

Definition 2.3 (Linear). A linear layer is defined by :

$$y = xA^T + b$$

that we apply to the input data x .

Definition 2.4 (LeakyReLU). Applies the element-wise function :

$$\text{LeakyReLU}(x) = \max(0, x) + \alpha * \min(0, x)$$

This activation function is one of the most popular in neural networks. It makes the neural network non linear. It is superior to the other functions because it accelerates the convergence thanks to its linearity, and it is very cheap in number of operations for the computer. To avoid that the neurons "die" after no activation, we put a small α , so that we have a small learning even with a weak gradient.

Definition 2.5 (Conv1D). Applies a 1D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, L) and output (N, C_{out}, L_{out}) can be precisely described as :

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_j}, k) \star \text{input}(N_i, k)$$

where \star is the valid cross-correlation operator, N is a batch size, C denotes a number of channels, L is a length of signal sequence.

Remark. The convolution operation is probably one of the most famous in the field of machine learning. It has been used for image analysis and is now more and more used to process temporal data. It is in this context that we develop our analysis.

The figure 2 illustrates a convolution operation. We can see what the padding corresponds to. The dilation corresponds to the space between 2 inputs of the convolution. On the figure we have here a dilation of 1. The kernel size is the size of the convolution kernel. We can see that on this figure it is 3.

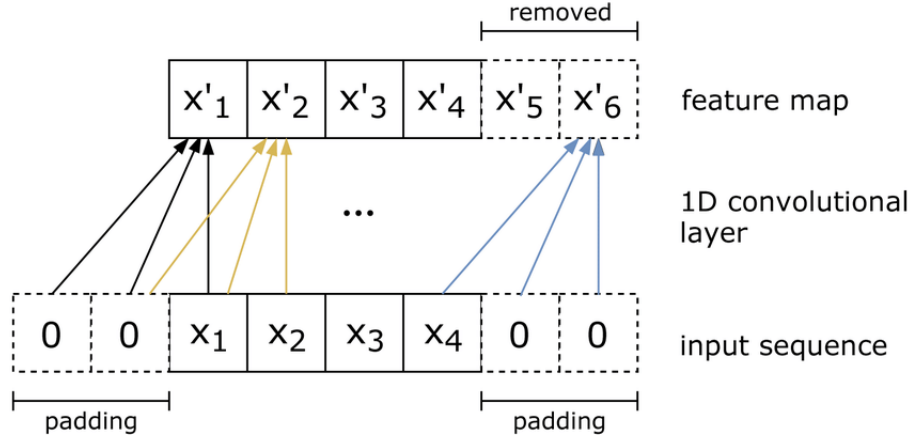


FIGURE 2 – Explanation of the Conv1D function, illustration

We talked about Channel in the definitions. It is an important concept because it is largely involved in the construction of our neural network. The figure 3 shows an operation with 1 channel in input and 1 channel in output. The figure 4 shows 3 channels in input and 1 channel in output. It is enough to understand intuitively this operation because it will be used in the following to build the neural network. In practice we have developed on Pytorch and all these elements are represented in "super-matrices" called tensors.

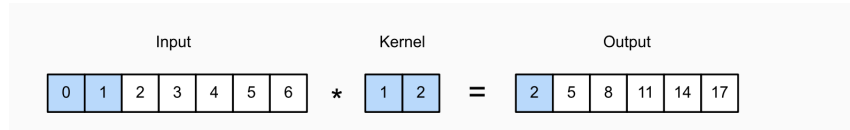


FIGURE 3 – Explanation of the Conv1D function, illustration

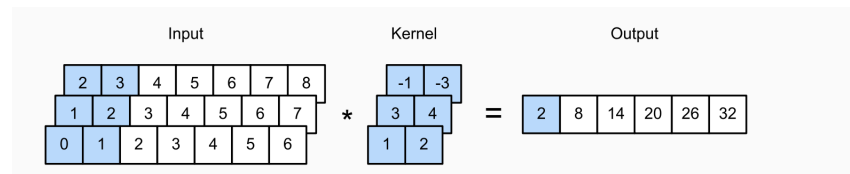


FIGURE 4 – Explanation of the Conv1D function, illustration

2.2 Vanishing and Exploding Gradient

One cannot do a complete work on neural networks without talking about the problem of the explosion or the decrease of the gradient. Learning, or calibrating neural networks, involves backpropagation of the error. The idea is to pass the gradient of the error layer by layer. However, when the network has several layers, as it will be in our case, the gradient can end up becoming very weak at the level of the last layers. These last layers do not learn anymore. Or on the contrary, the gradient can become much too important and we have an impossibility to do the learning. One solution

is to create identity connections between the different layers, as we have represented in figure 12. We speak here of residual blocks. The reader interested in these problems can find an excellent explanation in [Pascanu et al., 2013].

2.3 The 5 blocks of our neural network

We are actually going to use a Temporal Convolutional Network for the generator. The founding article is [Bai et al., 2018]. In order to understand the choice of our generator, we must go back to the properties of financial assets. Returns have a property of volatility **cluster**. When we look at a graph of returns, we can make clusters on it. The idea is that we observe zones of high volatility, where volatility is high and returns are highly variable, and zones of lower volatility, where on the contrary returns are lower. The idea is therefore to construct a function $f_{\mathcal{W}}^3$ that produces returns (r_1, \dots, r_n) that are not independent of each other. Another property is very important. We recall the form of $f_{\mathcal{W}}^3$:

$$f_{\mathcal{W}}^3 : \begin{cases} \mathbb{R}^n \longrightarrow \mathbb{R}^n \\ (\varepsilon_1, \dots, \varepsilon_n) \longrightarrow (r_1, \dots, r_n) \end{cases}$$

The yields must be F_t -measurable. In other words, the generator must only use $(\varepsilon_1, \dots, \varepsilon_i)$ to produce the returns (r_1, \dots, r_i) . This is the reason why we cannot use a feed-forward neural network as presented above ! Indeed, this one would use $(\varepsilon_1, \dots, \varepsilon_n)$ to produce the renderings (r_1, \dots, r_i) ! This is not realistic !

We have several candidates to successfully verify the temporal dependency condition. We could try RNNs (recurrent neural networks) for their ability to capture temporal dependencies. Then we switched to LSTMs, which are improvements of RNNs to overcome the Gradient Exploding/Vanishing problem. But since the arrival of TCNs [Bai et al., 2018], these two architectures have become less interesting because they are outperformed by TCNs. Moreover, it is possible to show that TCNs guarantee stationarity. We will not go into detail here on the very classical architectures of RNN and LSTM. There is a very large literature on this subject and it would make our TER much too long.

The very first block we use will allow us to pass the initial noise, which is actually 1 channel, to $h = 128$ channels after passing through Conv1D. Intuitively the idea is to proceed to a kind of "parallelization", by passing from layer to layer 128 channels, we can potentially find much faster how to improve the weights.

Below we give the table explaining the composition of the block, with the name of the layers used (see definitions above) as well as the dimensions of the inputs and outputs. Since we give a white Gaussian noise as input, it is natural that there is only one channel at the input, but there will be several at the output, we refer to the diagram 4.

NoiseToLatent	
Conv1d	$Channel_{in} = 1, Channel_{out} = h, kernel_size = 1, padding = 2, dilatation = 2$
LeakyReLU	<i>activation</i>
Conv1d	$Channel_{in} = h, Channel_{out} = h, kernel_size = 5, padding = 4, dilatation = 2$
LeakyReLU	<i>activation</i>

One of the first blocks we use to build our neural network is called BlockCNN and is based on the convolution operation we defined above.

Block CNN	
Conv1d	$Channel_{in} = h, Channel_{out} = h, kernel_size = 3, padding = 2, dilatation = 2$
LeakyReLU	<i>activation</i>

Then comes the Block Shift. In practice we work with Pytorch tensors. The Flatten instruction allows to resize the tensors. We are going to change the 128 channels we used to handle to 10 channels and we are going to use the "flatten" operation to "flatten" these channels and put them side by side. This very classical operation allows us to get back to the right dimension : latentDim, which is nothing else than the dimension of the white Gaussian noise that we pass in input to the generator.

Block Shift	
Conv1d	$Channel_{in} = h, Channel_{out} = 10, kernel_size = 3, padding = 2, dilatation = 2$
LeakyReLU	<i>activation</i>
Flatten	$start_dim = 1$
Linear	$input_taille = 10 * latent_dim, output_taille = 256$
LeakyReLU	<i>activation</i>

We also use a classic Block, which we call Block :

Block	
Linear	$input_{taille} = 256, output_{taille} = 256$
LeakyReLU	$activation$

Then the last block will be used to transform the output signal into a series of outputs.

LatentToOutput	
Linear	$input_{taille} = 256, output_{taille} = outputDimension$

The diagram 5 shows how these different blocks are connected to each other and how the input information passes through the neural network to produce a vector of output yields. Of course the dimensions of the outputs and inputs of the time series are consistent.

So we have described how $f_{\mathcal{W}}^3$ works. Now we have to find a way to make the gradient descent, that is to say to find $\omega \in \mathcal{W} / f_{\omega}^{original}(X) = f_{\omega}^3(X)$.

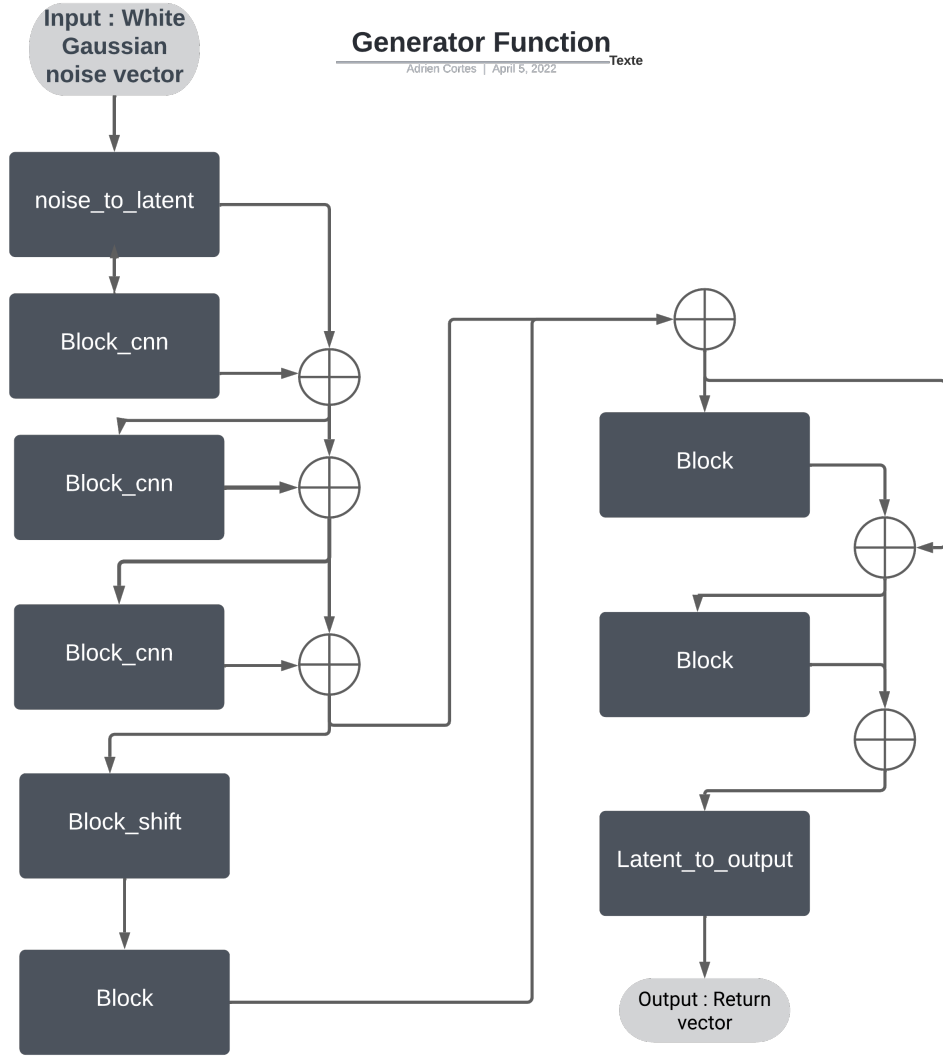


FIGURE 5 – Architecture of the generator we used

3 How to calibrate the function $f_{\mathcal{W}}^3$? Generative Adversial Networks

We have just seen the complex form of $f_{\mathcal{W}}^3$. The idea is now to know how to calibrate such a complex function. We will rely on a competition between two neural networks to achieve this learning.

3.1 Introduction to GAN

We are going to work on random data, so it is convenient for the following to introduce (Ω, F, \mathbb{P}) a probabilized space. We will focus on GANs or generative adversarial networks. GANs are relatively recent, we can find their origin in the paper [Goodfellow et al., 2014].

We will start by presenting how GANs work. GANs are composed of two connected neural networks, called respectively the *Discriminator* and the *Generator*. The generator will transform a random vector z of arbitrary size into a realization of the trajectory of a yield \hat{x} . In summary, the generator is a function $G : \mathbb{R}^n \rightarrow \mathbb{R}^p$ such that $G(z) = \hat{x}$.

Definition 3.1 (Random variable generated by a generator). Let G be a neural network with parameter space \mathcal{W}^g . Then the variable defined by :

$$\hat{x} : \Omega \times \mathcal{W}^g \rightarrow \mathbb{R}^p, (\omega, \theta) \mapsto G_\theta(z(\omega))$$

is called the random variable generated by the generator. G is called the generator. And z is a random variable on the probabilized space (Ω, F, \mathbb{P}) .

Definition 3.2 (Discriminator). Let \tilde{D} be a neural network with parameter space \mathcal{W}^d and $\sigma : \mathbb{R} \rightarrow [0, 1]^{\frac{1}{1}}$: $x \mapsto \frac{1}{1+e^{-x}}$ the sigmoid function. Then the function :

$$D : \mathbb{R}^p \times \mathcal{W}^d \rightarrow [0, 1], (x, \eta) \rightarrow \sigma \circ \tilde{D}_\eta(x)$$

is called a discriminator.

Remark. A classical discriminator is therefore a neural network as we have defined it, and where we put a sigmoid activation function at the end to be sure to have a number between 0 and 1, so a probability.

The discriminator is a function $D : \mathbb{R}^p \rightarrow [0, 1]$ which has an input x or \hat{x} gives the probability that the input is an authentic series. Thus, in the end of the discriminator training, we would like :

$$\begin{cases} D(x) = 1 \\ D(\hat{x}) = 0 \end{cases}$$

Or that :

$$\begin{cases} D(x) = 1 \\ D(G(z)) = 0 \end{cases} \quad (1)$$

We wish to train the generator to produce the most realistic \hat{x} output possible. The objective for the generator is therefore to eventually obtain :

$$D(G(z)) = 1 \quad (2)$$

The generator and the discriminator will each train to reach the objective given in the equations 1 and 2.

As in any neural network, we can use these equations to define a loss function that must be minimized. We note it $L(D, G)$. We can reformulate the problem as a min-max problem :

$$\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (3)$$

We train D to maximize the probability of assigning to $x \sim P_{\text{data}}$ the label 1 and to $x \sim P_g$ the label 0. We simply interpret that G is trained to minimize the probability that the returns from the generator are considered false. The equation 3 defines the loss function of GAN. To update the weights of the discriminator and the generator, we use the classical error backpropagation algorithm by computing the partial derivatives of the error with respect to the weights of the generator and the discriminator.

The figure 6 illustrates this concept.

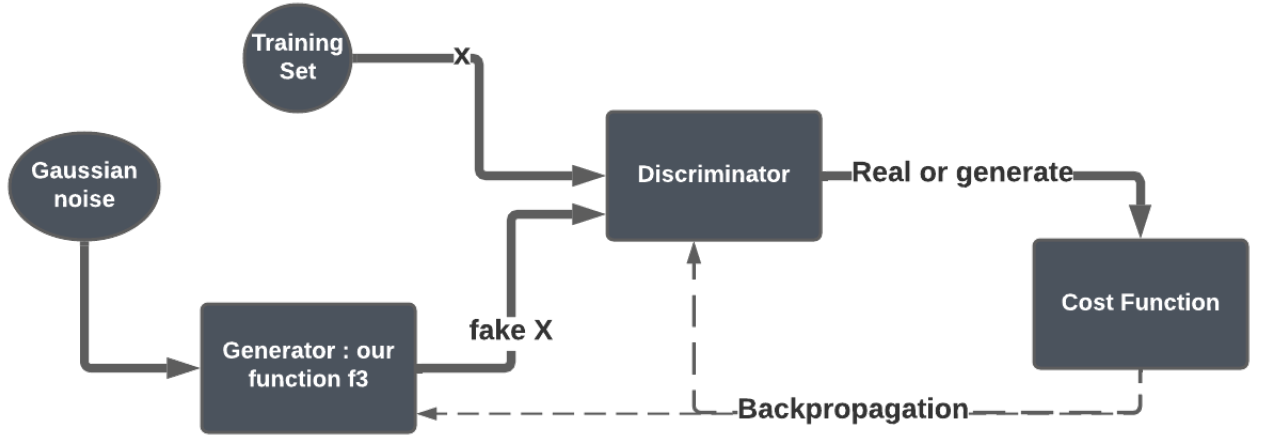


FIGURE 6 – We have created here an illustration of our GAN)

3.2 Wasserstein GAN

Artificial intelligence researchers at Facebook have come up with a very interesting paper [Arjovsky et al., 2017] that adapts GAN classify to time series like returns. This is very important for us, as we will be working with time series.

The idea is to modify the loss function of the GAN, $L(G, D)$, to fit the situation better than 3.

For this, we will rely on the Wasserstein distance. More information on this distance can be found in [Villani, 2009].

This distance is defined as :

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|] \quad (4)$$

Where $\Pi(P_r, P_g)$ corresponds to all the joint distributions $\gamma(x, y)$ whose marginal distributions are P_r and P_g respectively.

It is very interesting to note that we can rewrite the 4 as :

$$W(P_r, P_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{x \sim P_g} [f(x)] \quad (5)$$

The idea is that the discriminator approximates the function that maximizes the average distance between the samples of probability distribution P_r and P_g .

We can now present the Wasserstein GAN or WGAN (as presented in [Arjovsky et al., 2017]). We replace the cost function of the GAN $L(D, G)$ by the equation 5. Thus 3 becomes :

$$L(P_r, P_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{x \sim P_g} [f(x)]$$

Where the supremum is computed on all 1-lipchitzian functions. In the discriminator (concretely, we will replace the last activation function which was a sigmoid by a linear function, able to produce a scalar). We note \mathcal{W} the parameters of the discriminator D . We know that $D = f_w$, with $w \in \mathcal{W}$. The discriminator is trained to find the best weights $w \in \mathcal{W}$ that produce the function $D = f_w$ that maximizes the following equation :

$$L(P_{\text{data}}, P_g) = W(P_{\text{data}}, P_g) = \max_{w \in \mathcal{W}} \mathbb{E}_{x \sim P_{\text{data}}} [f_w(x)] - \mathbb{E}_{z \sim P_g(z)} [f_w(G_\theta(z))]$$

The problem here is that nothing forces f_w to be a 1-Lipchitzian function. But this is fundamental, because it allows us to make sure that the variations of the function are bounded. (otherwise it can be very difficult to get convergence).

Initially, the Facebook article [Arjovsky et al., 2017] proposes to force the weights $w \in \mathcal{W}$ to be such that $w \in [-c, c]$, with c fixed empirically. For various reasons, the article criticizes this technique as not efficient. Very recently, [Hogenboom, June 25, 2020] exposed a much more efficient method to force the function to be 1-Lipchitzian. And it is this method that we have chosen. It allows the WGAN-GP. The idea is to add a penalty to the cost function $L(P_{data}, P_g)$ to force it to be 1-lipchitzian. Indeed, [I. Gulrajani and Courville, 2017] uses the fact that a differentiable function is 1-lipchitzian if and only if its gradient has a norm at most equal to 1 almost everywhere.

This leads to the cost function that we will finally use :

$$L(P_r, P_g) = \mathbb{E}_{x \in P_r} [f_w(x)] - \mathbb{E}_{z \in P_g(z)} [f_w(G_\theta(z))] + \lambda \mathbb{E}_{x \in P_r} [(\|\nabla_x f_w(x)\|_2 - 1)^2] \quad (6)$$

We now have an efficient algorithm that summarizes the implementation of WGAN-GP that we used first to do our simulations. The algorithm of [Hogenboom, June 25, 2020] in the appendix 1 served as the basis for us to implement the WGAN-GP structure on Pytorch.

3.3 Technical details for learning

The risk in this calibration is to end up with a weight $\omega \in \mathcal{W}/f_\omega^3(X) = c$. With c a constant... This would simply mean always simulating the same trajectory. To prevent this, we bias the inputs and outputs of the generator by adding portions of real trajectories. This forces the generator to adapt and thus to produce a large variety of outputs.

We now propose to explain how we prepared the data, or did the "preprocessing" phase, which will allow the WGAN-GP to learn to simulate the assets. The diagram 7 explains how the data were prepared (the diagram is similar to that of many articles, notably [Magnus Wiese, 21 Dec 2019]).

Financial series are known to exhibit very heavy distribution tails in the returns series (normally applied for 1-step differentiation). The Lambert W transformation can be useful to generate wider distribution tails. We therefore used it, as described in this diagram.

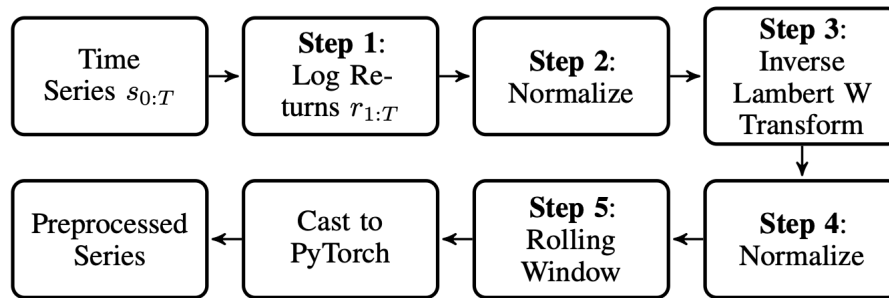


FIGURE 7 – Classic data preparation scheme

It now remains to train the neural network and to observe the results. We were able to train the generator to produce returns of size 240 from noises of size 100. The figure 8 allows us to see how fast the generator is learning. We see a slow convergence, but we did not take the risk of melting our computers to try to find the optimum. We let our computers run for 6 hours to get the result we will present. This is without counting the multiple attempts we made. Moreover, it is important to note that we trained 4 times more the discriminator than the generator. Otherwise there is no convergence possible, because the discriminator becomes too quickly fooled by the generator and cannot learn anymore.

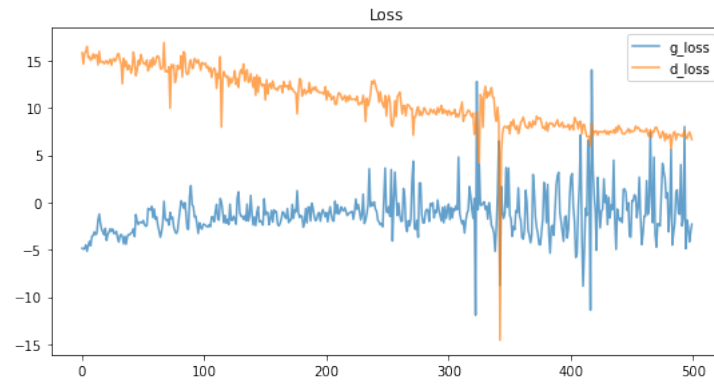


FIGURE 8 – Evolution of the generator and discriminator loss over the last 500 iterations

4 Use of Heston model

4.1 Reminders of the Black-Scholes model and extension to the Heston model

We begin with a brief review of the Black-Scholes model for pricing a European option, as well as its limitations and its extension to the Heston model.

Definition 4.1 (Perfect Market). In the following, we will call a perfect market a market where arbitrage is not possible, where short selling is possible, where there are no transaction costs, where there is a risk-free interest rate that is known in advance and constant, and where all underlyings are perfectly divisible

Definition 4.2 (Black-Scholes Model). A model in which the market is perfect and the price of the underlying asset S_t follows a geometric Brownian motion with constant volatility σ and constant drift μ . Moreover, in the case of a stock, it does not pay dividends between the time of the option's valuation and its expiration.

These assumptions allow us to demonstrate the Black-Scholes formula (i.e. the price of a European call as a function of different parameters) :

$$C(S_0, K, r, T, \sigma) = S_0 \mathcal{N}(d_1) - Ke^{-rT} \mathcal{N}(d_2)$$

with \mathcal{N} the distribution function of the normal distribution, $d_1 = \frac{1}{\sigma\sqrt{T}} \left[\ln\left(\frac{S_0}{K}\right) + \left(r + \frac{1}{2}\sigma^2\right)T \right]$ and $d_2 = d_1 - \sigma\sqrt{T}$

One limitation of this model is the fact that the volatility of the underlying asset is assumed to be constant over time, but we note that the implied volatility of the option depends on the strike of the option (we speak of a volatility smile) and also on the maturity of the option. This limitation has led researchers to develop models with non-constant but deterministic volatility, and then models with stochastic volatility (notably the Heston model).

Definition 4.3 (Heston Model). The Heston model is a stochastic volatility model, the assumptions of a perfect market are the same as in the Black-Scholes model, the difference is in the dynamics of the underlying where the volatility of the latter is stochastic and where the dynamics of this volatility are known. If we have $(\Omega, \mathcal{F}, \mathbb{P})$ a probability space with \mathbb{P} the historical probability, we have under this probability the dynamics of the underlying asset :

$$\begin{aligned} dS_t &= \mu S_t dt + \sqrt{v(t)} S_t dZ_1 \\ dv(t) &= \kappa(\theta - v(t))dt + \sigma\sqrt{v(t)}dZ_2 \\ \langle dZ_1, dZ_2 \rangle &= \rho dt \end{aligned}$$

where Z_1 and Z_2 are two Brownian motions under this probability.

Remark. We can see that there are two sources of risk (represented by the two Brownian motions) but only one risky asset in this market. We therefore conclude that the market is incomplete.

Definition 4.4 (Risk-Neutral Probability). Let (Ω, \mathcal{F}, P) be a probability space on which we define a price process S_t , we provide our space with the natural filtration of S_t . A risk-neutral probability \mathbb{Q} (or Equivalent Martingale Measure) is a probability measure equivalent to \mathbb{P} and such that the price process discounted at the risk-free rate r is a martingale under this probability.

Remark. We can show that if we place ourselves under a risk-neutral probability \mathbb{Q} (non-unique MME given that the market is incomplete) then the dynamics of the underlying asset becomes :

$$\begin{aligned} dS_t &= r S_t dt + \sqrt{v(t)} S_t d\tilde{Z}_1 \\ dv(t) &= \tilde{\kappa}(\tilde{\theta} - v(t))dt + \tilde{\sigma}\sqrt{v(t)}d\tilde{Z}_2 \\ \langle d\tilde{Z}_1, d\tilde{Z}_2 \rangle &= \tilde{\rho} dt \end{aligned}$$

where \tilde{Z}_1 and \tilde{Z}_2 are Brownian motions under \mathbb{Q} .

Remark. The interest of working in a risk-neutral world lies in the possible projection of our underlying asset. Indeed, even if under the historical probability it is possible to find some parameters from the derivatives (not using historical price data), it is much more difficult to find the parameter μ the return of the underlying asset since its calibration will strongly depend on the time scale used. Thus, to be able to project our asset (thanks to a discretization of the SDE) it is more interesting to find the parameters under \mathbb{Q} . However, there are very simple transformations to be applied to the model parameters to go from the real world to the risk-neutral world.

4.2 Calibration of the Heston model

Let (Ω, \mathcal{F}, P) be a probability space. We can define a price process S_t of an underlying on this probability space (P is thus the historical probability). We provide our probability space with F_t , the natural filtration of S_t (i.e. $F_t = \sigma(S_s, s < t)$). From the assumptions of the Heston model, it is possible under the measure \mathbb{P} to find a closed formula for the price of a European call with maturity T , and strike K . We will detail the formula and the method of obtaining it in the appendices.

Once this formula is established, the objective is to find the right model parameters and use them to model the trajectories of the underlying asset in question.

Remark. As explained above, it will be necessary to perform a transformation of these parameters to be able to exploit them in a risk neutral framework.

In order to calibrate the Heston model on a given underlying, we needed the model predictions and the observations to be as close as possible. But how to define this distance between the observations and the predictions of the model?

The first idea was to consider the classical Euclidean norm.

$$\sqrt{\sum_{i=1}^m [C_i - H(S, K_i, T_i, p, r)]^2}$$

where S is the current price of the underlying asset, r the risk-free rate (these two data are exogenous to the model), C_i is the price of the call with strike K_i , of maturity T_i , and $H(S, K, T, p, r)$ is the price of a call of strike K , maturity T of an underlying which is S in the Heston model where the parameters are in the vector p and where the risk-free rate is equal to r .

However, one quickly realizes that this measure gives the same weighting to all options. However, it can be observed that for options whose strike price is very far from the current price, the exchange prices of these options can deviate from the models because they can be the object of speculative behavior or, on the contrary, be the object of a very low demand and a lack of liquidity of these options.

We therefore need to find a measure of distance between observed and predicted prices that takes into account the distance between the strike price and the current price of the underlying and that gives more weight to options whose strike price is close to the current price of the underlying.

For this we define the following matrix M :

$$M = \begin{pmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_m \end{pmatrix}$$

where $\omega_i = \exp(-a|(S/K_i) - 1|)$. We notice that the matrix M is symmetric positive definite to which we can associate a scalar product. We can therefore define the norm associated with this scalar product and thus a distance.

Remark. According to the work of François M. Quitard-Pinon and Rivo Randrianarivony [Quitard-Pinon and Randrianarivony, 2008], we decide to fix $a=10$. An interesting study would be to look at the impact of this parameter on the calibration of the model.

Finally, we are interested in the following quantity :

$$\phi(p) = \sum_{i=1}^m \omega_i (H(S, K_i, T_i, p, r) - C_i)^2$$

Remark. For the sake of simplicity, we have removed the square root because it keeps the order anyway.

Once this measure is well defined, we need to find the vector of parameters that minimizes this cost function. So we are looking for $p^* = \arg \min_{p \in U} \phi(p)$.

For this we perform a projected gradient descent for which we will detail the algorithm and convergence results in appendices.

4.2.1 Calibration results

We decided to calibrate our model on Apple. After 200 iterations of our minimization algorithm, we have our vector of parameters. From these parameters, we can plot the observed and predicted price surfaces.

To judge the quality of our calibration, we are interested in the root of the weighted mean squared error defined by :

$$\text{RWMSE} = \sqrt{\sum_{i=1}^m \pi_i [C_i - H(S, K_i, T_i, p, r)]^2}$$

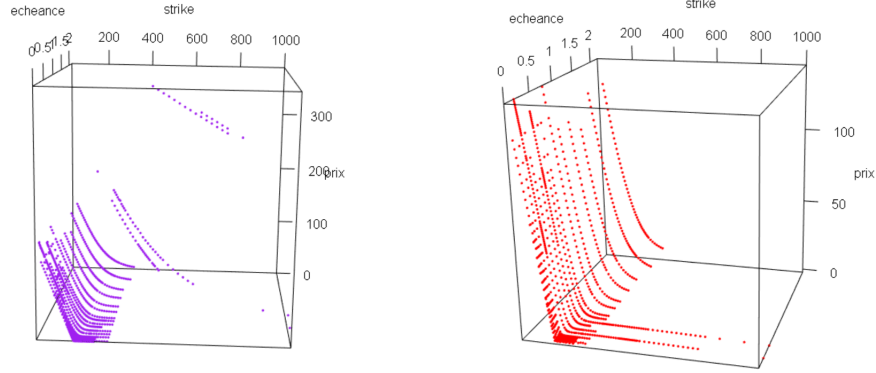


FIGURE 9 – Observed price surface and Price surface given by Heston model after calibration, illustration

In our calibration on Apple, we obtain an RWMSE of 8.71. This coefficient is rather satisfactory but could be improved.

Remark. The results we obtained were obtained by setting the parameter $a = 10$ in the coefficient ω_i . For the same parameters and choosing $a = 20$ we find a RWMSE of 2.25 which is much better. This comes from the fact that we give even less weight to options whose strike is far from the current price. However, we can see on the observed price surface that the prices are very high and far from the pricing models and that their values inflate the RWMSE if we put an insufficiently low weight on these options.

4.3 Model a price path of the underlying asset using the parameters of the Heston model

4.3.1 Transition from the real world to the risk-neutral world

Definition 4.5. Under the historical probability \mathbb{P} , the dynamics of S_t is as follows :

$$\begin{aligned} dS_t &= \mu S_t dt + \sqrt{v(t)} S_t dZ_1 \\ dv(t) &= \kappa(\theta - v(t))dt + \sigma \sqrt{v(t)} dZ_2 \\ \langle dZ_1, dZ_2 \rangle &= \rho dt \end{aligned}$$

where Z_1 and Z_2 are Brownian motions under \mathbb{P} .

Definition 4.6. Under the MME \mathbb{Q} , the dynamics of S_t is as follows :

$$\begin{aligned} dS_t &= r S_t dt + \sqrt{v(t)} S_t d\tilde{Z}_1 \\ dv(t) &= \tilde{\kappa}(\tilde{\theta} - v(t))dt + \tilde{\sigma} \sqrt{v(t)} d\tilde{Z}_2 \\ \langle d\tilde{Z}_1, d\tilde{Z}_2 \rangle &= \tilde{\rho} dt \end{aligned}$$

where \tilde{Z}_1 and \tilde{Z}_2 are Brownian motions under \mathbb{Q} .

Remark. When calibrating the Heston model in a real world, we need to find an additional parameter (which does not appear in the SDE) : the market price for volatility risk. This parameter will be noted λ . This parameter is used in the price formulas of a European option and reflects the price that the market gives to volatility (like the risk premium).

Remark. From the vector $p^* = (\rho, \kappa, \theta, v_0, \sigma, \lambda)$ optimal that we obtain in the real world, it is easy to find our vector $p_Q^* = (\rho, \tilde{\kappa}, \tilde{\theta}, \tilde{v}_0, \tilde{\sigma})$ (we notice that l does not appear anymore). Indeed, according to the demonstration in appendices we have :

$$\begin{aligned}\tilde{\kappa} &= \kappa + \lambda \\ \tilde{\theta} &= \kappa\theta/(\kappa + \lambda)\end{aligned}$$

All other parameters remain unchanged. For more readability in the rest of the work, we will only consider the parameters in a risk-neutral world and we will remove the *from the parameters*.

4.3.2 Choice of the function f^4

As a reminder, our objective was to simulate "plausible" trajectories of an asset. This generator of trajectories in a neutral-risk world is a very useful tool to compute ESG and find prices of derivatives. For this we started from the idea that we had to find a continuous function that distorts a random noise into a discretized trajectory of the asset. In previous approaches (Bachelier, Black-Scholes, GAN), we had

$$f : \begin{cases} \mathbb{R}^n \longrightarrow \mathbb{R}^n \\ (\varepsilon_1, \dots, \varepsilon_n) \longrightarrow (S_1, \dots, S_n) \end{cases} \cdot$$

with a sequence of independent and identically distributed variables $\varepsilon_i \sim N(0, 1)$.

Here, the objective will also be to define a function from the parameters of the Heston model that deforms a random noise into a tract of the asset. However, we will have to take care to modify the starting space. Indeed, contrary to the Black-Scholes model, there are here two sources of correlated risks, so we must look for

$$f : \begin{cases} \mathbb{R}^n \times \mathbb{R}^{n-1} \times \mathbb{R}^+ \times \mathbb{R}^+ \longrightarrow \mathbb{R}^n \\ [(\varepsilon_1, \dots, \varepsilon_n), (u_1, \dots, u_{n-1}), S_0, v_0] \longrightarrow (S_1, \dots, S_n) \end{cases} \cdot$$

with $\varepsilon_i \sim N(0, 1)$ a sequence of independent and identically distributed variables, and $u_i \sim N(0, 1)$ a sequence of independent and identically distributed variables such that $\forall i, j, \varepsilon_i$ and u_j are independent if $i \neq j$ and if $i = j$ then $\text{Corr}(u_i, \varepsilon_i) = \rho$

For this, we use the two SDE that we discretize.

Remark. In the Heston model, the variance process v_t is a CIR model and is $\mathbb{P} - p - p$ positive or zero. This property is no longer true if we do not work in continuous time and if we discretize the trajectory. To keep the positive character of the variance at each iteration, we will keep the positive part of v_t .

So, we can write

$$\begin{aligned}S_1 &= (1+r)S_0 + \sqrt{v_0}S_0\varepsilon_1 \\ v_1 &= (\kappa(\theta - v_0) + \sigma\sqrt{v_0}u_1)_+ \\ S_2 &= (1+r)S_1 + \sqrt{v_1}S_1\varepsilon_2 = S_0((1+r)^2 + (1+r)\sqrt{v_0}\varepsilon_1 + (1+r)\sqrt{(\kappa(\theta - v_0) + \sigma\sqrt{v_0}u_1)_+}\varepsilon_2 + \sqrt{(\kappa(\theta - v_0) + \sigma\sqrt{v_0}u_1)_+}\sqrt{v_0}\varepsilon_1\varepsilon_2) \\ &\dots\end{aligned}$$

Finally, we have defined a function f^4 that generates a "plausible" trajectory of an asset from two correlated noises.

5 Results analysis

For our study we decided to observe Apple's price over the period 03/01/2012 - 01/03/2022 and to verify if our trained neural network captures Apple's price characteristics. To compare the results, we will apply the Heston model calibrated on the Apple stock. Throughout our analysis we will talk about the Apple close price and log returns or continuously compounded returns at time t , denoted as r_t , are defined as :

$$r_t = \frac{P_t}{P_{t-1}}$$

where P_t and P_{t-1} are the closing prices of current and previous date respectively

Remark. This comparison with the Heston model has its limitations. Indeed, the calibration of the Heston model is not based on historical data but only on the derivatives at the date of the calibration. It is very delicate to compare retrospectively the predictions of the Heston model with historical data. Indeed, in our study our approach was to project an asset in the future under a risk-neutral probability of our asset (it is a very useful tool to compute prices by Monte-Carlo for example). There is no guarantee that we would have had the same coefficients if we had done this calibration 10 years ago for example.



FIGURE 10 – Apple price 03/01/2012 - 01/03/2022

We begin by studying the skewness and kurtosis coefficients on Apple's returns from the GAN and the Heston model. We simulated over 5,000 paths with the Heston model and GAN over periods of 226 market days, which corresponds to approximately one year on the financial market.

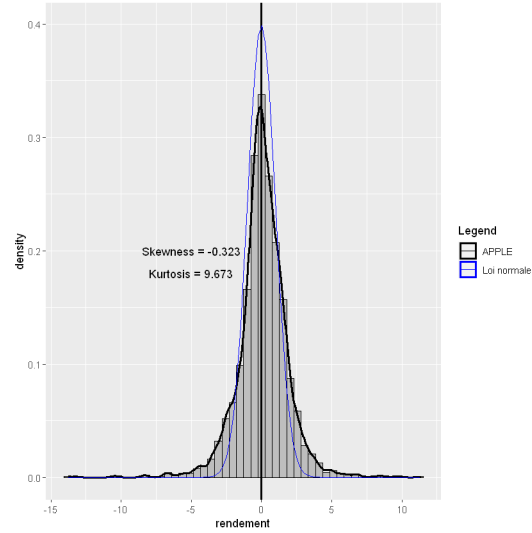


FIGURE 11 – Apple log returns density

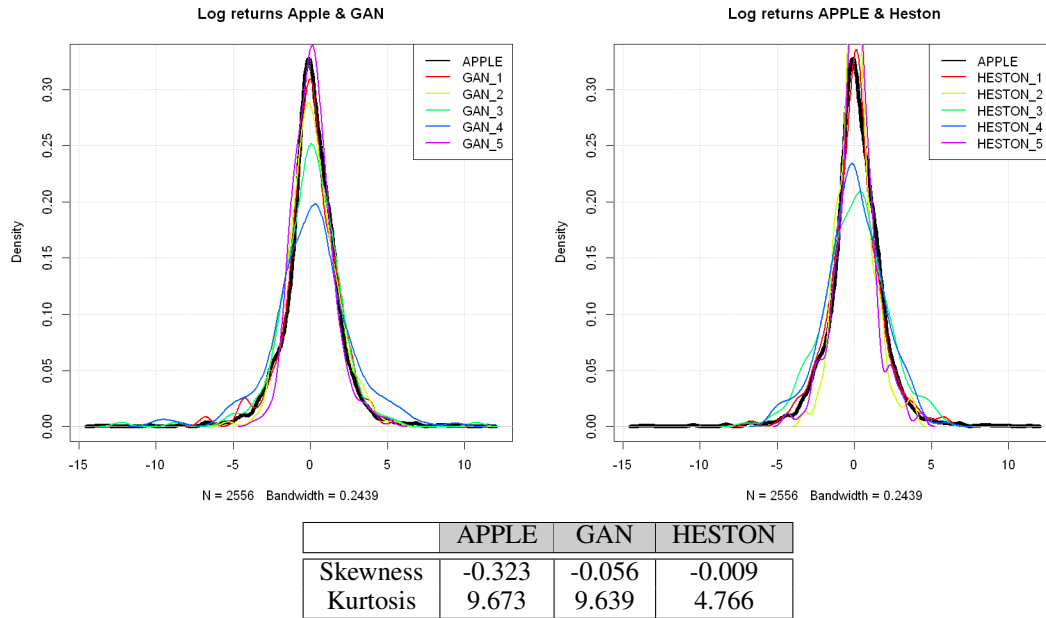


FIGURE 12 – log returns density of GAN and Heston on 5 fictitious trajectories. In black the density of the Apple log returns

We notice that the GAN kurtosis coefficient is very similar to Apple compared to the Heston model. The GAN skewness coefficient is 7 times smaller than Apple's, so there is still room for improvement. The important point to note is that the GAN log-returns do not follow a normal distribution as indicated in the Black-Scholes model. GAN therefore proposes a more realistic approach to log returns.

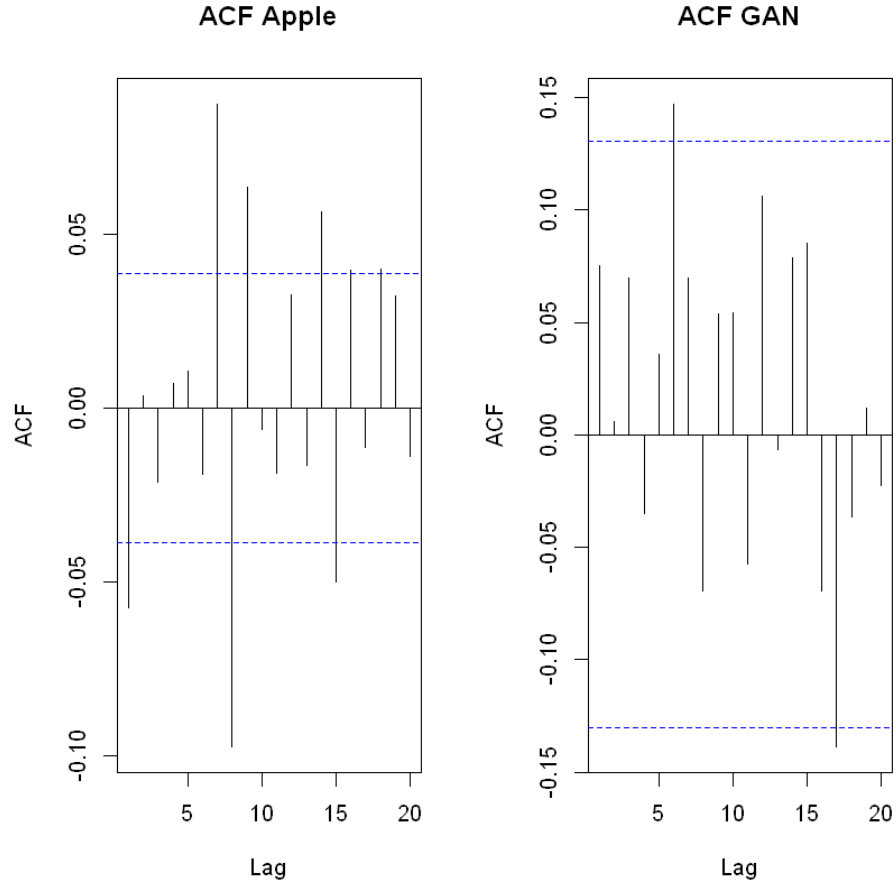


FIGURE 13 – Apple-GAN correlogram

Then we studied the behavior of the returns from the GAN using a correlogram. The purpose of the correlogram is to measure the autocorrelation of a phenomenon at a time t with measurements at previous times. We compared the ACF (Autocorrelation Function) of several GAN trajectories. We observe nothing disturbing on the two outputs. There is no trend.

For the following analysis, we simulated about ten trajectories using GAN on the same historical Apple starting price. We repeated the experiment for each year over the period 2010 - 2021. For each year we computed σ the historical volatility and μ the average growth rate of the asset price following the Black-Scholes model for both the GAN and Apple returns.

We recall that according to the *Black – Scholes* model log returns follow a normal distribution :

$$\mathcal{N}(\mu - \frac{1}{2}\sigma^2, \sigma^2)$$

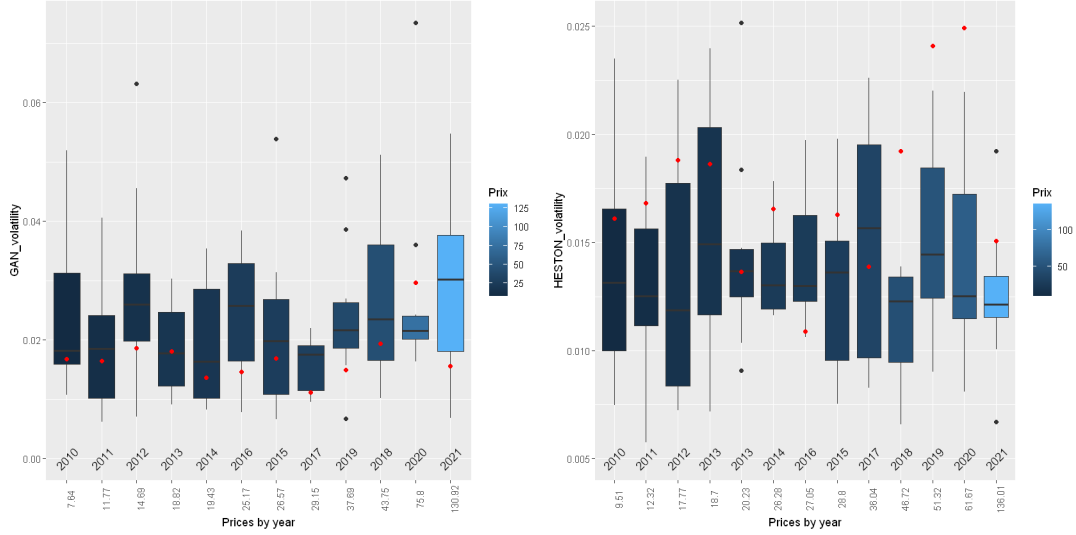


FIGURE 14 – Boxplot volatility σ as a function of the historical Apple price

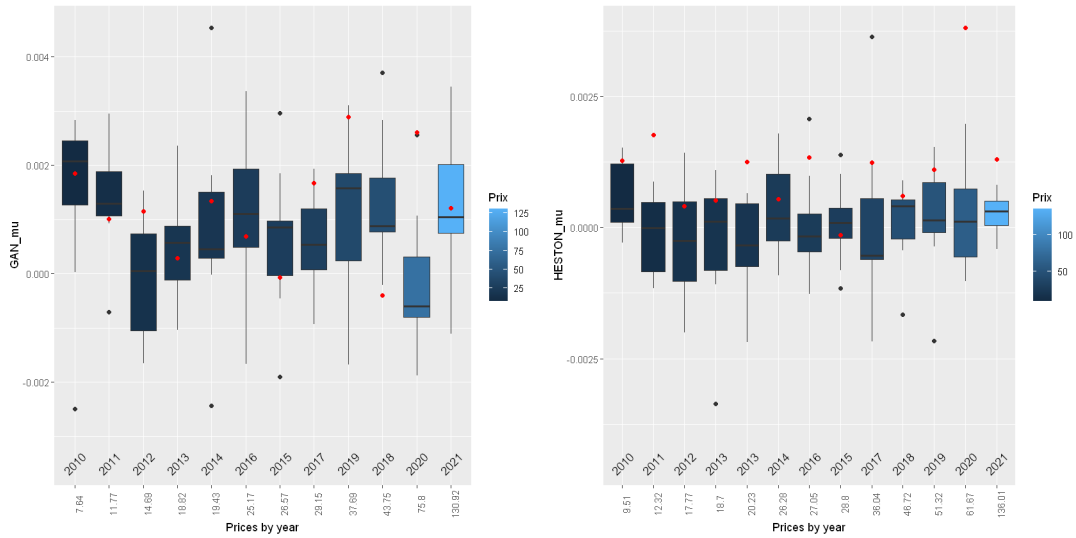


FIGURE 15 – Boxplot of the average growth rate of the asset price μ against the historical Apple price

On the graphs above, the red points correspond to Apple's values. The points outside the moustache box can be interpreted as points of vigilance or crisis points. Overall, the observations are asymmetrical.

We notice on the figure 14 that the GAN volatility of the Black-Scholes model includes the volatility of Apple. On average the neural network has assimilated the volatility well, with min and max that explode at times. This can be interpreted as periods of crisis where returns vary very strongly.

We have quite similar results on the figure 15. We notice extreme points (crisis points or vigilance points) outside the boxplot and sometimes even very close to an Apple point for a starting price displayed at 61.67. This is the period 2020-2021, the period of the health crisis. It is therefore understandable to find this Apple point very far from the box (expected values). What attracts our attention is the information provided by GAN on this period, because it indicates a point of vigilance very close to the real value on this period.

In this part, we propose to check if the GAN training has well remembered the 2007 - 2008 crisis.

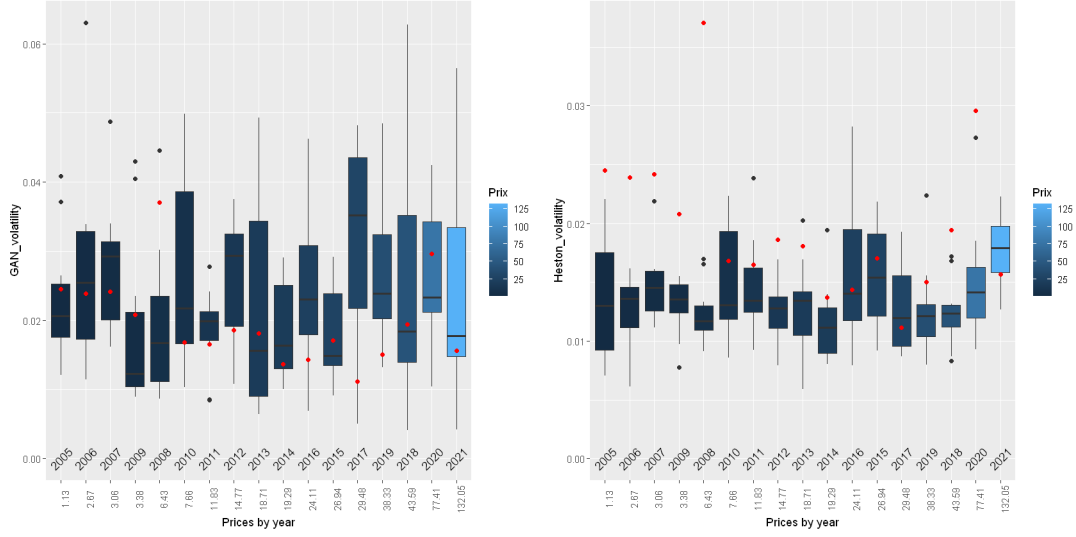


FIGURE 16 – Boxplot volatility σ as a function of the historical Apple price

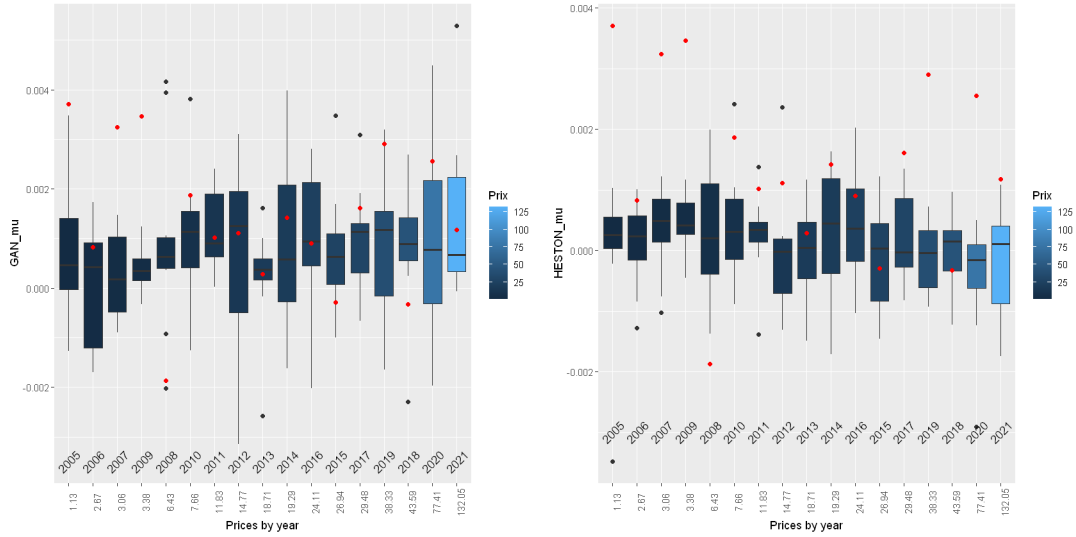


FIGURE 17 – Boxplot of the average growth rate of the asset price μ against the historical Apple price

We can see that Apple's observed values in 2008 are well outside the boxes for GAN and GAN proposes better results for σ and μ . Thus, the learning is both complete and better performing and more realistic than Heston's model.

To finish, here are some projections of the neural network and the Heston model on 227 days of market with as starting price the Apple price of 18/02/20 (79.75).

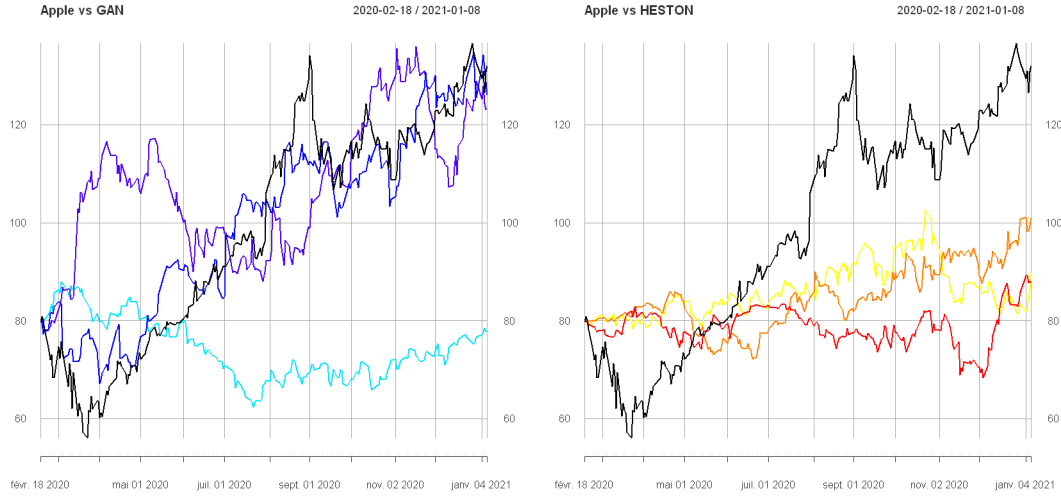


FIGURE 18 – Apple GAN and Heston price projections for 18/02/20 - 08/01/2021 period. In black the price of Apple

On this last figure 18 we notice that the neural network is very well calibrated on the Apple price. As a reminder, the neural network has been trained on the period 1980 - 2017. We also notice that despite the surprising results, the network sometimes proposes fewer fine values and requires more training in order to capture the behavior of Apple's share price as accurately as possible.

Remark. As explained above, this comparison with the Heston model is very delicate. Indeed, to project trajectories with parameters of Heston model, we work in a risk-neutral world. Each individual trajectory (like the one in the graph) is a possible realization of the price process in the real world. The difference with the real world is that in this risk-neutral world bad scenarios are more likely to occur [Hatfield and Neutral, 2009]. These trajectories are intended to be used to make Monte-Carlo. A good way to compare the neural network with the Heston model would be to project from the current price over a given period (1 year for example) with both methods and compare the results (compare predictions on the price of derivatives or statistical measures on returns). Moreover, it is important to note that by construction of our asset projection function, the expected return on our asset is the risk-free rate, so it is not wise to compare this parameter for both methods. However, it would be much more interesting to compare the volatilities.

6 Conclusion

To conclude, GAN has been able to assimilate the behavior and characteristics of the Apple price by the Heston model and provided a predictive tool for the volatility and average rate of return of the GAN is a promising simulator of Apple's price, and we can imagine much better results with a significant learning curve. The GAN is a promising simulator of Apple's price, we can imagine much better results with a consequent learning time. Indeed, for each training session we observed an important progression on the adjustment of the simulator while keeping the DNA of the Apple course. However, with more than 800,000 parameters, GAN is like a black box which makes it very complex to handle.

Références

- Louis Bachelier. Théorie de la spéculation. In *Annales scientifiques de l'École normale supérieure*, volume 17, pages 21–86, 1900.
- Nicole El Karoui, Monique Jeanblanc-Picquè, and Steven E Shreve. Robustness of the black and scholes formula. *Mathematical finance*, 8(2) :93–126, 1998.
- NM Mishachev. Backpropagation in matrix notation. *arXiv preprint arXiv :1707.02746*, 2017.
- Terence Parr and Jeremy Howard. The matrix calculus you need for deep learning. *arXiv preprint arXiv :1802.01528*, 2018.
- Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2) :251–257, 1991.
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.
- Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv :1803.01271*, 2018.
- Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.
- Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.
- Cédric Villani. The wasserstein distances. In *Optimal transport*, pages 93–111. Springer, 2009.
- Casper Hogenboom. Generation of synthetic financial time-series with generative adversarial networks. June 25, 2020.
- M. Arjovsky V. Dumoulin I. Gulrajani, F. Ahmed and A. Courville. Improved training of wasserstein gans montreal institute for learning algorithms. 2017.
- Ralf Korn Peter Kretschmer Magnus Wiese, Robert Knobloch. Quant gans : Deep generation of financial time series. 21 Dec 2019.
- François M Quittard-Pinon and Rivo Randrianarivony. Calibrage d'options pour trois modèles mixtes diffusions et sauts. *Finance*, 29(2) :103–130, 2008.
- Gary Hatfield and A Note Regarding Risk Neutral. Real world scenarios-dispelling a common misperception. *Product Matters*, 73, 2009.
- Eric Jeangirard Khaled Saleh Gilles Blanchet, Moncef Elacheche. Modèle d'heston, pricing d'options européennes et calibration. 2007.
- Florian Adjedj. Construction d'un modèle de profitabilité et tarification d'un contrat en unités de compte avec garantie plancher. 2015.
- Philippe G Ciarlet, Philippe Gaston Ciarlet, Bernadette Miara, and Jean-Marie Thomas. *Introduction to numerical linear algebra and optimisation*. Cambridge University Press, 1989.
- Marcos Lopez De Prado. *Advances in financial machine learning*. John Wiley & Sons, 2018.

A Appendices on Heston model

A.1 Obtaining the formula for the price of a call in the Heston model

We present here a proof of the pricing of a European call in the Heston model (the pricing of a put is obtained thanks to the call-put parity relation). Proof and notation elements have been taken from the work of Gilles Blanchet and his team [Gilles Blanchet, 2007]

Let (Ω, \mathcal{F}, P) be a probability space. We can define a price process S_t of an underlying on this probability space (P is thus the historical probability). We provide our probabilized space with F_t , the natural filtration of S_t (i.e. $F_t = \sigma(S_s, s < t)$). We keep the assumptions of AOA and perfect market. We also assume that under the probability \mathbb{P} we have

$$\begin{aligned} dS_t &= \mu S_t dt + \sqrt{v(t)} S_t dZ_1 \\ dv(t) &= \kappa(\theta - v(t))dt + \sigma \sqrt{v(t)} dZ_2 \\ \langle dZ_1, dZ_2 \rangle &= \rho dt \end{aligned}$$

Consider a European call with maturity T and strike price K . Let us note $C(S, v, t)$ its value at time t , where S is the value of the underlying at this same time, and v is the value of the variance at this same time.

In order to value this option, our objective will be to determine its hedge portfolio (if it exists) and to determine its price.

Remark. Since the market is incomplete (because there are two sources of risk), we cannot replicate the option payoffs with just the underlying risky asset and the risk-free asset. We need another asset of value V_1 that depends on the volatility (we cover all sources of risk).

Let $\Pi = C - \Delta S - \Delta_1 V_1$ be the value of the option's hedge portfolio (risk-free portfolio).

According to Itô's formula, we have

$$d\Pi = \left\{ \frac{\partial C}{\partial t} + \frac{1}{2}vS^2(t)\frac{\partial^2 C}{\partial S^2} + \rho\sigma vS(t)\frac{\partial^2 C}{\partial v\partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 C}{\partial v^2} \right\} dt - \Delta_1 \left\{ \frac{\partial V_1}{\partial t} + \frac{1}{2}vS^2(t)\frac{\partial^2 V_1}{\partial S^2} + \rho\sigma vS(t)\frac{\partial^2 V_1}{\partial v\partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V_1}{\partial v^2} \right\} dt + \left\{ \frac{\partial V}{\partial S} - \Delta_1 \frac{\partial V_1}{\partial S} - \Delta \right\} dS + \left\{ \frac{\partial C}{\partial v} - \Delta_1 \frac{\partial V_1}{\partial v} \right\} dv$$

Now we construct this portfolio in such a way that it is risk-free, so it is necessary that the terms in dS and in dv (the terms that cause the value of the portfolio to vary because of variations in the value of S and the value of v) are zero. Thus we have these relations :

$$\begin{cases} \Delta = \frac{\partial C}{\partial S} - \frac{\partial C/\partial v}{\partial V_1/\partial v} \frac{\partial V_1}{\partial S} \\ \Delta_1 = \frac{\partial C/\partial v}{\partial V_1/\partial v} \end{cases}$$

and we also have that

$$d\Pi = \left\{ \frac{\partial C}{\partial t} + \frac{1}{2}vS^2(t)\frac{\partial^2 C}{\partial S^2} + \rho\sigma vS(t)\frac{\partial^2 C}{\partial v\partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 C}{\partial v^2} \right\} dt - \Delta_1 \left\{ \frac{\partial V_1}{\partial t} + \frac{1}{2}vS^2(t)\frac{\partial^2 V_1}{\partial S^2} + \rho\sigma vS(t)\frac{\partial^2 V_1}{\partial v\partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V_1}{\partial v^2} \right\} dt$$

Since this portfolio is risk-free and thanks to the AOA hypothesis, we can say that the return on this portfolio is the risk-free rate r . So we have $d\Pi = r\Pi dt = r(C - \Delta S - \Delta_1 V_1)dt$

By replacing Δ and Δ_1 we have

$$\frac{1}{\partial C/\partial v} \left\{ \frac{\partial C}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 C}{\partial S^2} + \rho\sigma vS\frac{\partial^2 C}{\partial v\partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 C}{\partial v^2} + rS\frac{\partial C}{\partial S} - rC \right\} = \frac{1}{\partial V_1/\partial v} \left\{ \frac{\partial V_1}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 V_1}{\partial S^2} + \rho\sigma vS\frac{\partial^2 V_1}{\partial v\partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V_1}{\partial v^2} + rS\frac{\partial V_1}{\partial S} - rV_1 \right\}$$

The left term depends only on C while the right term depends only on V_1 . We can therefore deduce that these two terms are equal to the same quantity $f(S, v, t)$ which depends only on the parameters S, v and t .

So we have

$$\frac{\partial C}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 C}{\partial S^2} + \rho\sigma vS\frac{\partial^2 C}{\partial v\partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 C}{\partial v^2} + rS\frac{\partial C}{\partial S} - rC = \frac{\partial C}{\partial v} f \quad (7)$$

which can be rewritten as

$$\frac{\partial V}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma vS\frac{\partial^2 V}{\partial v\partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} + rS\frac{\partial V}{\partial S} - rV = \frac{\partial V}{\partial v} [\kappa(\theta - v) - \Lambda(S, v, t)\sigma\sqrt{v}]$$

where $\Lambda(S, v, t)$ is the market price of volatility.

We can put $\Lambda(S, v, t) = \frac{\lambda\sqrt{v}}{\sigma}$ so that we have

$$\frac{\partial V}{\partial t} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma vS\frac{\partial^2 V}{\partial v\partial S} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} + rS\frac{\partial V}{\partial S} - rV - \frac{\partial V}{\partial v} [\kappa(\theta - v) - \lambda v] = 0 \quad (8)$$

We are looking for $C(S, v, t)$ of the form

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

We pose $x = \ln(S)$ and $\tau = T - t$.

By injecting (9) into (8) we obtain $\forall j = 1, 2$

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

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

Moreover, since $C(S, v, t)$ is the value of the call at time t , we have $C(S, v, T) = (S_T - K)_+$. We can therefore derive the following terminal conditions :

$$P_j(x, v, T) = \mathbb{1}_{x \ln(K)}$$

Now the P_j can be interpreted as probabilities, so the characteristic functions of P_1 and P_2 (f_1 and f_2) are solutions of the PDE :

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

with the following terminal condition

$$f_j(x, v, T, z) = e^{izx}$$

We look for a solution in the form :

$$f_j(x, v, t, z) = \exp(C(\tau, z) + D(\tau, z)v + izx)$$

Injecting into (10), we obtain :

$$\begin{aligned} -\frac{z^2}{2} + \rho\sigma ziD + \frac{1}{2}\sigma^2 D^2 + u_j zi - b_j D + \frac{\partial D}{\partial t} &= 0 \\ rzi + aD + \frac{\partial C}{\partial t} &= 0 \end{aligned}$$

with $C(0, z) = D(0, z) = 0$.

We can solve theses ODE :

$$C(\tau, z) = rzi\tau + \frac{a}{\sigma^2} \left[(b_j - \rho\sigma zi + d)\tau - 2 \ln \left(\frac{1 - ge^{d\tau}}{1 - g} \right) \right]$$

$$D(\tau, z) = \frac{b_j - \rho\sigma zi + d}{\sigma^2} \left[\frac{1 - e^{d\tau}}{1 - ge^{d\tau}} \right]$$

with $g = \frac{b_j - \rho\sigma zi + d}{b_j - \rho\sigma zi - d}$ and $d = \sqrt{(\rho\sigma zi - b_j)^2 - \sigma^2(2u_j zi - z^2)}$

By inverse Fourier transform we obtain :

$$P_j = \frac{1}{2} + \frac{1}{\pi} \int_0^{+\infty} \text{Re} \left[\frac{e^{-iz \ln(K)} f_j}{iz} \right] dz$$

A.2 Transition from the real world to the risk-neutral world

We present here a proof of the transformations to be applied to the parameters of the Heston model in the real world to obtain those in the risk-neutral world. Some elements of proof and notation have been taken from Florian Adjedj's work [Adjedj, 2015].

Consider a delta hedging portfolio $\Pi_1(t) = C(S, v, t) - \Delta_t S_t$.

According to Ito's formula :

$$d\Pi_1(t) = dC - \Delta_t dS_t \quad (11)$$

$$= \left[\frac{\partial C}{\partial t} + \mu S \frac{\partial C}{\partial S} + \frac{1}{2} v S^2 \frac{\partial^2 C}{\partial S^2} + \kappa[\theta - v] \frac{\partial C}{\partial v} + \rho \sigma v S \frac{\partial^2 C}{\partial v \partial S} + \frac{1}{2} \sigma^2 v \frac{\partial^2 C}{\partial v^2} \right] dt \quad (12)$$

$$+ \left(\frac{\partial C}{\partial S} - \Delta \right) S \sqrt{v} dZ_1 + \frac{\partial C}{\partial v} \sigma \sqrt{v} dZ_2 - \mu \Delta S dt \quad (13)$$

Since this is a delta hedging portfolio, we can write : $\frac{\partial C}{\partial S} - \Delta = 0$

Then

$$d\Pi_1(t) = \left[\frac{\partial C}{\partial t} + \frac{1}{2} v S^2 \frac{\partial^2 C}{\partial S^2} + \kappa[\theta - v] \frac{\partial C}{\partial v} + \rho \sigma v S \frac{\partial^2 C}{\partial v \partial S} + \frac{1}{2} \sigma^2 v \frac{\partial^2 C}{\partial v^2} \right] dt + \left(\frac{\partial C}{\partial v} - \Delta \right) S \sqrt{v} dZ_1 + \frac{\partial C}{\partial v} \sigma \sqrt{v} dZ_2$$

So for the deflated portfolio :

$$\begin{aligned} d\Pi_1 - r\Pi_1 dt &= dC - \Delta dS - rC dt + r\Delta S dt \\ &= \left[\frac{\partial C}{\partial t} + \frac{1}{2} v S^2 \frac{\partial^2 C}{\partial S^2} + \kappa[\theta - v] \frac{\partial C}{\partial v} + \frac{1}{2} \sigma^2 v \frac{\partial^2 C}{\partial v^2} + \rho \sigma v S \frac{\partial^2 C}{\partial v \partial S} + S r \frac{\partial C}{\partial S} - rC \right] dt \\ &\quad + \sigma \sqrt{v} \frac{\partial C}{\partial v} dZ_2 \\ &= \frac{\partial C}{\partial v} (\kappa[\theta - v] dt + f dt + \sigma \sqrt{v} dZ_2) \\ &= \frac{\partial C}{\partial v} \sigma \sqrt{v} d\tilde{Z}_2 \end{aligned}$$

With

$$d\tilde{Z}_2 = dZ_2 + \Lambda dt \quad (14)$$

where $\Lambda = \frac{\kappa[\theta - v] + f}{\sigma \sqrt{v}}$, \tilde{Z}_2 is a Brownian motion under \mathbb{Q} and Λ is the market price of the volatility risk and f is the function obtained in (7).

Now, we try to build a vega-neutral portfolio Π_2 (immune to a variation in the volatility of the underlying).

$$\Pi_2 = C(S, v, t) - \Gamma_t V_1(v, t)$$

Indeed, as a reminder, V_1 is an asset that depends on the volatility of S and allows to hedge the volatility risk.

Now as Π_2 is vega neutral, we have : $\frac{\partial \Pi_2}{\partial v} = \frac{\partial C}{\partial v} - \Gamma_t \frac{\partial V_1}{\partial v} = 0$

$$\text{So } d\Pi_2 - r\Pi_2 dt = \left(\frac{\partial V}{\partial S} - \Gamma \frac{\partial V_1}{\partial S} \right) S [(\mu - r)dt + \sqrt{v} dZ_1] = \left(\frac{\partial V}{\partial S} - \Gamma \frac{\partial V_1}{\partial S} \right) S \sqrt{v} d\tilde{Z}_1$$

With

$$d\tilde{Z}_1 = dZ_1 + \mathcal{V} dt \quad (15)$$

where $\mathcal{V} = \frac{\mu - r}{\sqrt{v}}$, \tilde{Z}_1 is a Brownian motion under \mathbb{Q} and \mathcal{V} is the risk premium divided by the volatility (market price of interest rate risk).

Finally, we can rewrite under \mathbb{Q} :

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t \left(d\tilde{Z}_{1,t} - \mathcal{V} dt \right) = r S_t dt + \sqrt{v_t} S_t d\tilde{Z}_{1,t}$$

$$\begin{aligned} dv_t &= \kappa [\theta - v_t] dt + \sigma \sqrt{v_t} \left(d\tilde{Z}_{2,t} - \Lambda dt \right) \\ &= [\kappa [\theta - v_t] - \Lambda \sigma \sqrt{v_t}] dt + \sigma \sqrt{v_t} d\tilde{Z}_{2,t} \end{aligned}$$

This change of probability is possible by virtue of Girsanov's theorem.

Finally, if we pose $\tilde{\kappa} = \kappa + \lambda$ and $\tilde{\theta} = \frac{\theta\kappa}{\tilde{\kappa}}$ then the modeling of the asset under \mathbb{Q} becomes :

$$\begin{aligned} dS_t &= r S_t dt + \sqrt{v(t)} S_t d\tilde{Z}_{1,t} \\ dv_t &= \tilde{\kappa}(\tilde{\theta} - v(t))dt + \sigma \sqrt{v(t)} d\tilde{Z}_{2,t} \\ \langle d\tilde{Z}_1, d\tilde{Z}_2 \rangle &= \rho dt \end{aligned}$$

A.3 Obtaining data on options and problems encountered

As a reminder, our approach was to find the right parameters in the Heston model in order to project discretized trajectories. To do so, we wanted to minimize the error between the predictions and the observed data. Therefore, we needed to have the options market data for a given underlying asset. In this data we also needed some diversity in strike prices and maturities.

One of the first problems we encountered was to be able to collect the desired data on enough options. On Yahoo-finance it is possible to directly collect the price history of an asset in csv format, but there is no similar possibility to directly retrieve the options data on one's machine and to be able to exploit it afterwards.

In order to obtain this data in real time, we used an API (Application Programming Interface) on RapidAPI that we implemented in R. This allowed us to collect a good number of options on Yahoo-finance (about 1000 for Apple) with a lot of data for each contract (its nature, the last exchange price, etc.). We have therefore kept as parameters the strike, the ask, the bid and the maturity. Indeed the mid-price is not displayed, we used the average between the ask and the bid (with a weighting 0.5 / 0.5).

You can see our code at <https://github.com/cladrien/TER.git>

A.4 Constrained optimization of the cost function

Definition A.1 (Projector on a convex). Let E be a Hilbert space, with a norm $\|\cdot\|$ induced by a scalar product (\cdot, \cdot) , and let K be a nonempty closed convex of E . Then, for any $x \in E$, there exists a unique $x_0 \in K$ such that $\|x - x_0\| \leq \|x - y\|$ for any $y \in K$. Let $x_0 = p_K(x)$ be the orthogonal projection of x onto K .

Remark. In our study, we will focus on the convex $U = [-1, 1] \times \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+$. Indeed our vector $p = (\rho, \kappa, \theta, v_0, \sigma, \lambda) \in U$. But if $K = \prod_{i=1,n} [\alpha_i, \beta_i]$, so $(p_K(y))_i = \max(\alpha_i, \min(y_i, \beta_i))$, $\forall i = 1, \dots, n$

Definition A.2 (Projected guard algorithm with fixed pitch). We consider $E = \mathbb{R}^n$. Let $f \in C^1(\mathbb{R}^n, \mathbb{R})$ a convex function, and K a non-empty closed convex. Let $\rho \geq 0$ given, we consider the following algorithm :

Initialization : $x_0 \in K$

Iteration : x_k known, $x_{k+1} = p_K(x_k - \rho \nabla f(x_k))$.

Remark. This algorithm presents interesting results on its convergence [Ciarlet et al., 1989] Indeed, if :

- 1) There is $\alpha \geq 0$ such as $\forall (x, y) \in \mathbb{R}^n \times \mathbb{R}^n$, $(\nabla f(x) - \nabla f(y), x - y) \geq \alpha |x - y|^2$
- 2) There is $M \geq 0$ such as $\forall (x, y) \in \mathbb{R}^n \times \mathbb{R}^n$, $|\nabla f(x) - \nabla f(y)| \leq M |x - y|$

Then :

- 1) There exists a unique element \bar{x} solution of the constrained minimization problem.
- 2) If $0 < \rho < \frac{2\alpha}{M^2}$, the sequence x_k defined by the previous algorithm converges to \bar{x} when $n \rightarrow +\infty$.

Remark. To minimize our cost function Φ we do not use this algorithm as it is. Indeed we could see that the two conditions were not respected. However, we can be inspired by the projection on the convex.

Definition A.3 (Algorithm used). Let $\varepsilon \geq 0$ given.

Initialization : $x_0 \in U$

We compute $\nabla \Phi(x_0)$,

If $\nabla\Phi(x_0) = 0_{\mathbb{R}^n}$, so we return x_0
 Else, $\rho = \frac{1}{10\|\nabla\Phi(x_0)\|_\infty}$
 $x_1 = p_U(x_0 - \rho\nabla\Phi(x_0))$
 Iteration :
 x_k et x_{k-1} known
 While $|\Phi(x_k) - \Phi(x_{k-1})| > \varepsilon$:
 We compute $\nabla\Phi(x_k)$.
 If $\nabla\Phi(x_k) = 0_{\mathbb{R}^n}$, so we return x_k
 Else $\rho = \frac{1}{10\|\nabla\Phi(x_k)\|_\infty}$
 $x_{k+1} = p_U(x_k - \rho\nabla\Phi(x_k))$
 If $|\Phi(x_{k+1}) - \Phi(x_k)| \leq \varepsilon$:
 We return p_{k+1} .

Remark. In fact, we add a parameter of maximum number of iterations. Indeed the computation of the gradient of the cost function is quite heavy.

Remark. Concerning the choice of the step, we chose this quantity in order not to converge too quickly towards the edges and to remain blocked. Indeed, in our study we had at the point $p = (0, 1, 1, 1, 1)$ (relatively close to the boundary of the set U), $\|\nabla\Phi(p_0)\|_\infty = 25773$ which is quite high. We need to take a step which takes into account the value of the gradient in order not to approach the frontier too quickly (we can suppose that the optimal vector is relatively close to this frontier).

B Learning algorithm for GAN

Algorithm 1 Implemented WGAN-GP training algorithm. Base values for parameters $m = 56, n_{\text{critic}} = 5$, base $lr_c = 3e - 4, \max lr_c = 1e - 3$, base $lr_g = 1e - 4, \max lr_g = 1e - 3$

Require: baselr , lower bound on learning rate. max lr , higher bound on learning rate. m , batch size. epochs , number of training iterations. p , probability to add instance noise. C , critic with parameters \mathcal{W}_c . G , generator with parameters \mathcal{W}_g . P_{data} , distribution of train data. P_z , sample noise distribution

```

for epoch in range (epochs) do
  for  $i$  in range ( $n_{\text{critic}}$ ) do
    | sample batch real,  $X_{\text{real}} = \{x_{\text{real}}^{(1)}, \dots, x_{\text{real}}^{(m)}\}$  from data  $P_{\text{data}}$ ;
    sample batch noise,  $Z = \{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $P_z$ ;
     $\text{scores}_{\text{real}} = C(X_{\text{real}})$ ;
     $\text{scores}_{\text{fake}} = C(G(Z))$ ;
    GP =  $\text{gradient}_{\text{penalty}}(\text{scores}_{\text{real}}, \text{scores}_{\text{fake}})$ ;
     $c_{\text{loss}} = \frac{1}{m} \sum_{i=1}^m \text{scores}_{\text{fake}}^{(i)} - \frac{1}{m} \sum_{i=1}^m \text{scores}_{\text{real}}^{(i)} + \text{GP}$ ;
     $\alpha_d = \text{CyclicLR}(\text{epoch}, \text{base}lr_c, \max lr_c)$ ;
     $\mathcal{W}_c = \mathcal{W}_c + \alpha_d \cdot \text{RMSprop}(\text{loss}_c, \mathcal{W}_c)$ 
  end for
  sample batch noise,  $Z = \{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $P_z$ 
   $\text{scores}_{\text{fake}} = D(G(Z))$ ;
   $g_{\text{loss}} = \frac{1}{m} (\sum_{i=1}^m \text{scores}_{\text{fake}}^{(i)})$ 
   $\alpha_g = \text{CyclicLR}(\text{epoch}, \text{base}lr_g, \max lr_g)$ ;
   $\mathcal{W}_g = \mathcal{W}_g \rightarrow \alpha_g \cdot \text{RMSprop}(\text{loss}_g, \mathcal{W}_g)$ 
end for

```

C An avenue to explore

To the best of our knowledge, all papers apply to model the *returns*. This series is obtained by 1-step differentiation from the asset price. The question then arises : Why would log-return be optimal ? Would over-differentiation be the reason for the efficient markets hypothesis ? All these questions are posed in [De Prado, 2018].

Arbitrage forces financial series to exhibit a very low signal-to-noise ratio. And to make matters worse, stationary transformations further reduce the signal by removing memory. But we believe that price series have memory (hypothesis

supported in [De Prado, 2018]). This is because each value is dependent on a long history of all previous levels. Of course, if there is no memory, and even with all the statistical techniques we can imagine, we will get nothing.

Stationarity is an essential assumption in machine learning, but we don't want to overwrite all the memory because memory is the basis of the predictive power of the model. In order to model the returns well, we propose to take into account the memory of the prices. This has not yet been done in asset modeling papers, and constitutes our contribution.

We consider the Backshift operator B applied to the price series S_t , where $B^k S_t = S_{t-k}$, for all k positive. For example, $(1 - B)^2 S_t = S_t - 2S_{t-1} + S_{t-2}$. Now we have :

$$(x + y)^n = \sum_{k=0}^n \binom{n}{k} x^k y^{n-k} = \sum_{k=0}^n \binom{n}{k} y^k x^{n-k}$$

We have the generalization of Newton's binomial for real d :

$$(1 + x)^d = \sum_{k=0}^{\infty} \binom{d}{k} x^k :$$

We arrive at our point of interest :

$$(1 - B)^d = \sum_{k=0}^{\infty} \binom{d}{k} (-B)^k$$

If we develop a little :

$$(1 - B)^d = \sum_{k=0}^{\infty} (-B)^k \prod_{i=0}^{k-1} \frac{d-i}{k-i}$$

Instead of considering the series of returns (1 step differentiation), we plan to use the series :

$$\tilde{X}_t = \sum_{k=0}^{\infty} \omega_k X_{t-k}$$

With the weights $\omega = \left\{1, -d, \frac{d(d-1)}{2!}, -\frac{d(d-1)(d-2)}{3!}, \dots, (-1)^k \prod_{i=0}^{k-1} \frac{d-i}{k-i}, \dots\right\}$.

This series is intended to preserve memory. Moreover, we find in [De Prado, 2018] the proof that demonstrates the recurrence relation that makes it much easier and more efficient to compute this series :

$$\omega_k = -\omega_{k-1} \frac{d-k+1}{k}$$

This allows us to demonstrate very simply the convergence of the weights, because $\left| \frac{\omega_k}{\omega_{k-1}} \right| = \left| \frac{d-k+1}{k} \right| < 1$.

Now we have to discuss the choice of the value of d . The aim is to find the smallest value allowing the series to be stationary (using an ADF test). This allows us (according to our experiments), to find a stationary series correlated to the course at about 98% compared to a correlation of less than 1% for the series of the 1 step differentiation !