Neural Network method for GARCH parameters calibration

Mohamed Raed Blel*,[1]

[1]Laboratoire de Probabilités Statistiques et Modélisation, LPSM

October 2, 2024

Contents

*Corresponding author: www.linkedin.com/in/mohamed-raed-blel-link

1 Introduction

This study introduces a novel neural network approach for calibrating GARCH parameters. The proposed method includes three key advancements: first, the development of an enhanced neural network model for predicting GARCH parameters; second, the creation of a comprehensive neural network capable of accurately estimating all parameters simultaneously; and third, the introduction of a modified loss function with a penalization term to ensure that the predicted parameters satisfy the ergodic conditions. This approach significantly diverges from previous methods, offering superior performance and versatility in various applications, including the calibration of the SABR model and the estimation of implied volatility.

1.1 Completed work

We summarize the key accomplishments of this study as follows:

1. **Volatility Approximation of the SABR Model:** We propose a novel approach to approximate the volatility of the SABR model using the GARCH model. Inspired by previous research, we refine certain theoretical aspects and introduce a neural network designed to accurately estimate the parameters governing the GARCH volatility trajectory.

2. **Comparison with Direct Minimization Algorithms:** To evaluate the effectiveness of our neural network model, we conduct a comprehensive comparison with various direct minimization algorithms. Our findings indicate that the neural network outperforms these algorithms, especially when dealing with high-order moment challenges.

3. **Model Robustness:** The robustness of our model is demonstrated through its stability under significant data perturbations, distinguishing it from other direct minimization algorithms, which often exhibit instability in challenging regions.

4. **Application to Time Series Data:** We showcase the model's efficiency in predicting GARCH parameters in high-sample scenarios through practical applications on time series data. A comparative analysis with the maximum likelihood method establishes equivalent performance, with the added advantage of superior computational efficiency offered by our neural network model.

Remarks

- The primary objective is to calibrate the parameters (ν, β, and ρ) of the SABR model using the forward rate time series, without direct reliance on the volatility time series, which is often not directly observable in practice. By focusing on the forward rate time series, we aim to relate the GARCH volatility to the SABR model volatility, leveraging the results from the "CE-GEN algorithm" project.
- Another potential approach for calibrating the SABR model involves using standard methods based on GARCH volatility approximation. Comparing these conventional methods with our proposed neural network approach could provide valuable insights into the strengths and limitations of each, contributing to a broader understanding of SABR model calibration.

2 Neural network GARCH parameters estimation method

2.1 Estimating GARCH Parameters by Neural Networks

In this section, we investigate the work developed by L. DeClerk and S. Savlev in their article on estimating GARCH parameters using a Neural Networks method. The idea is to assume that the volatility of our time series can be expressed as a function of the previous variance along with a stochastic variable x_t to forecast future variance. The variable x_t is generally assumed to be the log return of the price between t and $t-1$, and is considered to be conditionally Gaussian. A GARCH(1,1) model is then defined by:

$$\sigma_t^2 = \alpha_0 + \alpha_1 x_{t-1}^2 + \beta_1 \sigma_{t-1}^2, \tag{1}$$

where $x_t = \sigma_t Z_t$ and Z_t is an independent, identically distributed random variable with mean zero and variance equal to one. When Z_t is conditionally normal, the model is referred to as GARCH-normal(1,1).

To fit the parameters of the GARCH model to describe market stock data, the maximum likelihood method can be used. However, these techniques become

time-consuming, especially in high-frequency settings. To address this problem, a neural network is trained to fit the parameters of the GARCH model using explicit equations provided for the second-order moment $E[x^2]$, the fourth-order standardized moment Γ_4, and the sixth-order standardized moment Γ_6 as follows:

$$E[x^2] = \sigma^2 = \frac{\alpha_0}{1 - \alpha_1 - \beta_1}, \tag{2}$$

$$\Gamma_4 = \frac{E[x^4]}{(E[x^2])^2} = 3 + \frac{6\alpha_1^2}{1 - 3\alpha_1^2 - 2\alpha_1\beta_1 - \beta_1^2}, \tag{3}$$

$$\Gamma_6 = \frac{E[x^6]}{(E[x^2])^3} = \frac{15(1 - \alpha_1 - \beta_1)}{3\left(1 + \frac{3(\alpha_1+\beta_1)}{1-\alpha_1-\beta_1} + \frac{3\left(1+\frac{2(\alpha_1+\beta_1)}{1-\alpha_1-\beta_1}\right)(\beta_1^2+2\alpha_1\beta_1+3\alpha_1^2)}{1-3\alpha_1^2-2\alpha_1\beta_1-\beta_1^2}\right) - 15\alpha_1^3 - 9\alpha_1^2\beta_1 - 3\alpha_1\beta_1^2 - \beta_1^3}. \tag{4}$$

The authors also suggest using the autocovariance function with lag n as input to the neural network, as the autocovariance function provides more comprehensive information over a time period than moments alone, which offer insights for specific points in time. Bollerslev derived an equation for the autocovariance of the square of the return, $\mathrm{Cov}(x_t^2, x_{t+n}^2)$, with lag n, denoted as γ_n, by:

$$\hat{\gamma}_n = (E[x^2]) \left[2 + \frac{6\alpha_1^2}{1 - 3\alpha_1^2 - 2\alpha_1\beta_1 - \beta_1^2}\right] (\alpha_1 + \beta_1)^n. \tag{5}$$

2.1.1 Correction of $\hat{\gamma}_n$ Formula

To further enhance the model, a correction was made to the initial formula for $\hat{\gamma}_n$. Starting from the GARCH(1,1) equation, we can write:

$$x_t^2 = \alpha_0 + (\alpha_1 + \beta_1)x_{t-1}^2 - \beta_1\nu_{t-1} + \nu_t,$$

where $\nu_t = x_t^2 - \sigma_t^2$. Thus, the covariance $\mathrm{Cov}(x_t^2, x_{t+n}^2)$ can be expressed as:

$$\mathrm{Cov}(x_t^2, x_{t+n}^2) = (\alpha_1 + \beta_1)\mathrm{Cov}(x_t^2, x_{t+n-1}^2) - \beta_1\mathrm{Cov}(x_t^2, \nu_{t+n-1}) + \mathrm{Cov}(x_t^2, \nu_{t+n}).$$

Given that $E[\nu_t] = 0$, we simplify:

$$\mathrm{Cov}(x_t^2, \nu_{t+n-1}) = E[x_t^2\nu_{t+n-1}] = 0,$$

as

$$\mathbb{E}[\nu_t] = \mathbb{E}[x_t^2] - \mathbb{E}[\sigma_t^2] = 0,$$

we obtain,

$$Cov(x_t^2, \nu_{t+n-1}) = \mathbb{E}[x_t^2 \nu_{t+n-1}]$$

knowing that $x_t^2 = Z_t^2 \sigma_t^2$ we get

$$\mathbb{E}[x_t^2 \nu_{t+n-1}] = \mathbb{E}[\mathbb{E}[x_t^2 \nu_{t+n-1}|_{\mathcal{F}_{t-1}}]] = \mathbb{E}[x_t^2 \mathbb{E}[\nu_{t+n-1}|_{\mathcal{F}_{t-1}}]] = 0,$$

as we have independence between σ_t and Z_t, and we know that $(Z_t)_t$ is an independent normal family we get $\mathbb{E}[\nu_{t+n-1}|_{\mathcal{F}_{t-1}}] = \mathbb{E}[\sigma_{t+n-1}^2(Z_{t+n-1}^2 - 1)|_{\mathcal{F}_{t-1}}] = \mathbb{E}[\sigma_{t+n-1}^2|_{\mathcal{F}_{t-1}}]\mathbb{E}[(Z_{t+n-1}^2 - 1)|_{\mathcal{F}_{t-1}}] = 0$.
finally,

$$\gamma_n = (\alpha_1 + \beta_1)\gamma_{n-1},$$

hence, the corrected formula for γ_n is:

$$\gamma_n = (E[x^2])^2 \left[2 + \frac{6\alpha_1^2}{1 - 3\alpha_1^2 - 2\alpha_1\beta_1 - \beta_1^2}\right](\alpha_1 + \beta_1)^n. \tag{6}$$

The neural network is trained on a dataset formed by $(E[x^2], \Gamma_4, \Gamma_6)$ to learn α_1, and from this, β_1 is deduced using:

$$\beta_1 = \sqrt{1 - 2\alpha_1^2 - \frac{6\alpha_1^2}{\Gamma_{4,emp} - 3}} - \alpha_1, \tag{7}$$

and α_0 is computed by:

$$\alpha_0 = \sigma_{emp}^2(1 - \alpha_1 - \beta_1), \tag{8}$$

where $\Gamma_{4,emp}$ and σ_{emp} are empirically obtained from the time series. Following the same procedure as in the referenced article, numerical instabilities were encountered. To address this, the ReLU activation function was replaced with a sinusoidal function.

2.1.2 Neural Network Architecture

The neural network developed in this work differs significantly from the one used by previous authors. We implemented a neural network with four layers containing 1280, 2048, 2048, and 1280 nodes, respectively. In contrast, the referenced study used ReLU activation functions, while we employed a sinusoidal activation function and added a sigmoid output activation function to ensure that the predicted parameters are constrained within the range [0, 1]. Both approaches utilized Min-Max scaling for data normalization and the Adam optimization algorithm, but with different learning rates: 10^{-4} in our approach versus 10^{-2} in the previous work. Our adjustments have resulted in improved stability and performance.

2.1.3 Data Set

The training set was generated by uniformly sampling the variables α_0, α_1, and β_1 within the ranges $[10^{-2}, 10^{-3}]$, $[0, 0.3]$, and $[0, 1]$, respectively. It was crucial to ensure that $\alpha_1 + \beta_1 < 1$ to maintain the existence of the GARCH solution. We sampled $M = 5 \times 10^4$ data points of $(\alpha_0, \alpha_1, \beta_1)$, which were then used to generate the training data sets $(E[x^2], \Gamma_4, \Gamma_6)$ using equations (2), (3), and (4), or $(E[x^2], \Gamma_4, \hat{\gamma}_n)$ using equations (2), (3), and (6). The mean squared difference (MSD) was used as a loss function to measure the distance between the predicted α_1^θ (output of the neural network) and the actual model value α_1. The loss function is defined as follows:

$$MSD = \frac{1}{M} \sum_{i=1}^{M} (\alpha_{1i}^\theta - \alpha_{1i})^2. \tag{9}$$

2.2 Numerical Results Using Different Architectural Networks

In this section, we present the outcomes of the training and validation phases, employing the same architecture and hyperparameters as specified by the authors in the initial study. We also introduce our own results obtained after examining various parameters in the following sections.

2.2.1 A Standard Neural Network with Fixed Learning Rate

In all visual representations, the out-of-sample predictions generated by the neural network are depicted in blue against the model's α_1 values. The yellow line represents the ideal fitting function $y = x$, and the green line signifies the best-fitting curve. All depicted results are based on a dataset comprising 10^4 data points.

Figure 1 provides an illustrative example of fitting the parameter α_1 using the neural network model, with further refinements anticipated in subsequent sections.

Moving from Figure 2 to Figure 5, we showcase results obtained from distinct training sets, namely $(E[x^2], \Gamma_4, \hat{\gamma}_1)$, $(E[x^2], \Gamma_4, \hat{\gamma}_2)$, $(E[x^2], \Gamma_4, \hat{\gamma}_3)$, and $(E[x^2], \Gamma_4, \hat{\gamma}_4)$. While these results exhibit favorable patterns, there is ongoing work to enhance their stability. Consequently, we investigate the impact of specific hyperparameters, such as the learning rate, and explore alternative neural network models, including convolutional networks.

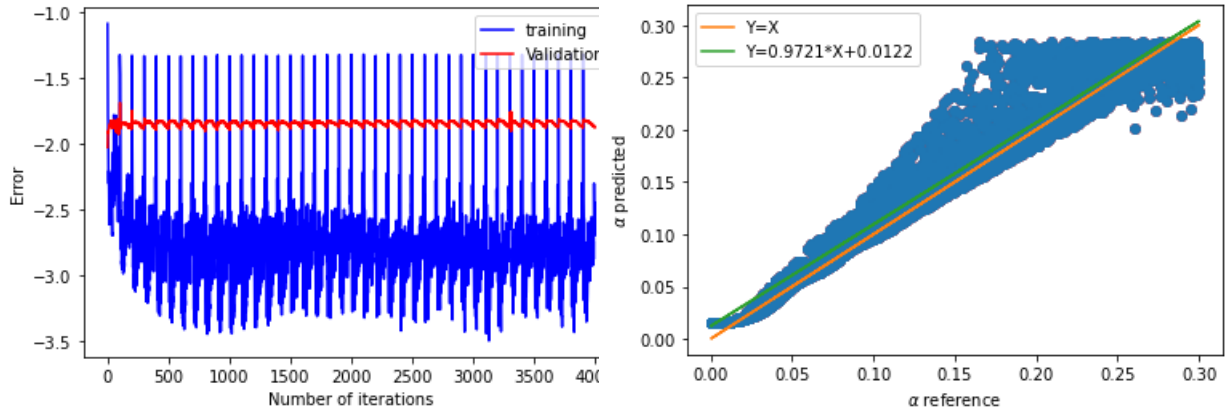


Figure 1: Left: Training error and validation error for the training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) in logarithmic scale; Right: Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6).

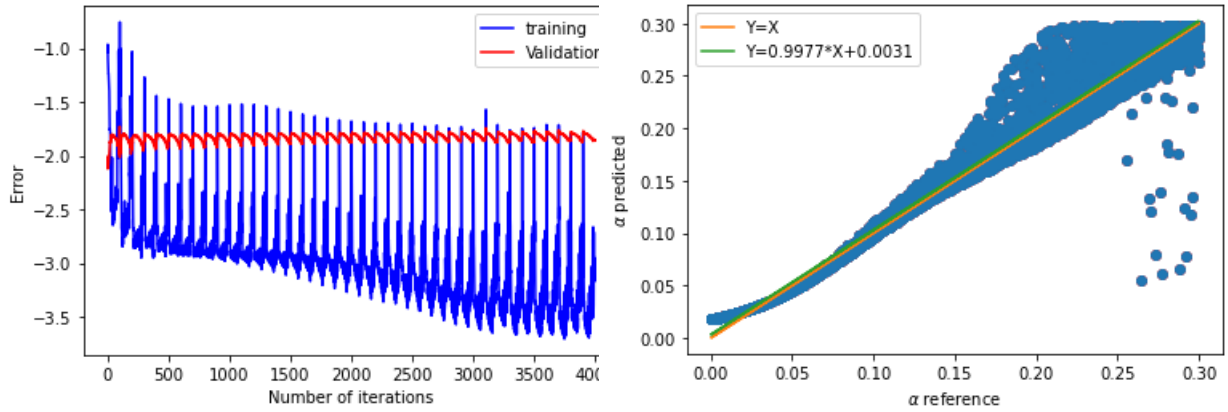


Figure 2: Left: Training error and validation error for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_1$) in logarithmic scale; Right: Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_1$).

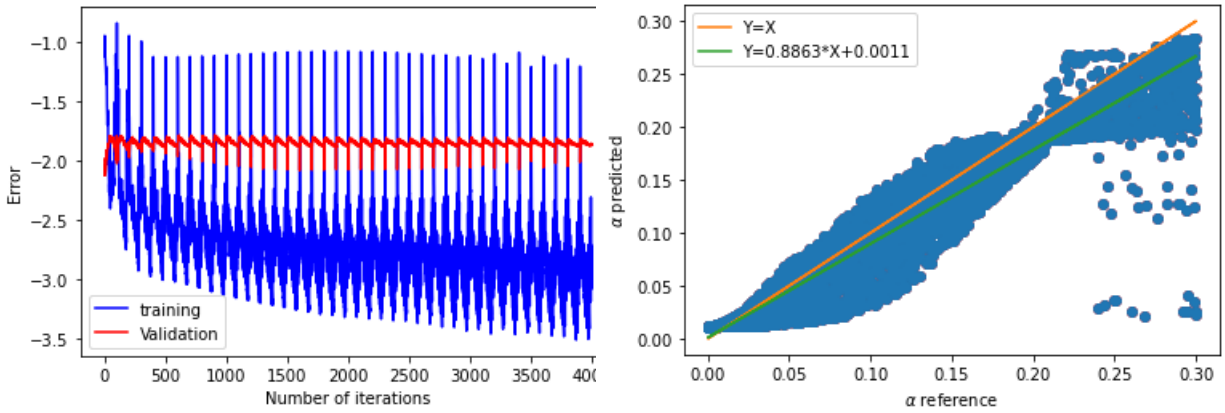


Figure 3: Left: Training error and validation error for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_2$) in logarithmic scale; Right:Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_2$).

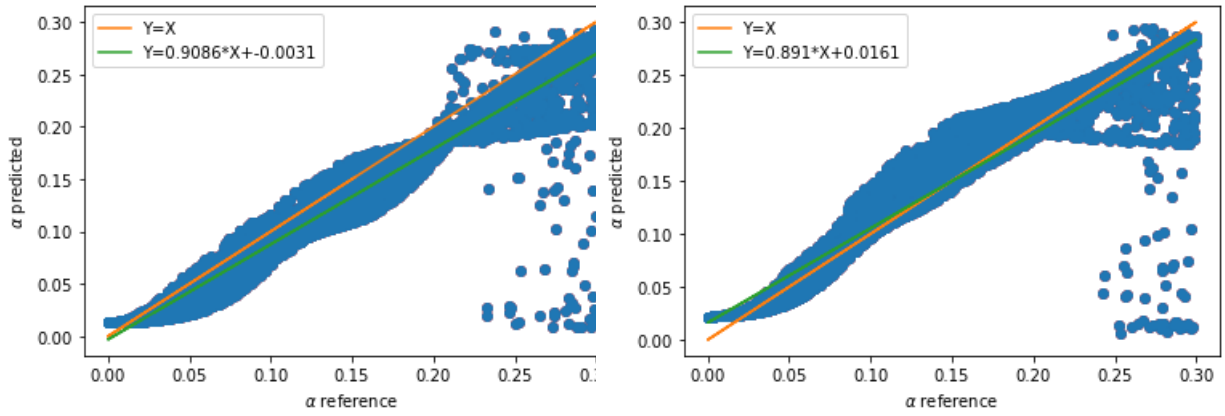


Figure 4: Left: Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_3$); Right: Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_4$).

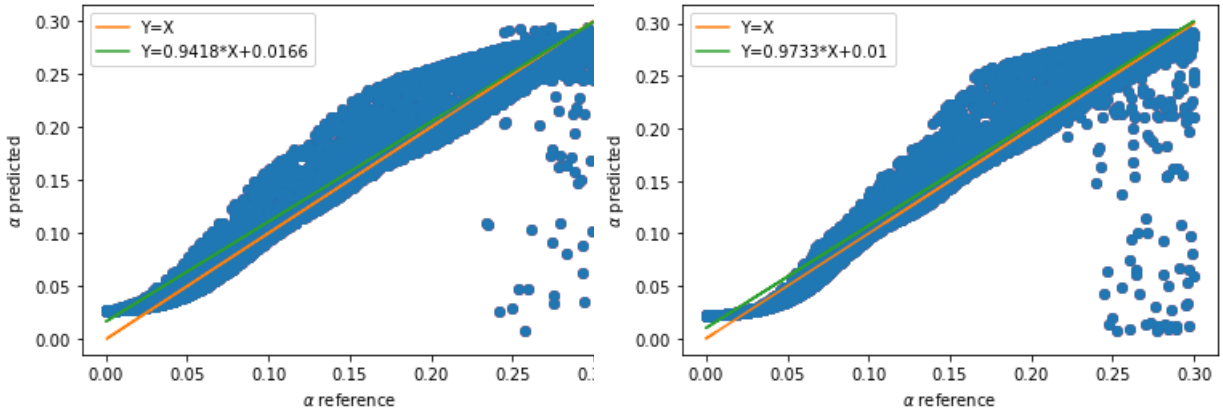


Figure 5: Left: Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_5$).; Right: Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_7$).

Impact of Hyperparameters

In this section, we conduct an in-depth analysis of the effects of two key hyperparameters: the learning rate and the number of neurons in the first layer, on our model's performance. While there is extensive literature on this topic, we focus on a concise set of results for clarity. In Figure 6, we use a fixed learning rate and two different neuron configurations: 128 neurons on the left and 1280 on the right. The results demonstrate that using 1280 neurons leads to better performance. Increasing the number of neurons appears to enhance the model's ability to capture more metastable regions, providing a plausible reason for this improvement.

Figure 7 explores a dynamic learning rate approach, where the learning rate changes between $(10^{-1}, 10^{-2}, 10^{-3}, 10^{-4})$. This adaptive learning rate is controlled by a secondary neural network tasked with adjusting the rate. The results indicate that this method is less effective. It seems the network tends to jump between different potential wells without thoroughly exploring any single one, suggesting challenges in achieving stable and meaningful learning rate adaptations. Further refinements may be required to improve the effectiveness of this adaptive strategy.

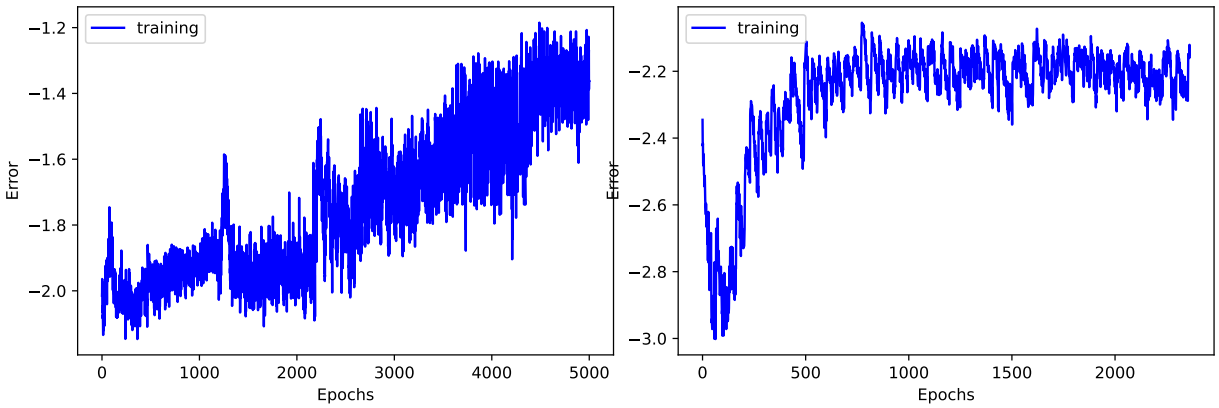


Figure 6: Left: Training error for the training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) in logarithmic scale using 128 neurons in the first layer; Right: Training error for the same set using 1280 neurons in the first layer. Fixed learning rate $lr = 10^{-4}$.

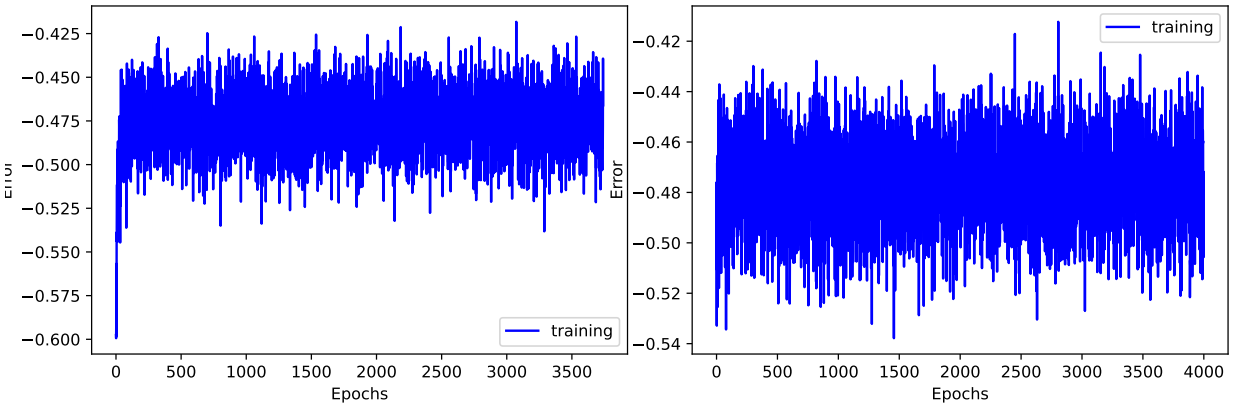


Figure 7: Left: Training error for the training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) in logarithmic scale using dynamic learning rate and using 640 neurons in the first layer; Right: Training error for the same set using 1280 neurons in the first layer. Dynamic learning rate.

2.2.2 A Convolutional Network with a Fixed Learning Rate

In this section, we employ a convolutional neural network consisting of four convolutional layers, with each layer containing 1280, 2048, 2048, and 1280 neurons, respectively. The sinusoidal function is used as the activation function, although

we also experimented with the ReLU function and other activation functions. For the final layer, we use the sigmoid activation function to ensure that the output values are constrained within the range $[0, 1]$.

Using L-Moments for the Training Set

We propose using a fixed learning rate of $lr = 10^{-4}$ for the generator, along with a training dataset composed of the first five L-moments of the time series, defined as:

$$\lambda_r = r^{-1}\binom{N}{r}^{-1} \sum_{x_1<\cdots<x_j<\cdots<x_r} (-1)^{r-j}\binom{r-1}{j} x_j$$

Additionally, we use the L-moment ratios for the third, fourth, and fifth L-moments, defined as:

$$\tau_i = \frac{\lambda_i}{\lambda_2} \quad \text{for} \quad i \in \{3, 4, 5\}$$

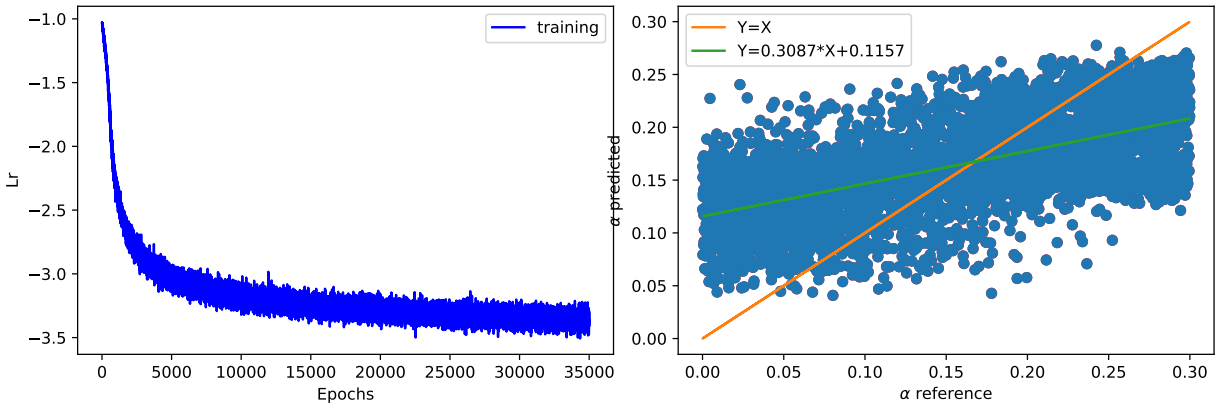


Figure 8: Left: Training error for the training set $(\lambda_1, \lambda_2, \tau_3, \tau_4, \tau_5)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 compared to α_1 of the reference. We use here 5.10^4 data.

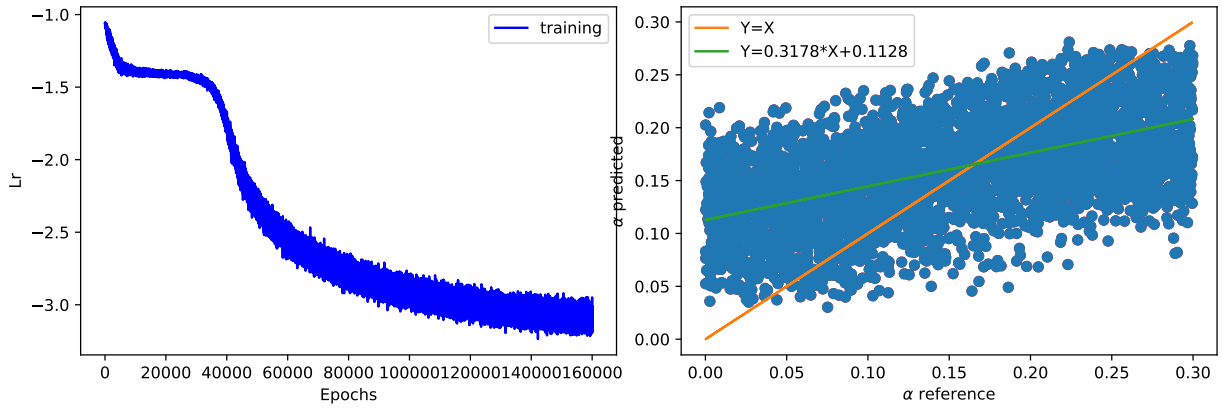


Figure 9: Left: Training error for the training set $(\lambda_1, \lambda_2, \tau_3, \tau_4, \tau_5)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 compared to α_1 of the reference. We use here 5.10^5 data.

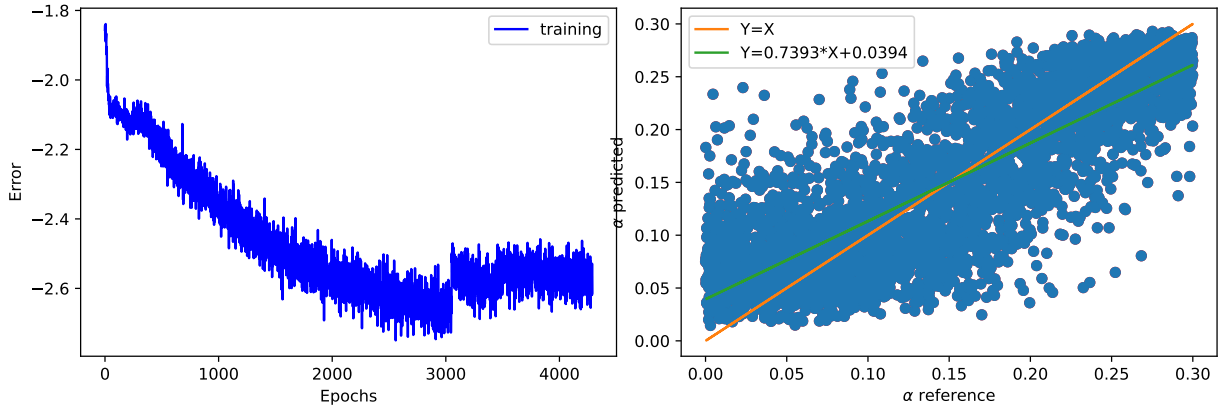


Figure 10: Left: Training error for the training set (τ_3, τ_4, τ_5) in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 compared to α_1 of the reference. We use here 5.10^4 data.

We remark that composing the training data by the L-moments using convolutional neural network is less effective.

2.2.3 A Convolutional Network Using Dynamic Learning Rate

In this section, we propose using a dynamic learning rate for the generator by employing two neural networks, one nested within the other. The first network is

the generator that adjusts the parameter α_1, while the second network, referred to as the "decision maker," is tasked with learning to select the optimal learning rate.

The decision maker is an unsupervised neural network consisting of a linear layer with four neurons, followed by a softmax activation function, which allows the output to be interpreted as a set of probabilities assigned to each neuron. Each neuron index corresponds to one of four possible learning rates. The objective is to train the decision maker to select the learning rate that maximizes the probability of achieving better performance. To this end, we introduce the concept of a reward, which is defined as $reward = -generator_{loss}$. The decision maker's loss is therefore given by:

$$DecisionMaker_{Loss} = -\log(p) \cdot reward$$

where p represents the probability of the selected learning rate. The decision maker's loss is computed as the negative expected reward, which incentivizes it to choose learning rates that lead to higher rewards (i.e., better model performance).

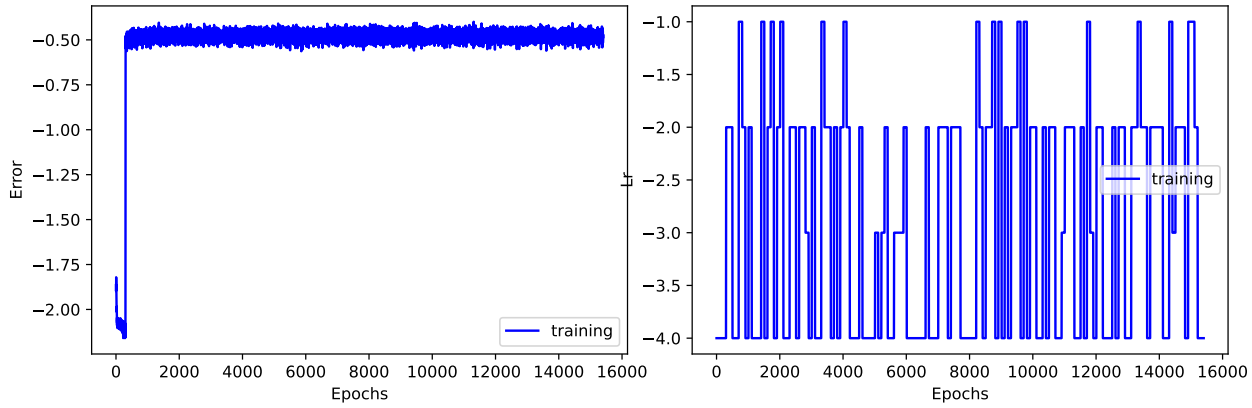


Figure 11: Left: Left: Training error for the training set (τ_3, τ_4, τ_5) in logarithmic scale using adaptive learning rate and using 1280 neurons in the first layer; Right: Evolution of the learning rate during the learning phase.

Message 1: In these initial results, we encountered several challenges in predicting the GARCH parameters, even after enhancing the training set with additional observables such as L-moments or the autocovariance of the squared returns with lag n. We also experimented with different neural network architectures using adaptive learning rates, but the results were still unsatisfactory. In the next section, we explain that this is due to the underlying geometry of the problem.

Specifically, we are dealing with a geometry characterized by many local minima (wells), and the data needs to be chosen carefully to satisfy multiple constraints, particularly those involving the first moments.

2.3 Comparison Between Direct Minimization Algorithms and Neural Network Models for Solving GARCH Parameter Moment Equations

To solve the constrained problem defined by the system of equations (10), (11), and (12), under the constraints (13), (14), and (15), for each triplet $(\mathbb{E}[x^2], \Gamma_4, \Gamma_6)$, we applied several minimization methods. These included Sequential Least Squares Programming (SLSQP), Constrained Optimization by Linear Approximation (COBYLA), trust-region constrained optimization, Nelder-Mead, Conjugate Gradient (CG), L-BFGS-B, Trust-Krylov, Differential Evolution, Random Search, and Couenne. The best results were obtained using the SLSQP, Differential Evolution, Random Search, and Couenne algorithms, as detailed in the following section.

The GARCH moment equations to be solved are:

$$\mathbb{E}[x^2] = \sigma^2 = \frac{\alpha_0}{1 - \alpha_1 - \beta_1}, \tag{10}$$

$$\Gamma_4 = \frac{\mathbb{E}[x^4]}{\mathbb{E}[x^2]^2} = 3 + \frac{6\alpha_1^2}{1 - 3\alpha_1^2 - 2\alpha_1\beta_1 - \beta_1^2}, \tag{11}$$

$$\Gamma_6 = \frac{E(x^6)}{(E(x^2))^3} = \frac{15(1-\alpha_1-\beta_1)^3(1 + \frac{3(\alpha_1+\beta_1)}{1-\alpha_1-\beta_1} + \frac{3(1+\frac{2(\alpha_1+\beta_1)}{1-\alpha_1-\beta_1})(\beta_1^2+2\alpha_1\beta_1+3\alpha_1^2)}{1-3\alpha_1^2-2\alpha_1\beta_1-\beta_1^2})}{1 - 15\alpha_1^3 - 9\alpha_1^2\beta_1 - 3\alpha_1\beta_1^2 - \beta_1^3}, \tag{12}$$

under the constrains:

$$1 - \alpha_1 - \beta_1 > 0, \tag{13}$$

$$1 - 3\alpha_1^2 - 2\alpha_1\beta_1 - \beta_1^2 > 0, \tag{14}$$

$$1 - 15\alpha_1^3 - 9\alpha_1^2\beta_1 - 3\alpha_1\beta_1^2 - \beta_1^3 > 0. \tag{15}$$

First, we present the geometrical challenges faced by any algorithm attempting to solve the constrained problem described above.

Graph of Γ_4 and Γ_6 as Functions of α_1 and β_1:

The following figures illustrate the irregular geometry of the functions Γ_4 and Γ_6 as functions of α_1 and β_1. Note that α_1 is restricted to the range $[0, 3]$ to ensure the finiteness of higher moments (up to the eighth moment), while β_1 lies in $[0, 1]$. As shown in Figures 12 and 16, we observe several irregular zones with multiple peaks, where Γ_4 and Γ_6 can abruptly change signs, transitioning from highly positive to highly negative values with variations smaller than 10^{-14}. This

drastic behavior poses significant challenges for minimization solvers. For instance, one can verify the values of Γ_4 and Γ_6 at $(\alpha_1 = 0.29, \beta_1 = 0.6220307012376284)$ and at $(\alpha_1 = 0.29, \beta_1 = 0.622030701237628)$.

In addition to the issue of peaks, there are flat regions in the geometry, as shown in Figure 14. In these areas, direct minimization algorithms encounter difficulties because the gradient vanishes, leading to an ill-conditioned gradient matrix. As a result, approximating the inverse of the Hessian matrix becomes problematic. In contrast, Figure 13 shows a convex region, far from the flat zones, where all algorithms produce successful results.

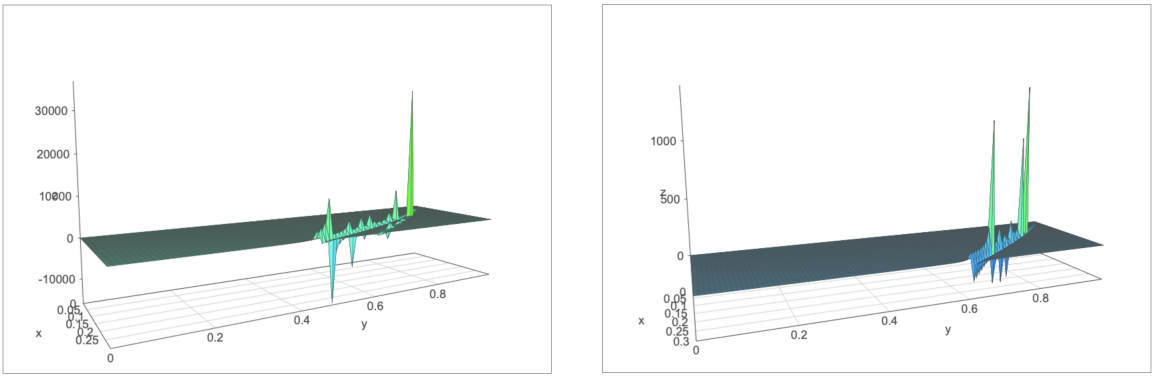


Figure 12: Left: Γ_4 as a function of $\alpha_1 \in [0, 0.3]$ and $\beta_1 \in [0, 1]$; Right: Γ_6 as a function of $\alpha_1 \in [0, 0.3]$ and $\beta_1 \in [0, 1]$.

Regular graph for Γ_4 and Γ_6 on the set $set1 = [0, 0.1] \times [0, 0.7]$

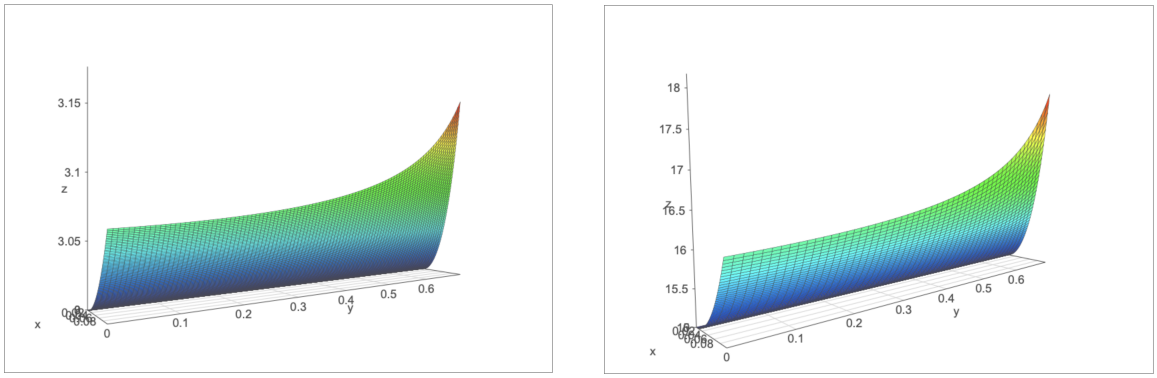


Figure 13: Left: Γ_4 as a function of $\alpha_1 \in [0, 0.1]$ and $\beta_1 \in [0, 0.7]$; Right: Γ_6 as a function of $\alpha_1 \in [0, 0.1]$ and $\beta_1 \in [0, 0.7]$.

Regular flat graph for Γ_4 and Γ_6 on the subset $set11 = [0, 0.01] \times [0, 0.7]$

We observe here the flat zone, where Γ_4 varies only with an order of 10^{-4} for $\alpha_1 \in [0, 0.01]$ and $\beta_1 \in [0, 0.7]$.

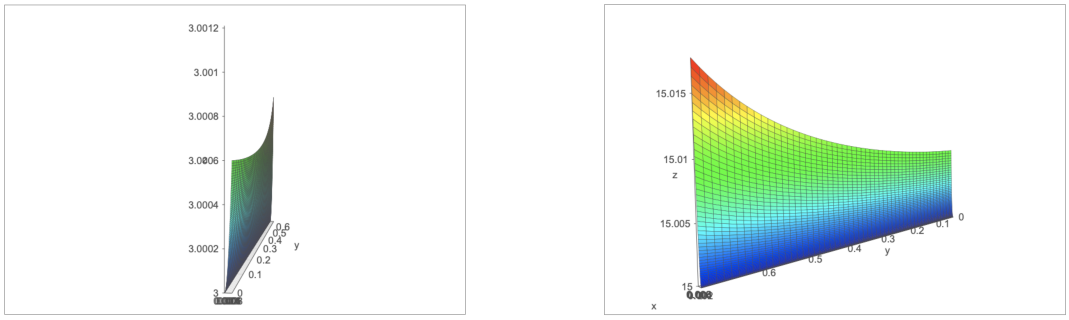


Figure 14: Left: Γ_4 as a function of $\alpha_1 \in [0, 0.01]$ and $\beta_1 \in [0, 0.7]$; Right: Γ_6 as a function of $\alpha_1 \in [0, 0.01]$ and $\beta_1 \in [0, 0.7]$.

Irregular graph for Γ_4 and Γ_6 on the set $set2 = [0, 0.3] \times [0.7, 1]$

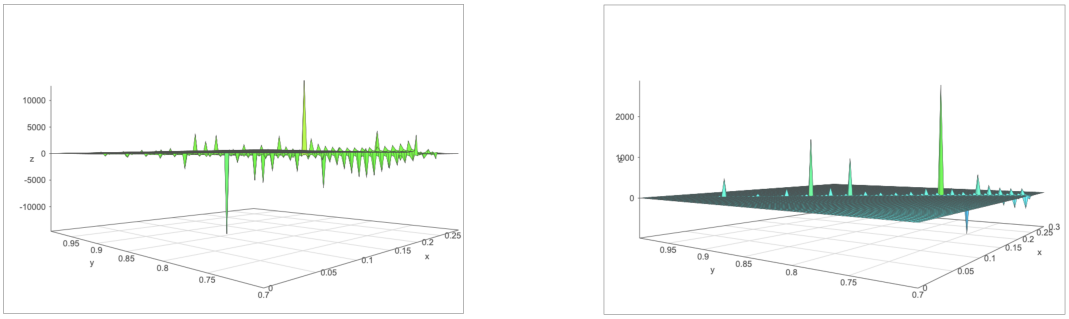


Figure 15: Γ_4 as a function of $\alpha_1 \in [0, 0.3]$ and $\beta_1 \in [0.7, 1]$; Right: Γ_6 as a function of $\alpha_1 \in [0, 0.3]$ and $\beta_1 \in [0.7, 1]$.

Irregular graph for Γ_4 and Γ_6 on the set $set3 = [0.1, 0.3] \times [0, 0.7]$

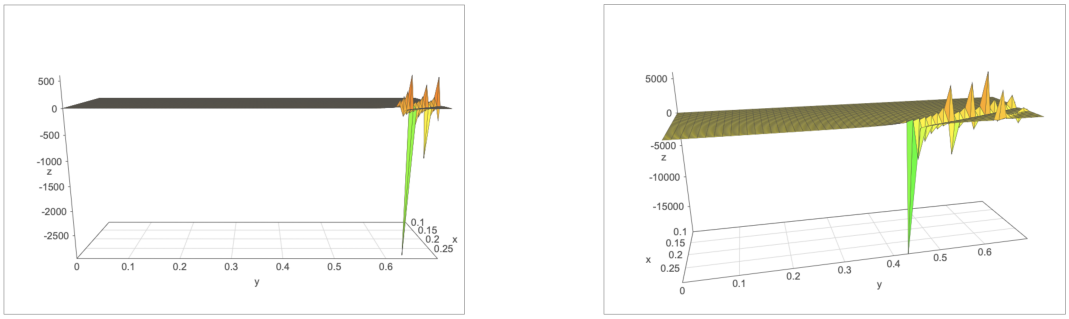


Figure 16: Γ_4 as a function of $\alpha_1 \in [0.1, 0.3]$ and $\beta_1 \in [0, 0.7]$; Right: Γ_6 as a function of $\alpha_1 \in [0.1, 0.3]$ and $\beta_1 \in [0, 0.7]$.

Test case on the first Set $set1 = [0, 0.1] \times [0, 0.7]$ using direct minimization and Neural network comparison:

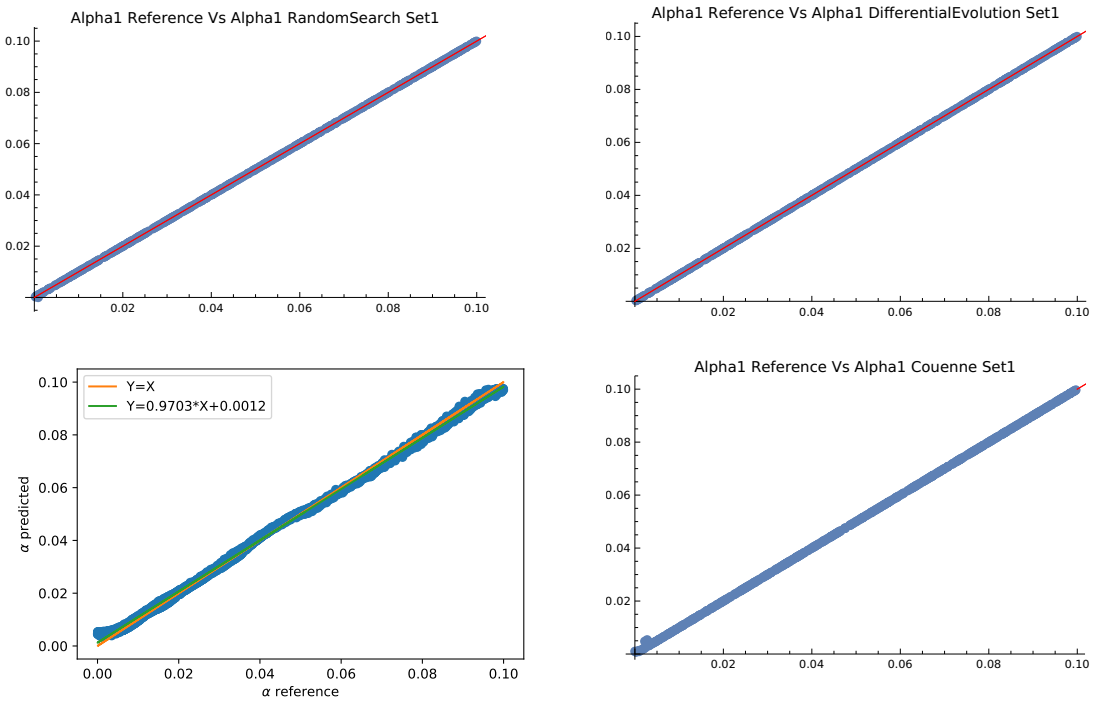
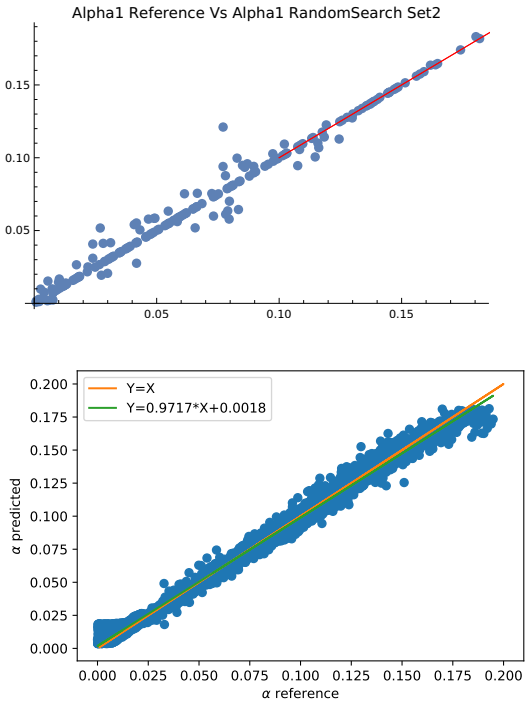
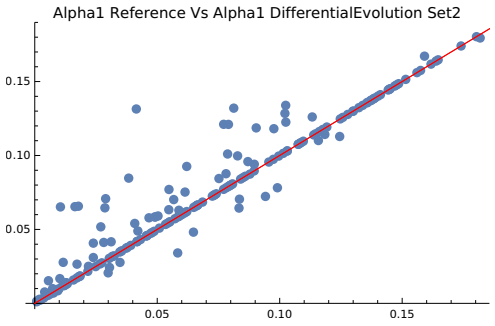


Figure 17: Up and Left:Prediction of α_1 using Random Search algorithm on mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) ; Up and Right: Prediction of α_1 using Differential Evolution algorithm on mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) ;down and left: Prediction of α_1 using Neural Network algorithm on python for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6); down and Right: Prediction of α_1 using Couenne algorithm on mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) on the set of parameters $set1 = [0, 0.1] \times [0, 0.7]$.

Message 2: In the regular zone, the algorithms Random Search, Differential Evolution, Couenne, and the neural network have successfully solved/predicted the GARCH constrained problem with excellent performance. The neural network stands out in terms of computational efficiency, requiring less than 1 second to compute the result for 5000 parameters on a GPU (once trained, with the training session taking approximately 2-3 hours on the GPU). In comparison, direct minimization algorithms take around 3 minutes to converge for only 500 parameters (this was the maximum number of points allowed for the resolution in Mathematica).

Test case on the second Set $set2 = [0, 0.3] \times [0.7, 1]$ using direct minimization and Neural network comparison:





L'image 'aaa' est introuvable.

Figure 18: Top Left: Prediction of α_1 using the Random Search algorithm in Mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6); Top Right: Prediction of α_1 using the Differential Evolution algorithm in Mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6); Bottom Left: Prediction of α_1 using the Neural Network algorithm in Python for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6); Bottom Right: Prediction of α_1 using the Couenne algorithm in Mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) on the parameter set $set1 = [0, 0.3] \times [0.7, 1]$.

Message 3: In the irregular zone (set2), the Random Search and Differential Evolution algorithms were unable to solve the GARCH constrained problem. However, the neural network successfully predicted the GARCH parameters with good performance and continued to outperform in terms of computational time, taking less than 1 second to compute results for 5000 parameters on a GPU (once trained, with the training session taking around 2-3 hours on the GPU). In comparison, direct minimization algorithms took around 3 minutes to converge for only 500 parameters.

Test case on the third Set $set3 = [0.1, 0.3] \times [0, 0.7]$ using direct minimization and Neural network comparison:

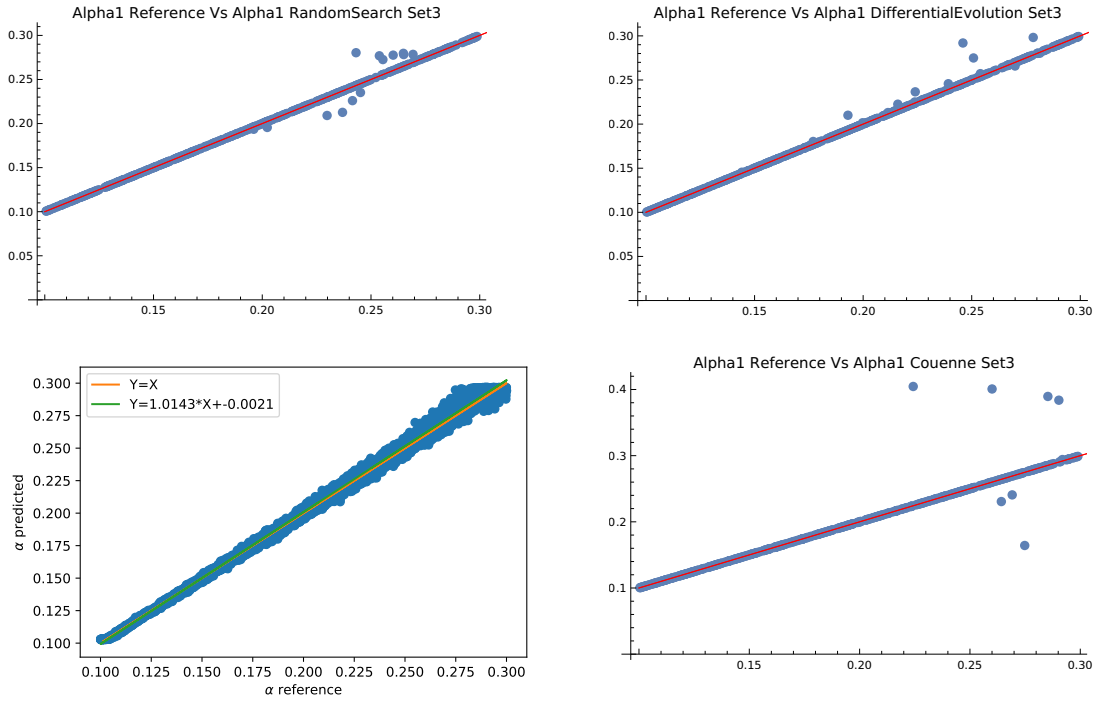


Figure 19: Up and Left:Prediction of α_1 using Random Search algorithm on mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) ; Up and Right: Prediction of α_1 using Differential Evolution algorithm on mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) ;down and left: Prediction of α_1 using Neural Network algorithm on python for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6); down and Right: Prediction of α_1 using Couenne algorithm on mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) on the set of parameters $set3 = [0.1, 0.3] \times [0, 0.7]$.

Message 4: In the irregular zone, the Random Search, Differential Evolution, and Couenne algorithms were able to solve the GARCH constrained problem with good performance, but they failed at certain points. Note that the resolution was performed on a set of 500 parameters, so the error is expected to increase as the number of parameters grows. The neural network showed better prediction in this zone, particularly because the number of parameters for the neural network was 5000. It also remained superior in terms of computational efficiency, taking less than 1 second to compute the result for 5000 parameters on a GPU. In comparison, the direct minimization algorithms took around 10 minutes to converge for only 500 parameters in this zone.

Perturbation Test Cases on the First Set $set1 = [0, 0.1] \times [0, 0.7]$: Comparison Between Direct Minimization and Neural Network

In this section, we investigate the performance of direct minimization algorithms compared to the neural network model for predicting the parameter α_1 by adding noise to the data. We plot several figures showing different magnitudes of the standard deviation of the noise.

Let $\epsilon_1, \epsilon_2, \epsilon_3$ be random normal variables with mean zero and standard deviations that vary from test to test, as shown in the following figures.

All models (neural network or direct minimization) were trained or solved on the perturbed training set $(\mathbb{E}[x^2] + \epsilon_1, \Gamma_4 + \epsilon_2, \Gamma_6 + \epsilon_3)$ for a set of parameters (α_1, β_1) belonging to $set1 = [0, 0.1] \times [0, 0.7]$.

In figure 20 the standard deviation of the noise is taken such that: $std(\epsilon_1) \sim 10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 10^{-2} std(\Gamma_6)$.

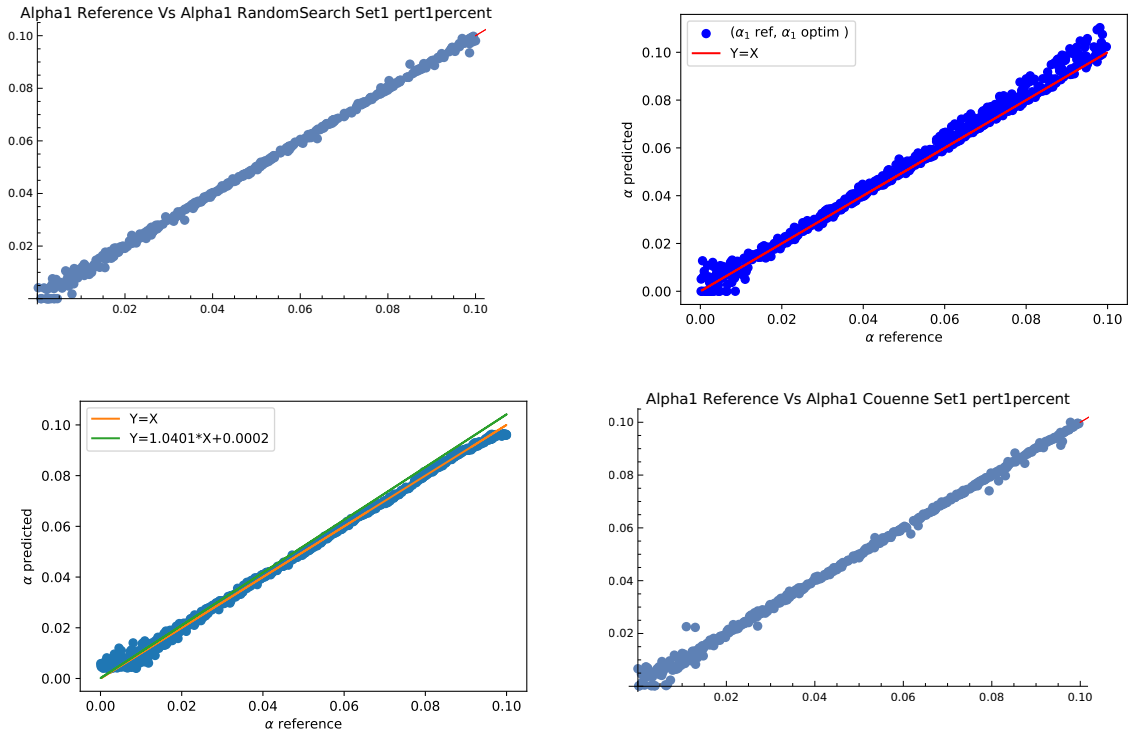
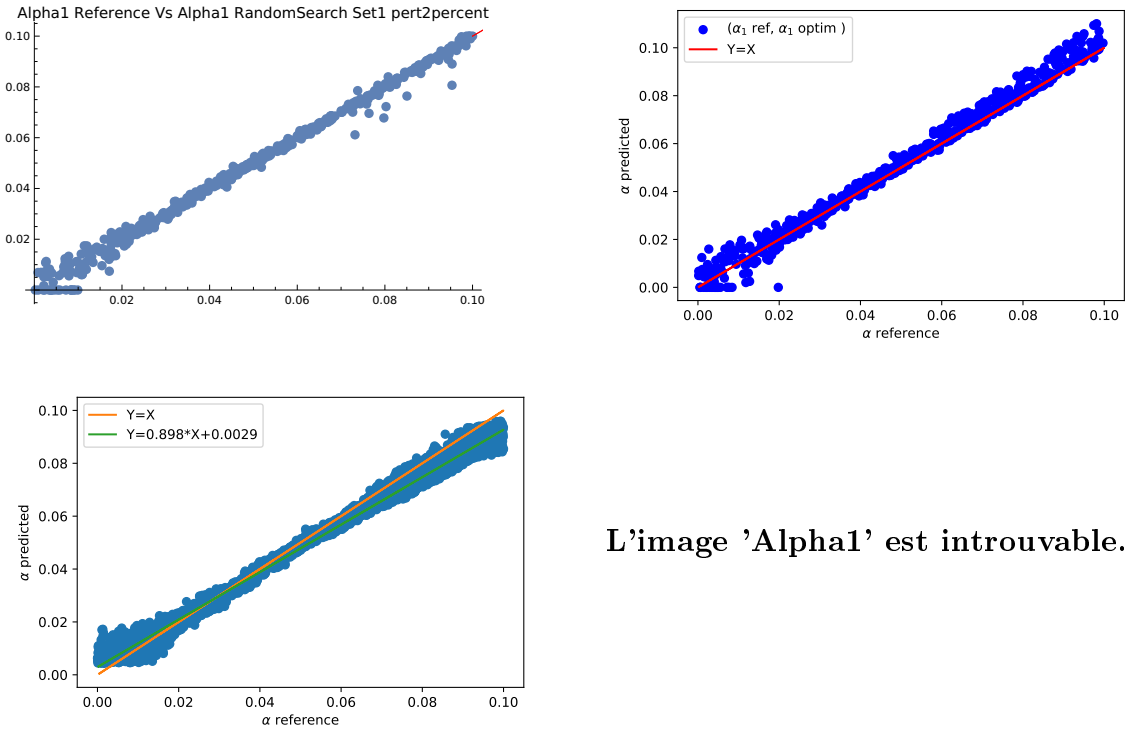


Figure 20: Up and left: Prediction of α_1 using Random Search algorithm; Up and right: Prediction of α_1 using SLSQP algorithm; Down and left: Prediction of α_1 using Neural Network algorithm; Down and right: Prediction of α_1 using Couenne algorithm for the perturbed training set $(\mathbb{E}[x^2] + \epsilon_1, \Gamma_4 + \epsilon_2, \Gamma_6 + \epsilon_3)$ on the set of parameters $set1 = [0, 0.1] \times [0, 0.7]$.

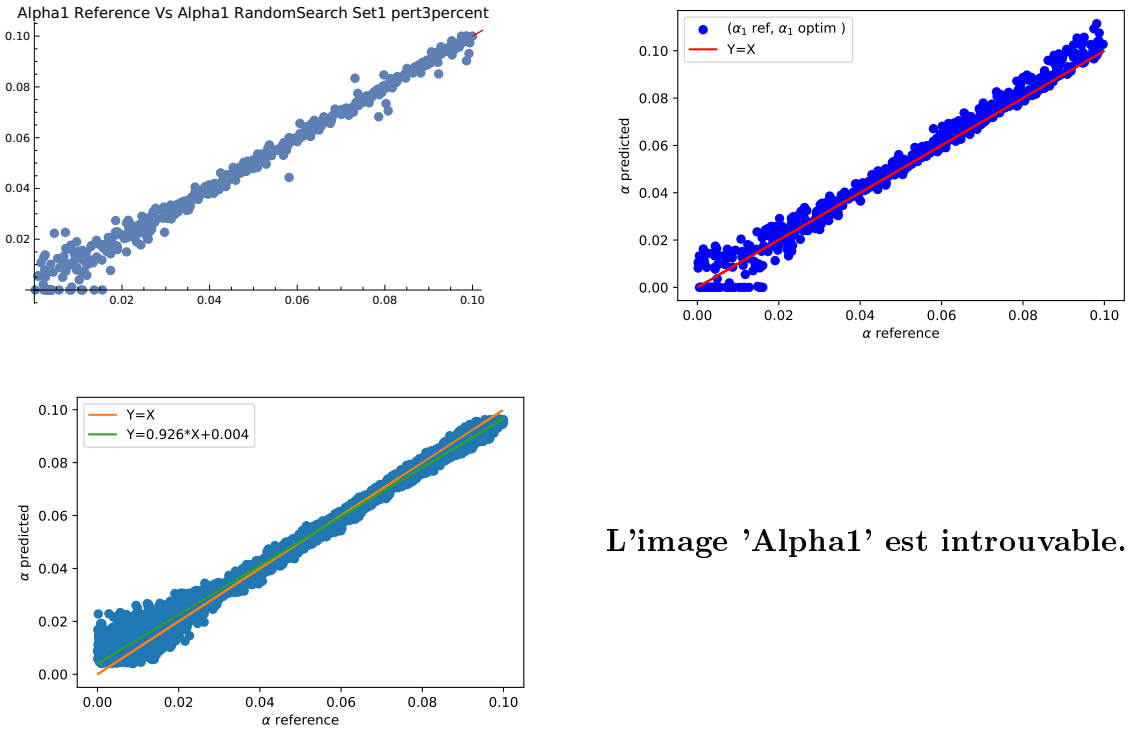
In figure 21, all models are trained (for the neural network model) or solved (for the direct minimization model) on the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) for a set of parameters (α_1, β_1) belonging to $set1 = [0, 0.1] \times [0, 0.7]$, and the standard deviation of the noise is taken such that: $std(\epsilon_1) \sim 2.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 2.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 2.10^{-2} std(\Gamma_6)$.



L'image 'Alpha1' est introuvable.

Figure 21: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using SLSQP algorithm on python; Down and left: Prediction of α_1 using Neural Network algorithm on python for the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) using $std(\epsilon_1) \sim 2.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 2.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 2.10^{-2} std(\Gamma_6)$ on the set of parameters $set1 = [0, 0.1] \times [0, 0.7]$.

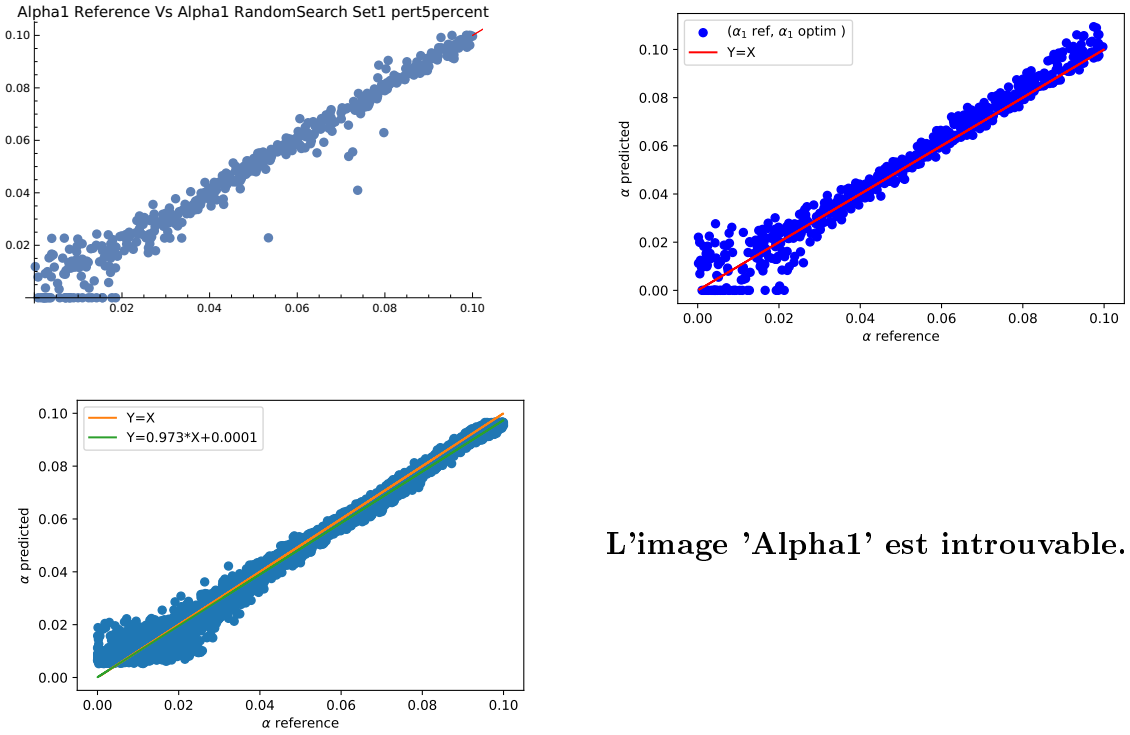
In figure 22, all models are trained (for the neural network model) or solved (for the direct minimization model) on the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) for a set of parameters (α_1, β_1) belonging to $set1 = [0, 0.1] \times [0, 0.7]$, and the standard deviation of the noise is taken such that: $std(\epsilon_1) \sim 3.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 3.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 3.10^{-2} std(\Gamma_6)$.



L'image 'Alpha1' est introuvable.

Figure 22: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using SLSQP algorithm on python; Down and left: Prediction of α_1 using Neural Network algorithm on python for the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) using $std(\epsilon_1) \sim 3.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 3.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 3.10^{-2} std(\Gamma_6)$ on the set of parameters $set1 = [0, 0.1] \times [0, 0.7]$.

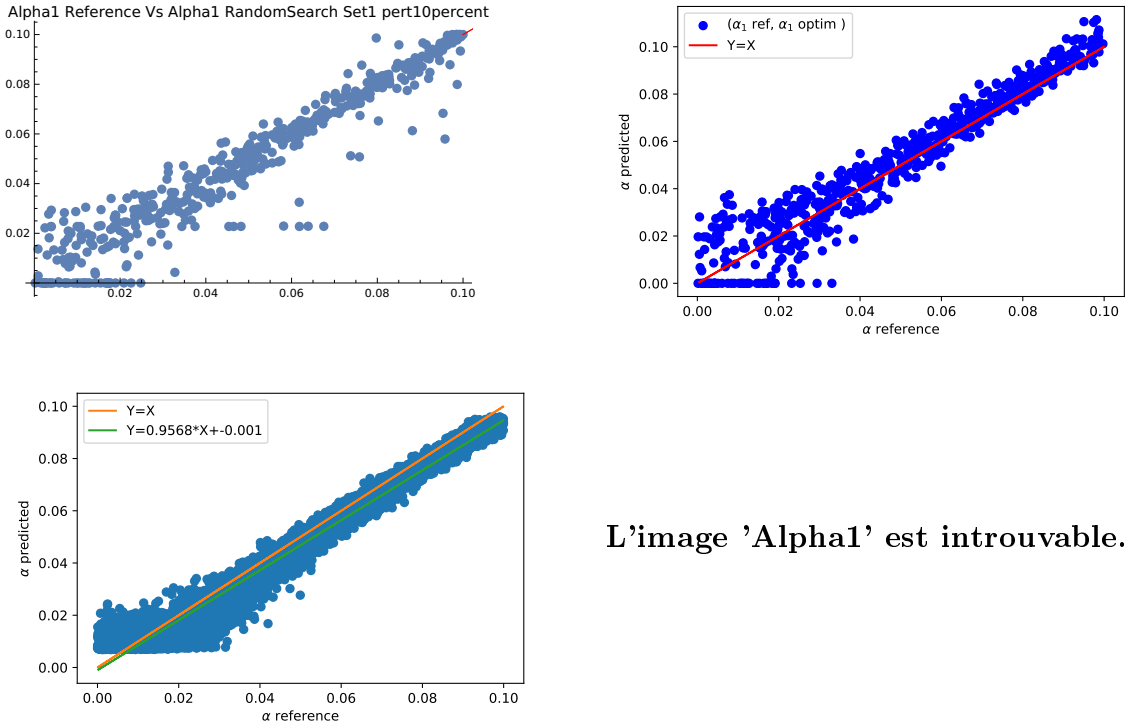
In figure 23, all models are trained (for the neural network model) or solved (for the direct minimization model) on the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) for a set of parameters (α_1, β_1) belonging to $set1 = [0, 0.1] \times [0, 0.7]$, and the standard deviation of the noise is taken such that: $std(\epsilon_1) \sim 5.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 5.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 5.10^{-2} std(\Gamma_6)$.



L'image 'Alpha1' est introuvable.

Figure 23: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using SLSQP algorithm on python; Down and left: Prediction of α_1 using Neural Network algorithm on python for the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) using $std(\epsilon_1) \sim 5.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 5.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 5.10^{-2} std(\Gamma_6)$ on the set of parameters $set1 = [0, 0.1] \times [0, 0.7]$.

In figure 24, all models are trained (for the neural network model) or solved (for the direct minimization model) on the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) for a set of parameters (α_1, β_1) belonging to $set1 = [0, 0.1] \times [0, 0.7]$, and the standard deviation of the noise is taken such that: $std(\epsilon_1) \sim 10^{-1} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 10^{-1} std(\Gamma_4)$ and $std(\epsilon_3) \sim 10^{-1} std(\Gamma_6)$.



L'image 'Alpha1' est introuvable.

Figure 24: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using SLSQP algorithm on python; Down and left: Prediction of α_1 using Neural Network algorithm on python for the perturbed training set ($\mathbb{E}[x^2] + \epsilon_1$,$\Gamma_4 + \epsilon_2$, $\Gamma_6 + \epsilon_3$) using $std(\epsilon_1) \sim 10^{-1} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 10^{-1} std(\Gamma_4)$ and $std(\epsilon_3) \sim 10^{-1} std(\Gamma_6)$ on the set of parameters $set1 = [0, 0.1] \times [0, 0.7]$.

Message 5: In the regular zone with perturbed data, the algorithms Random Search, SLSQP and Neural network have shown similar performances to solve the Garch constrained problem for the different magnitude of the noise.
The neural network stays also better in computational time as it takes less than 1 second to compute the result for 5000 parameters on GPU (once it is trained, the training session takes around 2-3 hours on GPU). The direct minimization algorithms take in this zone around 10 minutes to converge for only 500 parameters.

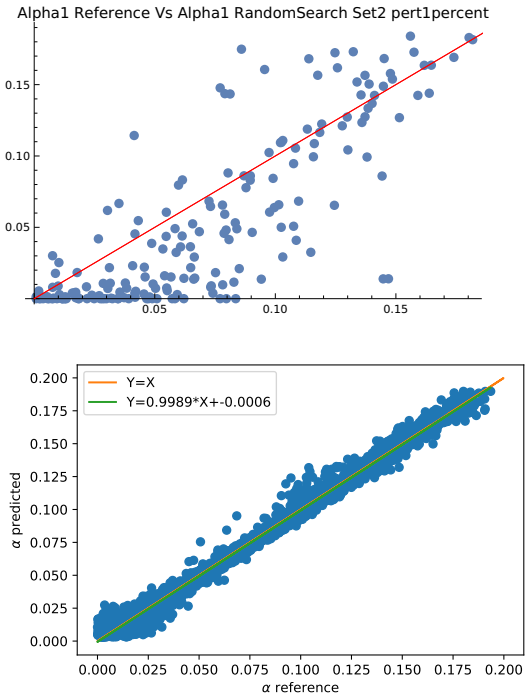
Perturbation Test cases on the second Set $set2 = [0, 0.3] \times [0.7, 1]$ using direct minimization and Neural network comparison:

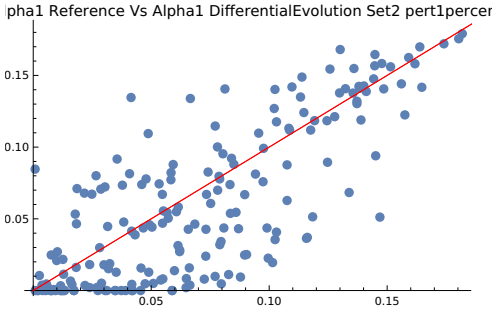
In this section, we investigate the performance of direct minimization algorithms compared to that of the neural network model in estimating the parameter α_1 by introducing noise into the data. We plot several figures showing different magnitudes of the standard deviation of the noise.

Let ϵ_1, ϵ_2, and ϵ_3 be random normal variables with mean zero and varying standard deviations across different tests, as depicted in the following figures.

All models, whether trained (for the neural network) or solved (for direct minimization), are applied to the perturbed training set $(\mathbb{E}[x^2]+\epsilon_1, \Gamma_4+\epsilon_2, \Gamma_6+\epsilon_3)$ for a set of parameters (α_1, β_1) belonging to $set1 = [0, 0.3] \times [0.7, 1]$.

In figure 25 the standard deviation of the noise is taken such that: $std(\epsilon_1) \sim 10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 10^{-2} std(\Gamma_6)$.

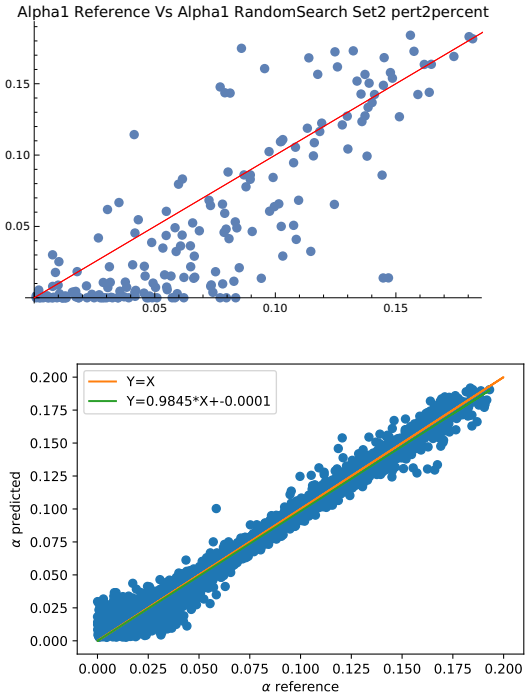


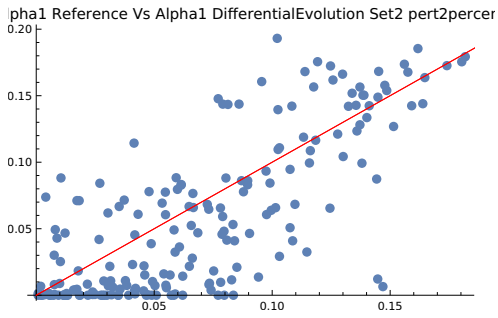


L'image 'aaa' est introuvable.

Figure 25: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using SLSQP algorithm on python; Down and left: Prediction of α_1 using Neural Network algorithm on python for the perturbed training set $(\mathbb{E}[x^2] + \epsilon_1, \Gamma_4 + \epsilon_2, \Gamma_6 + \epsilon_3)$ using $std(\epsilon_1) \sim 10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 10^{-2} std(\Gamma_6)$ on the set of parameters $set2 = [0, 0.3] \times [0.7, 1]$.

In figure 26, all models are trained (for the neural network model) or solved (for the direct minimization model) on the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) for a set of parameters (α_1, β_1) belonging to $set1 = [0, 0.1] \times [0, 0.7]$, and the standard deviation of the noise is taken such that: $std(\epsilon_1) \sim 2.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 2.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 2.10^{-2} std(\Gamma_6)$.





L'image 'Alpha1' est introuvable.

Figure 26: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using Differential Evolution algorithm on mathematica; Down and left: Prediction of α_1 using Neural Network algorithm on python for the perturbed training set ($\mathbb{E}[x^2] + \epsilon_1$,$\Gamma_4 + \epsilon_2$, $\Gamma_6 + \epsilon_3$) using $std(\epsilon_1) \sim 2.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 2.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 2.10^{-2} std(\Gamma_6)$ on the set of parameters $set2 = [0, 0.3] \times [0.7, 1]$.

Message 6: In the irregular zone (set2) with perturbed data, the Random Search, Differential Evolution, and Neural Network algorithms demonstrated similar performances in solving the GARCH constrained problem across different noise magnitudes. The neural network continues to excel in computational time, requiring less than 1 second to compute the result for 5000 parameters on a GPU (once trained, with the training session taking around 2-3 hours on the GPU). In contrast, direct minimization algorithms take approximately 10 minutes to converge for only 500 parameters in this zone.

Perturbation Test Cases on the Third Set $set3 = [0.1, 0.3] \times [0, 0.7]$: Comparison Between Direct Minimization and Neural Network

In figure 27, all models are trained (for the neural network model) or solved (for the direct minimization model) on the perturbed training set ($\mathbb{E}[x^2] + \epsilon_1$,$\Gamma_4 + \epsilon_2$, $\Gamma_6 + \epsilon_3$) for a set of parameters (α_1, β_1) belonging to $set3 = [0.1, 0.3] \times [0, 0.7]$, and the standard deviation of the noise is taken such that: $std(\epsilon_1) \sim 10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 10^{-2} std(\Gamma_6)$.

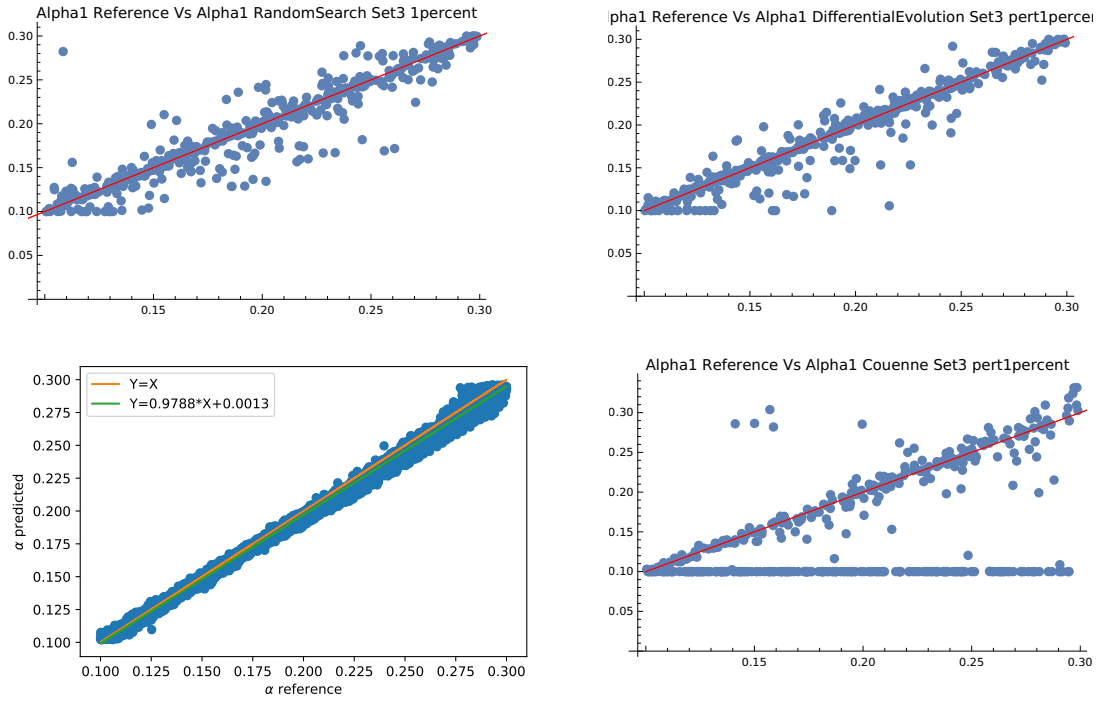


Figure 27: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using Differential Evolution algorithm on mathematica; Down and left: Prediction of α_1 using Neural Network algorithm on python; Down and right: Prediction of α_1 using Couenne algorithm on mathematica for the perturbed training set ($\mathbb{E}[x^2] + \epsilon_1$,$\Gamma_4 + \epsilon_2$, $\Gamma_6 + \epsilon_3$) using $std(\epsilon_1) \sim 10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 10^{-2} std(\Gamma_6)$ on the set of parameters $set3 = [0.1, 0.3] \times [0, 0.7]$.

In figure 28, all models are trained (for the neural network model) or solved (for the direct minimization model) on the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) for a set of parameters (α_1, β_1) belonging to $set3 = [0.1, 0.3] \times [0, 0.7]$, and the standard deviation of the noise is taken such that: $std(\epsilon_1) \sim 2.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 2.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 2.10^{-2} std(\Gamma_6)$.

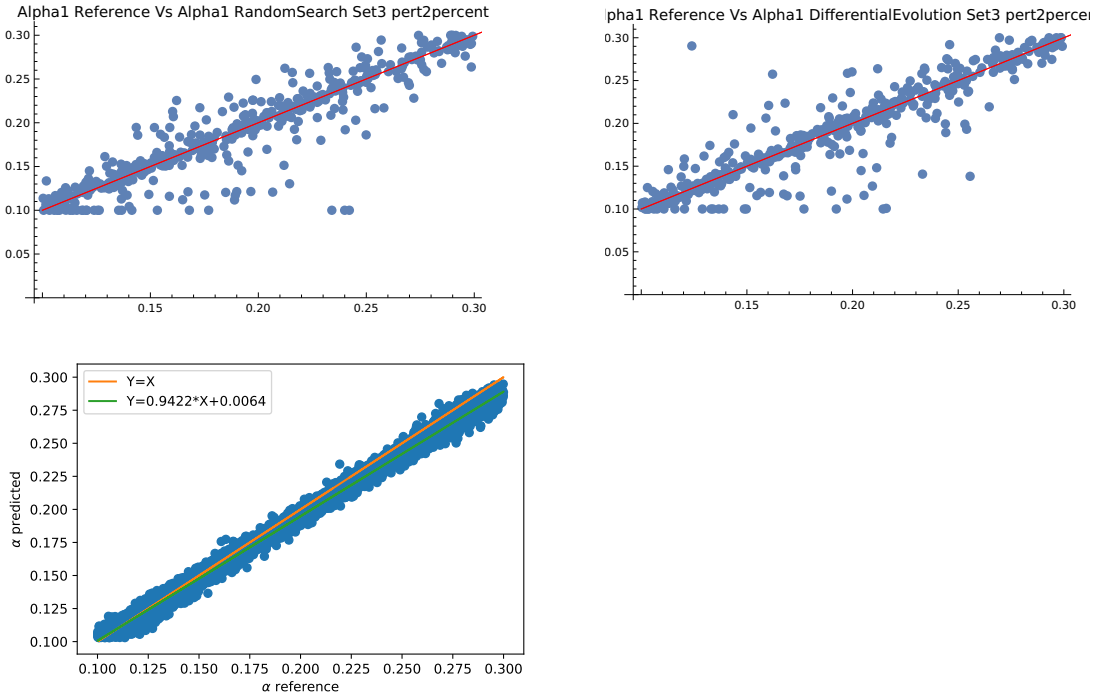


Figure 28: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using Differential Evolution algorithm on mathematica; Down and left: Prediction of α_1 using Neural Network algorithm on python; Down and right: Prediction of α_1 using Couenne algorithm on mathematica for the perturbed training set ($\mathbb{E}[x^2] + \epsilon_1$,$\Gamma_4 + \epsilon_2$, $\Gamma_6 + \epsilon_3$) using $std(\epsilon_1) \sim 2.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 2.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 2.10^{-2} std(\Gamma_6)$ on the set of parameters $set3 = [0.1, 0.3] \times [0, 0.7]$.

In figure 29, all models are trained (for the neural network model) or solved (for the direct minimization model) on the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) for a set of parameters (α_1, β_1) belonging to $set3 = [0.1, 0.3] \times [0, 0.7]$, and the standard deviation of the noise is taken such that: $std(\epsilon_1) \sim 3.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 3.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 3.10^{-2} std(\Gamma_6)$.

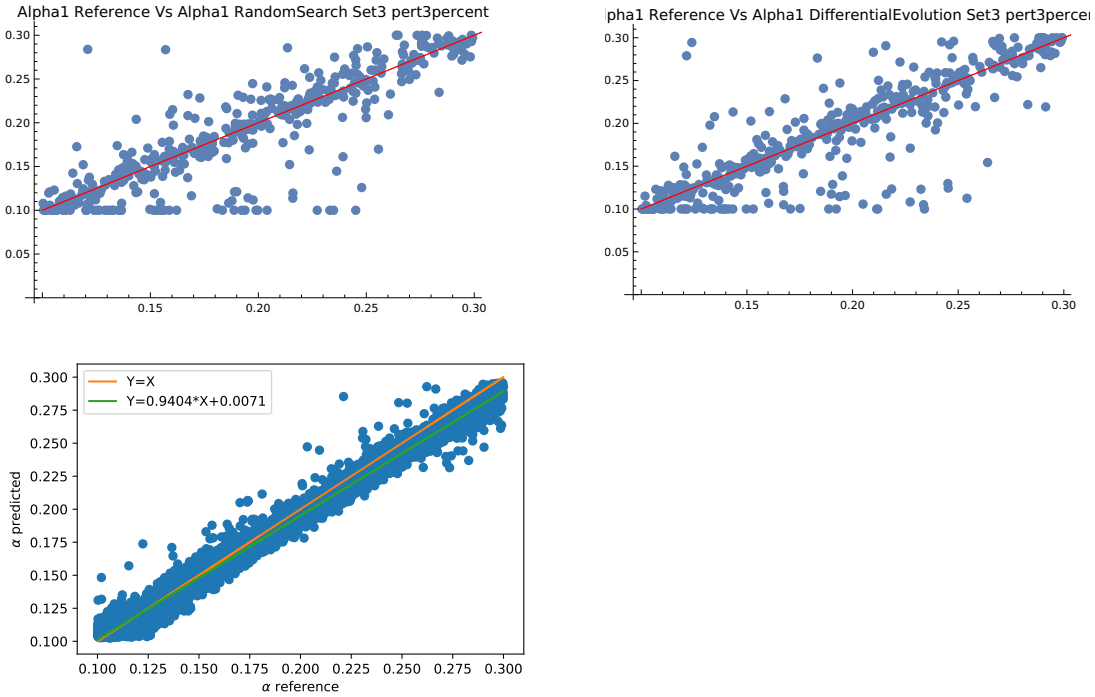


Figure 29: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using Differential Evolution algorithm on mathematica; Down and left: Prediction of α_1 using Neural Network algorithm on python; Down and right: Prediction of α_1 using Couenne algorithm on mathematica for the perturbed training set ($\mathbb{E}[x^2] + \epsilon_1$,$\Gamma_4 + \epsilon_2$, $\Gamma_6 + \epsilon_3$) using $std(\epsilon_1) \sim 3.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 3.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 3.10^{-2} std(\Gamma_6)$ on the set of parameters $set3 = [0.1, 0.3] \times [0, 0.7]$.

In figure 30, all models are trained (for the neural network model) or solved (for the direct minimization model) on the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) for a set of parameters (α_1, β_1) belonging to $set3 = [0.1, 0.3] \times [0, 0.7]$, and the standard deviation of the noise is taken such that: $std(\epsilon_1) \sim 5.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 5.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 5.10^{-2} std(\Gamma_6)$.

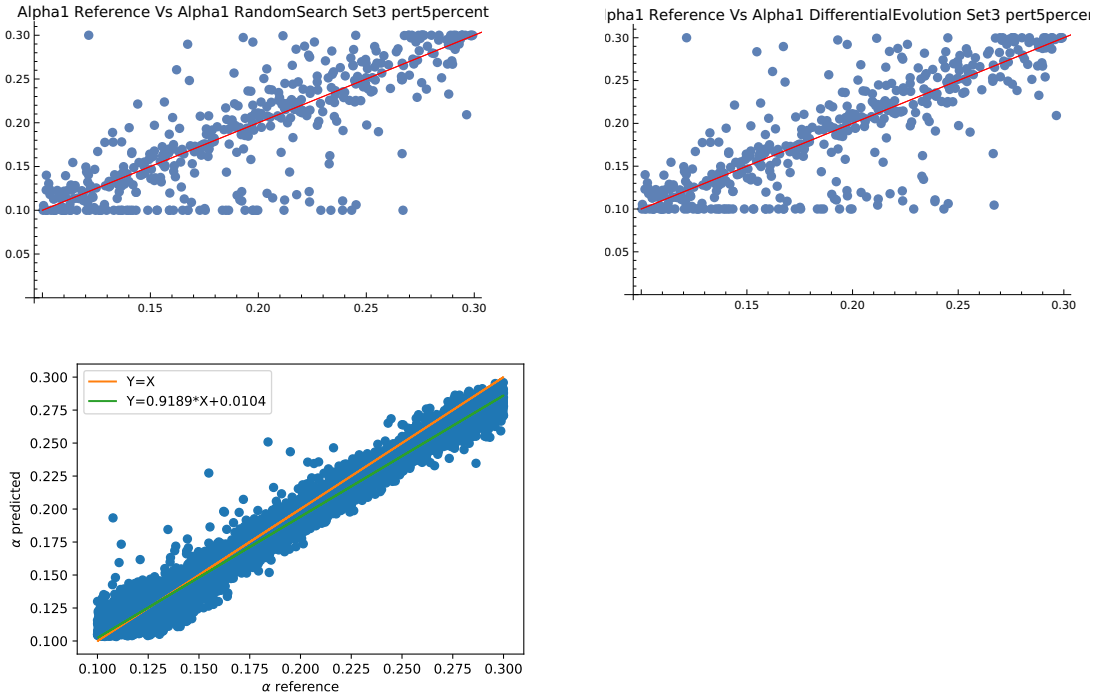


Figure 30: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using Differential Evolution algorithm on mathematica; Down and left: Prediction of α_1 using Neural Network algorithm on python; Down and right: Prediction of α_1 using Couenne algorithm on mathematica for the perturbed training set ($\mathbb{E}[x^2] + \epsilon_1$,$\Gamma_4 + \epsilon_2$, $\Gamma_6 + \epsilon_3$) using $std(\epsilon_1) \sim 5.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 5.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 5.10^{-2} std(\Gamma_6)$ on the set of parameters $set3 = [0.1, 0.3] \times [0, 0.7]$.

Message 7: In the irregular zone (set3) with perturbed data, the Random Search, Differential Evolution, and Couenne algorithms failed to solve the GARCH constrained problem across different noise magnitudes. However, the neural network demonstrated strong performance, with 95% of the predicted data falling within the region bounded by the lines $y = x + 0.1$ and $y = x - 0.1$, for a noise standard deviation of 5%. The neural network continues to outperform in computational efficiency, taking less than 1 second to compute results for 5000 parameters on a GPU.

Summary of the performance and computational time of the methods:

Method \ Precision	set1	set2	set3	set1 noised	set2 noised	set3 noised	computational time
Random Search	✓	×	✓	✓	×	×	$3-10$mins
Differential Evolution	×	×	✓	✓	×	×	$3-10$mins
Couenne	✓	×	✓	✓	×	×	$3-10$mins
SLSQP	✓	×	×	✓	×	×	2mins
Neural network	✓	✓	✓	✓	✓	✓	<1s

2.4 Statistical Data to Train the Network for the GARCH Parameters

Since the neural network method has shown good results on perturbed data, we now investigate its performance on time series data by using moment estimators, which will introduce statistical error assuming our model is ergodic. We will first examine the performance of a standard method, the maximum likelihood method, for determining the GARCH parameters, and then compare it with the results from the neural network.

2.4.1 Test Case Using Maximum Likelihood Method

Let us derive the probability function that we wish to maximize. We begin with the GARCH(1,1) model:

$$\sigma_t^2 = \alpha_0 + \alpha_1 x_{t-1}^2 + \beta_1 \sigma_{t-1}^2 \tag{16}$$

where the return is defined as:

$$x_t = z_t \sigma_t \tag{17}$$

and $z_t \sim \mathcal{N}(0,1)$.
Thus, conditionally on $\mathcal{F}_t$, we have:

$$x_t \sim \mathcal{N}(0, \sigma_t^2)$$

To perform maximum likelihood estimation, we are interested in the joint distribution:

$$f(x_t, x_{t-1}, x_{t-2}, \ldots, x_0, \theta)$$

where $\theta = (\alpha_0, \alpha_1, \beta_1)$ is the vector of parameters. We can write:

$$f(x_t, x_{t-1}, x_{t-2}, \ldots, x_0, \theta) = f(x_t, x_{t-1}, \ldots, x_1, \theta | x_0) f(x_0, \theta) \tag{18}$$

Iterating this, we obtain:

$$f(x_t, x_{t-1}, \ldots, x_0, \theta) = \prod_{i=1}^{t} f(x_i, \theta | x_{i-1}, \ldots, x_0) f(x_0, \theta) \tag{19}$$

Since $f(x_i, \theta | x_{i-1}, \ldots, x_0)$ is the density of a normal variable with variance σ_i^2, we have:

$$f(x_t, x_{t-1}, \ldots, x_0, \theta) = f(x_0, \theta) \prod_{i=1}^{t} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{x_i^2}{2\sigma_i^2}\right) \tag{20}$$

Finally, we take the logarithm to get the function to maximize:

$$\log(f(x_t, x_{t-1}, \ldots, x_0, \theta)) = \sum_{i=1}^{t} \left[\frac{1}{2}\left(-\log(2\pi) - \log(\sigma_i^2) - \frac{x_i^2}{2\sigma_i^2}\right)\right] \tag{21}$$

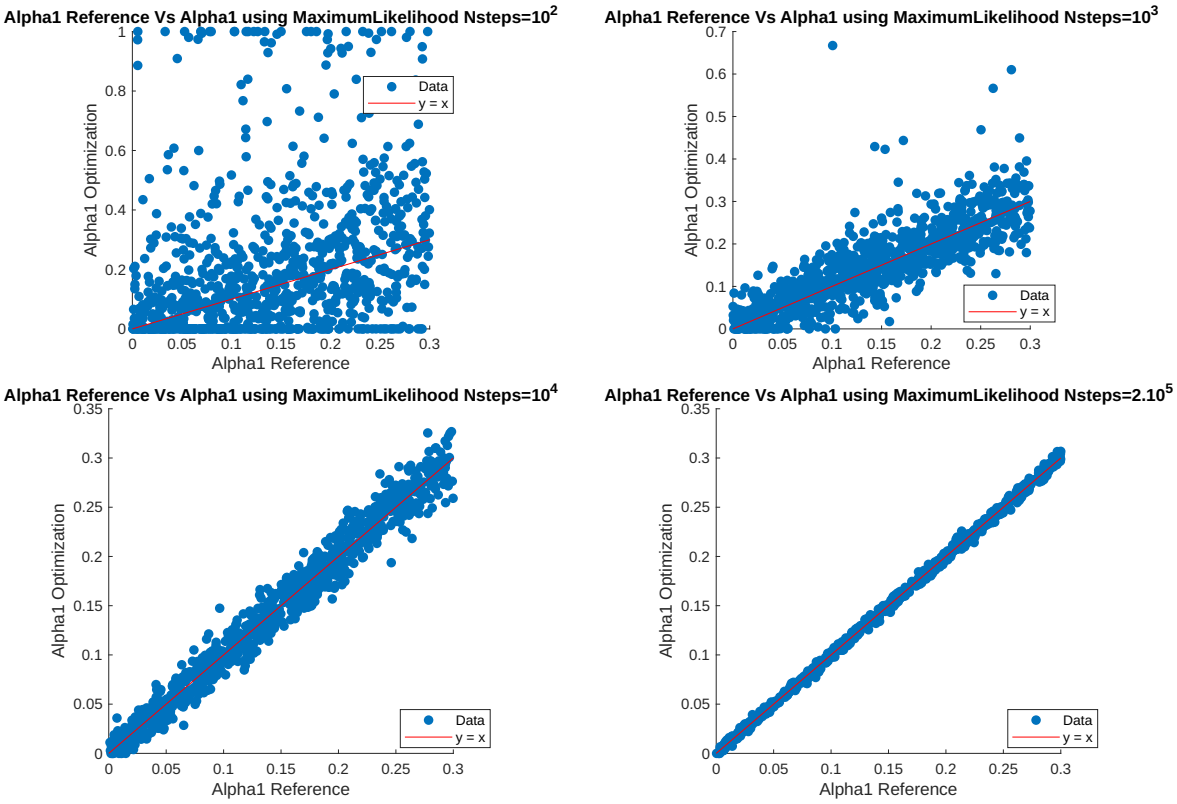


Figure 31: Up and left: Prediction of α_1 on a trajectory with total steps number $N = 10^2$; Up and right: Prediction of α_1 on a trajectory with total steps number $N = 10^3$; Down and left: Prediction of α_1 on a trajectory with total steps number $N = 10^4$; down and Right: Prediction of α_1 on a trajectory with total steps number $N = 2.10^5$.

Message 8: The previous results show that the maximum likelihood method gives very good results for a sample number N higher than 2.10^5, and it takes up to 10 minutes to solve the problem for a number of parameters equal to 500.

2.4.2 Some tests on the statistical error on ($\mathbb{E}[x^2]$, Γ_4, Γ_6) depending on T and N

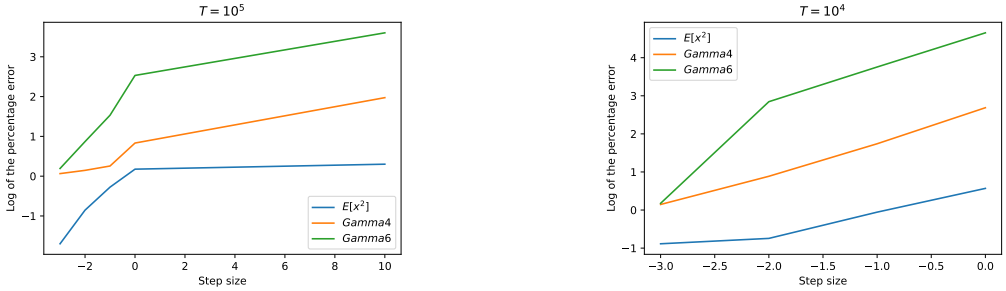


Figure 32: On the left: the logarithmic error of the percentage error on ($\mathbb{E}[x^2]$, Γ_4, Γ_6) for each time step with final time $T = 10^5$; On the right: same thing but with final time $T = 10^4$.

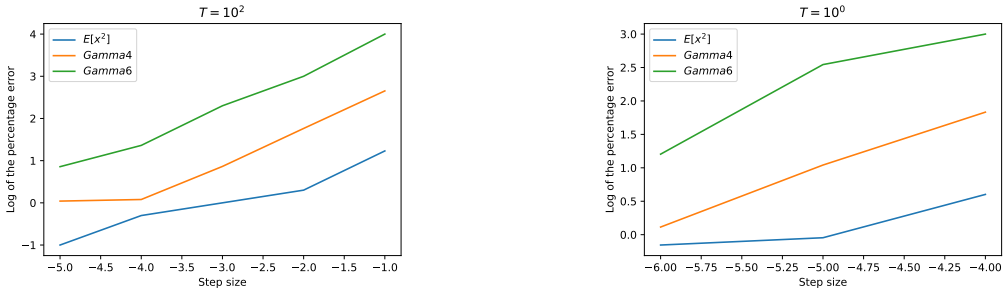


Figure 33: On the left: the logarithmic error of the percentage error on ($\mathbb{E}[x^2]$, Γ_4, Γ_6) for each time step with final time $T = 10^2$; On the right: same thing but with final time $T = 10^0$.

Message 9: The previous results on the statistical error in the training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) indicate that we require a sampling size greater than 10^6 on the trajectory to validate ergodicity, at least for the first moments. This is necessary to ensure a statistical relative error of less than 10% for $T = 1$.

Time period	N	Relative Error
$T = 1$	10^6	≤ 10 percent
$T = 10^2$	10^7	≤ 10 percent
$T = 10^4$	10^7	≤ 1 percent
$T = 10^5$	10^7	≤ 1 percent

2.4.3 Test Case Using a Number of Steps N Under Observed Ergodicity for $\mathbb{E}[x^2]$, $\mathbb{E}[x^4]$, and $\mathbb{E}[x^6]$

It is proven in [3] that the GARCH model has a unique stationary and ergodic solution under the condition $\alpha_1 + \beta_1 < 1$. In this section, we investigate the performance of our neural network on a time series by constructing estimators $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ for the quantities $(\mathbb{E}[x^2], \Gamma_4, \Gamma_6)$.

As discussed in the previous section, we can assume that our trajectory is ergodic for certain test functions, such as x^2, x^4, and x^6. Therefore, we leverage the ergodicity principle along the trajectory to create our estimators. This approach provides insights into the neural network's ability to capture and replicate statistical properties throughout the time series.

We recall that in an ergodic system with an invariant measure p_∞, we have:

$$\lim_{T\to\infty} \frac{1}{T} \int_0^T \Phi(x_s)\, ds = \mathbb{E}_{p_\infty}[\Phi(x)],$$

which can be approximated using the trajectory of $(x_t)_{t\in[0,T]}$ as:

$$\mathbb{E}_N[x] = \frac{1}{N} \sum_{i=1}^{N} x_i,$$

where $(x_i)_{0\le i\le N}$ are points along the trajectory. Thus, we use the following estimators:

$$\mathbb{E}_N[x^2] = \frac{1}{N} \sum_{i=1}^{N} x_i^2, \quad \Gamma_4^N = \frac{\mathbb{E}_N[x^4]}{(\mathbb{E}_N[x^2])^2}, \quad \Gamma_6^N = \frac{\mathbb{E}_N[x^6]}{(\mathbb{E}_N[x^2])^3}.$$

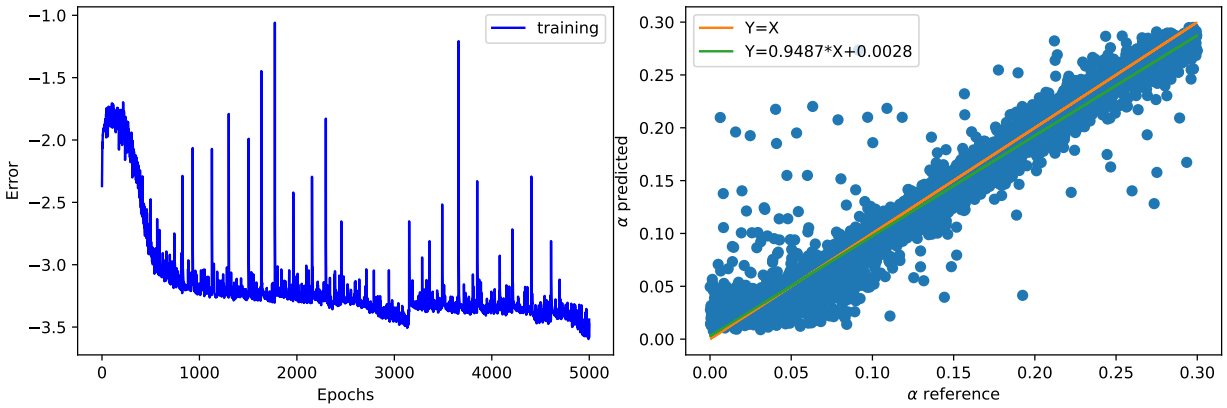


Figure 34: Left: Training error for the training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ using $N = 10^5$.

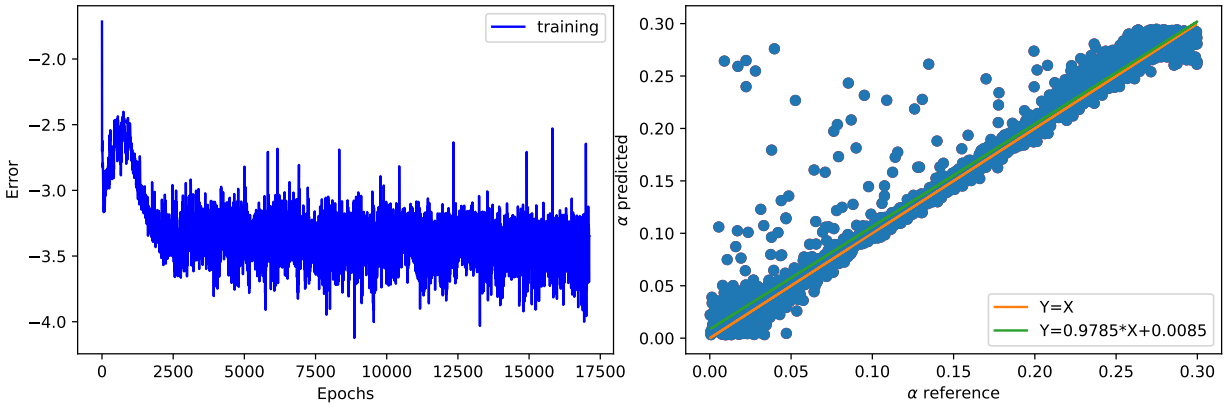


Figure 35: Left: Training error for the training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ using $N = 10^6$.

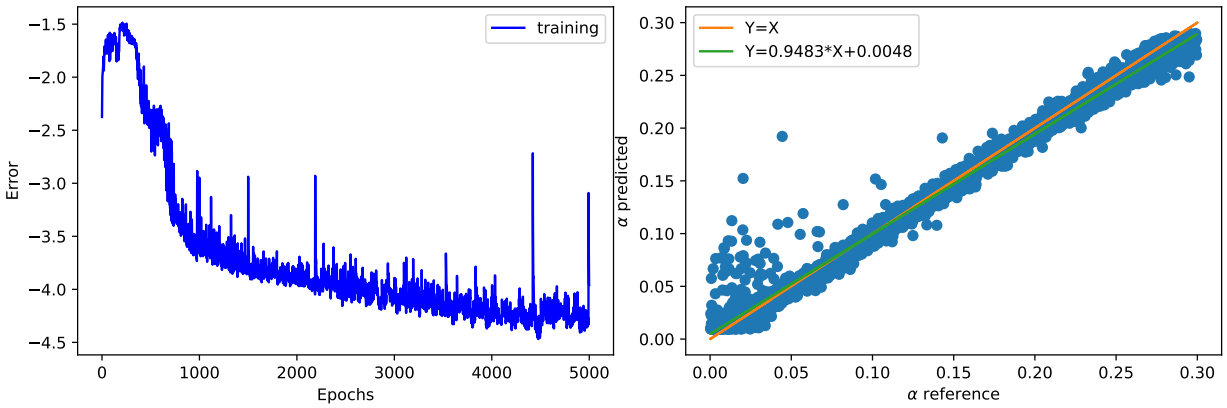


Figure 36: Left: Training error for the training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ using $N = 10^7$.

Message 10: The previous results, obtained from training data collected from the estimators $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ of $(\mathbb{E}[x^2], \Gamma_4, \Gamma_6)$, have demonstrated excellent performance on the time series. As expected, increasing the number of samples along the trajectory reduces the statistical error, leading to improved results. The neural network model exhibits remarkable effectiveness on the trajectory, comparable to the performance achieved by the maximum likelihood method. Notably, the computational efficiency of the neural network model is significantly better, highlighting its practical advantages in terms of speed.

2.4.4 Full GARCH Parameters from the Neural Network under Loss Penalization

In section **??**, we observed that once we obtain the parameter α_1, we can deduce the parameters α_0 and β_1 using equations (**??**) and (**??**) with the empirical estimators Γ_4 and Γ_6. However, this method has yielded poor results for estimating α_0 and β_1. This is due to the error that appears in the denominator term for Γ_4 in (**??**) when estimating β_1, which subsequently propagates to α_0 through (**??**). To address this issue, we propose a neural network that outputs all three parameters simultaneously.

Neural Network Architecture

The neural network developed in this section is similar to the previous one. It consists of four layers with $(1280, 2048, 4096, 1280)$ nodes, respectively, and uses

a sinusoidal activation function. Additionally, a sigmoid activation function is applied to the output layer to ensure that the resulting parameters lie in the range $[0, 1]$. The Adam optimizer is used with a learning rate of 10^{-5}.

Loss Function with a Penalization Term

An important aspect not previously discussed is whether our final neural network ensures the condition $\alpha_1 + \beta_1 < 1$. This condition is not guaranteed by default. To address this, we introduce a penalization term in the loss function, acting as a Lagrange multiplier or regularization term:

$$\lambda \max(0, (\alpha_1 + \beta_1 - 1))$$

This penalization discourages outputs that violate the condition $\alpha_1 + \beta_1 < 1$. The Lagrange factor, λ, was chosen to be 0.1 as it showed better results for both penalization and neural network performance. Although higher values for the Lagrange factor are typically preferred, in our case, using a larger value negatively affected the results.

Results

In Figure 37, we plot the results for α_0, α_1, and β_1 predicted by the neural network, comparing them to the exact values. The neural network was trained on the dataset $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ with $N = 10^7$ and $T = 1$.

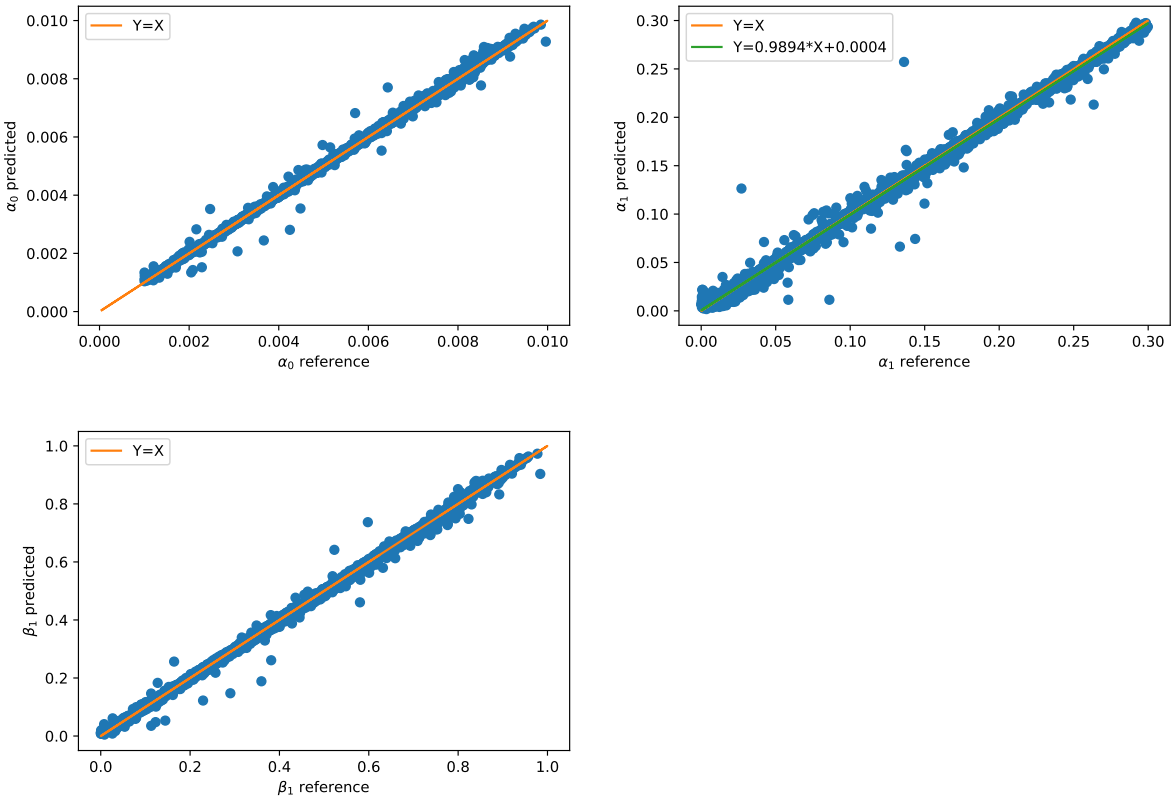


Figure 37: Up and Left:Prediction of α_0; Up and Right: Prediction of α_1 ;down and left: Prediction of β_1 using Neural Network algorithm on python for the empirical training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ with $N = 10^7$ and $T = 1$.

2.4.5 Implied volatility as a performance measure:

Application using Geometric Brownian model:

Assume that our stock follows the Black and Scholes dynamic,

$$S_{t+\Delta t} = S_t e^{(r-\frac{\sigma^2}{2})\Delta t+\sigma\sqrt{\Delta t}Z_t}, \tag{22}$$

assume that the volatility in the previous equation is time dependent following the Garch approximation,

$$\sigma^2_{t+\Delta t} = \alpha_0 + \alpha_1 x_t^2 + \beta_1 \sigma_t^2, \tag{23}$$

then we can write the stock price as

$$S_{t+\Delta t} = S_t e^{(r-\frac{\sigma_t^2}{2})\Delta t+\sigma_t\sqrt{\Delta t}Z_t}, \tag{24}$$

hence on $[0, T]$ we can write the stock price as,

$$S_T = S_0 \exp\left(rT - 0.5 \sum_{t=0}^{N_T} \sigma_t^2 \Delta t + \sum_{t=0}^{N_T} \sigma_t \sqrt{\Delta t} Z_t \right), \tag{25}$$

We then compute the call price $c(0)$ using a Monte Carlo simulation,

$$c(0) = e^{(-rT)} \mathbb{E}[(S_T - K)^+] \simeq \frac{1}{M} \sum_{i=1}^{M} (S_T^i - K)^+ \tag{26}$$

The implied volatility is then obtained by inverting the Black scholes call price and knowing the market call price that we approximate here by equation (26),

$$c(0) = \mathbb{E}[S_T] N(d_1) - K e^{-rT} N(d_2) \tag{27}$$

where,

$$d_1 = \frac{log(\mathbb{E}[S_T]/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}, \quad \text{and} \quad d_2 = d_1 - \sigma\sqrt{T} \tag{28}$$

We present in the following figure the implied volatility obtained by inverting equation 27 to obtain σ, the blue curve using a GARCH parameters $(\alpha_0, \alpha_1, \beta_1) = (0.007, 0.09, 0.8)$ and the orange curve using a GARCH parameters obtained from the neural network procedure $(\alpha_0, \alpha_1, \beta_1) = (0.0071, 0.092, 0.798)$, for different maturities $T = 1/12$, $T = 1/2$, $T = 1$ and $T = 10$. we obtain a relative error equal to 1 per cent in average on the implied volatility.

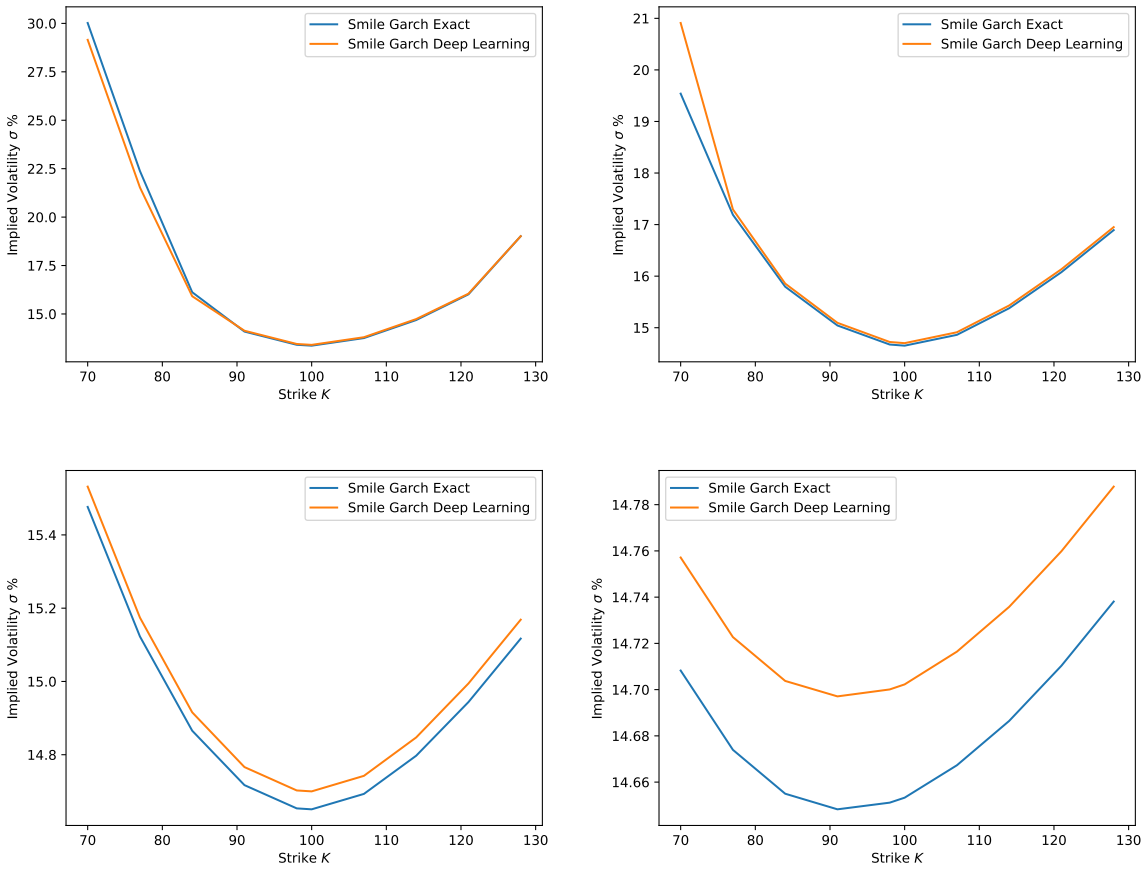


Figure 38: Up and left: Smile curve for $T = 1/12$; Up and right: Smile curve for $T = 1/2$; Down and left: Smile curve for $T = 1$; down and Right: Smile curve for $T = 10$.

3 Appendix

3.1 Extended results for the perturbed cases

Test case using Neural network with a normal perturbation of the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6):

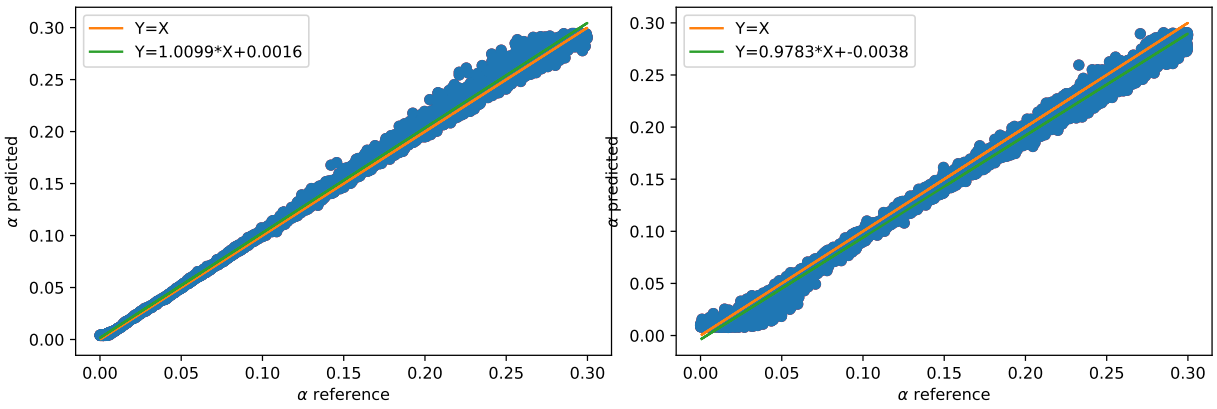


Figure 39: Left:Prediction of α_1 for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6); Right: Prediction of α_1 for the perturbated training set ($\mathbb{E}[x^2]+\epsilon_1$, $\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) using $std(\epsilon_1) \sim 10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 10^{-2} std(\Gamma_6)$.

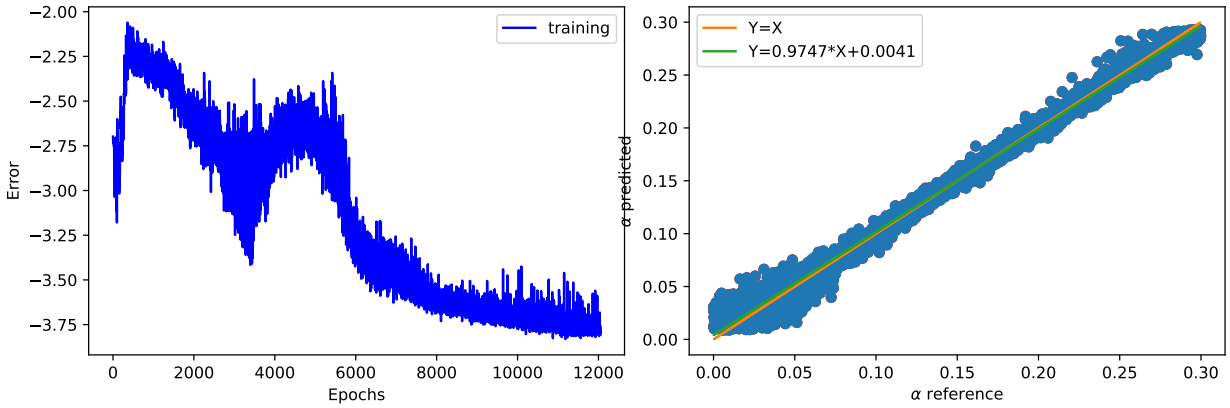


Figure 40: Left:Training error for the training set $(\mathbb{E}[x^2] + \epsilon_1,\ \Gamma_4 + \epsilon_2,\ \Gamma_6 + \epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbed training set $(\mathbb{E}[x^2] + \epsilon_1,\ \Gamma_4 + \epsilon_2,\ \Gamma_6 + \epsilon_3)$ using $std(\epsilon_1) \sim 2.10{-}2std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 2.10{-}2std(\Gamma_4)$ and $std(\epsilon_3) \sim 2.10{-}2std(\Gamma_6)$.

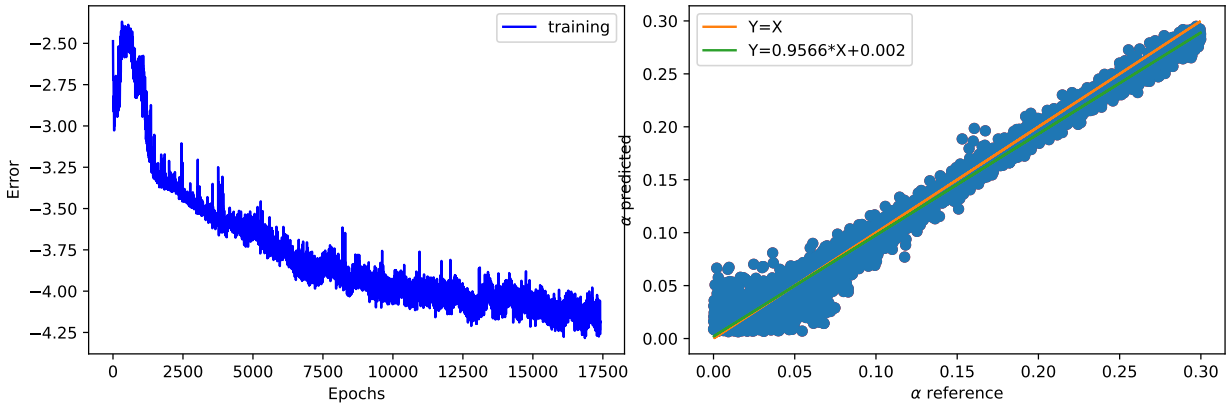


Figure 41: Left:Training error for the training set $(\mathbb{E}[x^2] + \epsilon_1,\ \Gamma_4 + \epsilon_2,\ \Gamma_6 + \epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2] + \epsilon_1,\ \Gamma_4 + \epsilon_2,\ \Gamma_6 + \epsilon_3)$ using $std(\epsilon_1) \sim 3.10^{-2}std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 3.10^{-2}std(\Gamma_4)$ and $std(\epsilon_3) \sim 3.10^{-2}std(\Gamma_6)$.

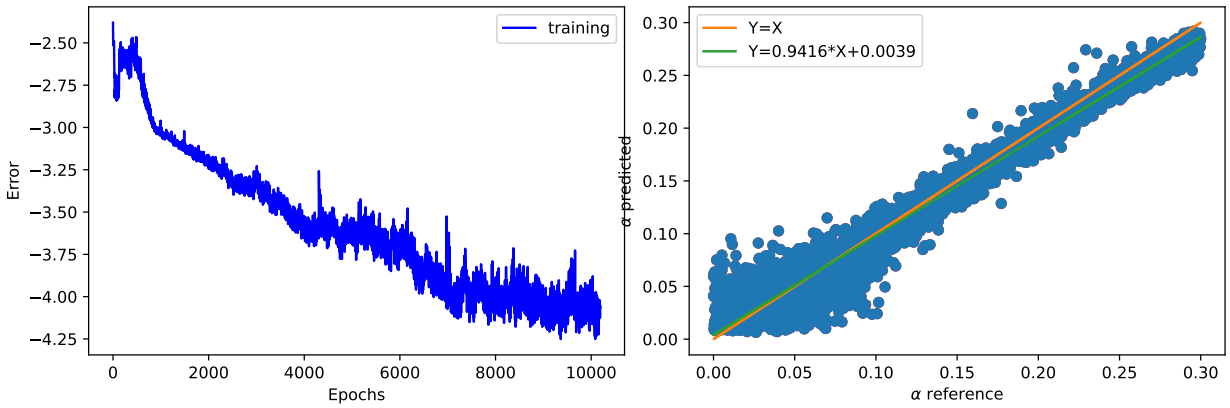


Figure 42: Left:Training error for the training set $(\mathbb{E}[x^2] + \epsilon_1,\ \Gamma_4 + \epsilon_2,\ \Gamma_6 + \epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2] + \epsilon_1, \Gamma_4 + \epsilon_2, \Gamma_6 + \epsilon_3)$ using $std(\epsilon_1) \sim 5.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 5.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 5.10^{-2} std(\Gamma_6)$.

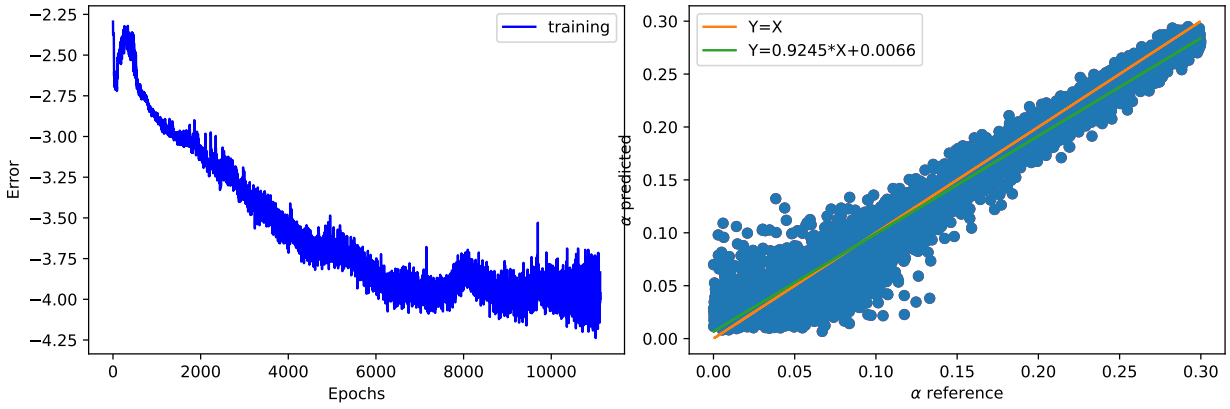


Figure 43: Left:Training error for the training set $(\mathbb{E}[x^2] + \epsilon_1,\ \Gamma_4 + \epsilon_2,\ \Gamma_6 + \epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2] + \epsilon_1, \Gamma_4 + \epsilon_2, \Gamma_6 + \epsilon_3)$ using $std(\epsilon_1) \sim 8.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 8.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 8.10^{-2} std(\Gamma_6)$.

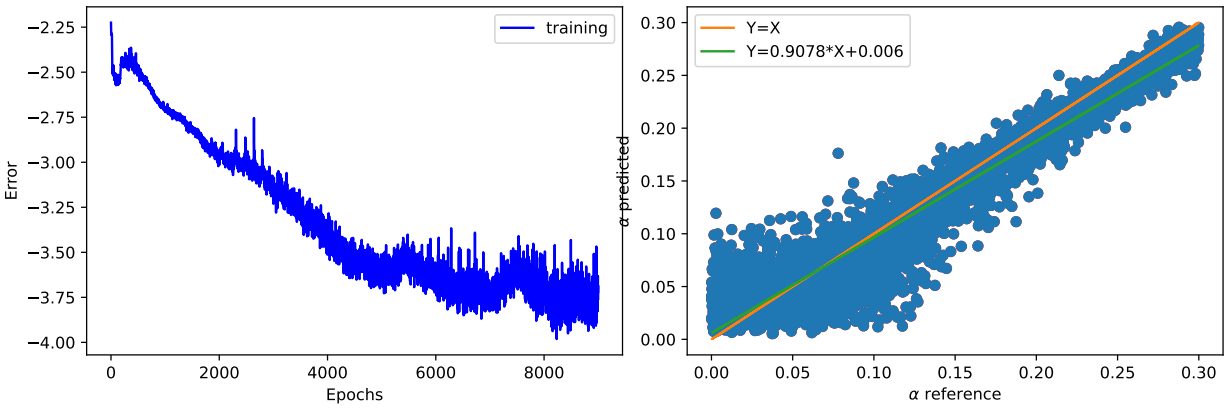


Figure 44: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1, \Gamma_4+\epsilon_2, \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1, \Gamma_4+\epsilon_2, \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 10^{-1} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 10^{-1} std(\Gamma_4)$ and $std(\epsilon_3) \sim 10^{-1} std(\Gamma_6)$.

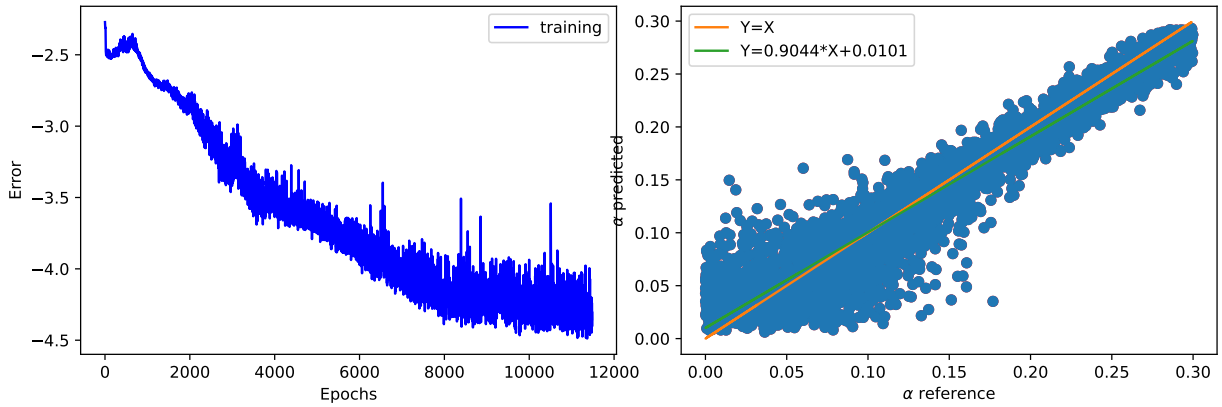


Figure 45: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1, \Gamma_4+\epsilon_2, \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1, \Gamma_4+\epsilon_2, \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 12.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 12.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 12.10^{-2} std(\Gamma_6)$.

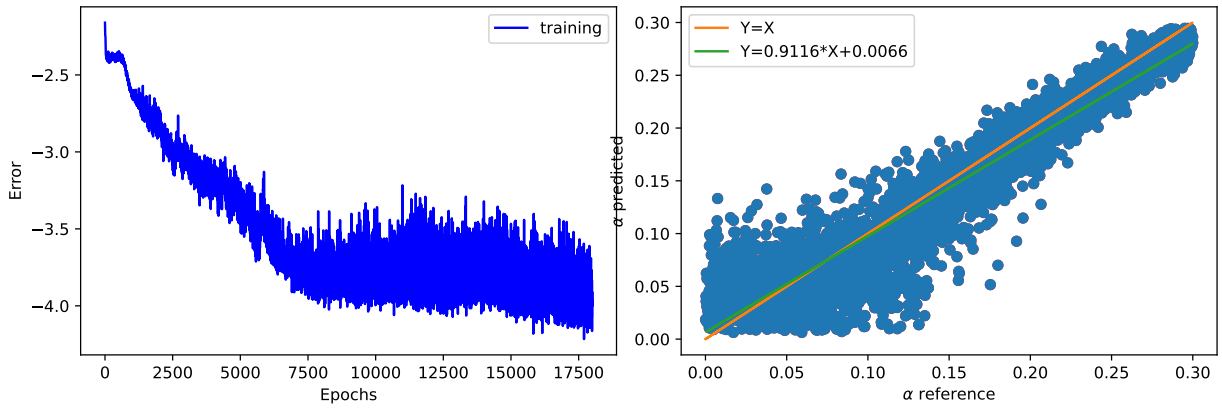


Figure 46: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 14.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 14.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 14.10^{-2} std(\Gamma_6)$.

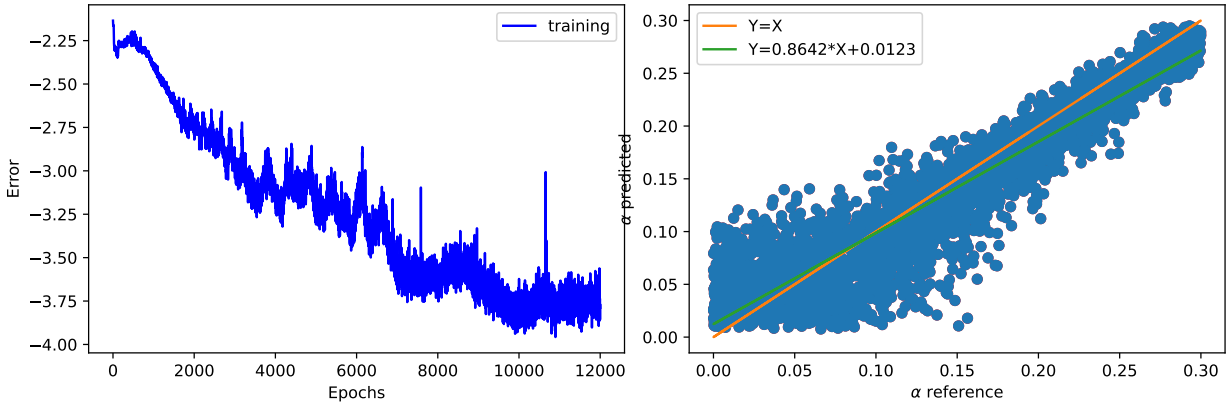


Figure 47: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 16.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 16.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 16.10^{-2} std(\Gamma_6)$.

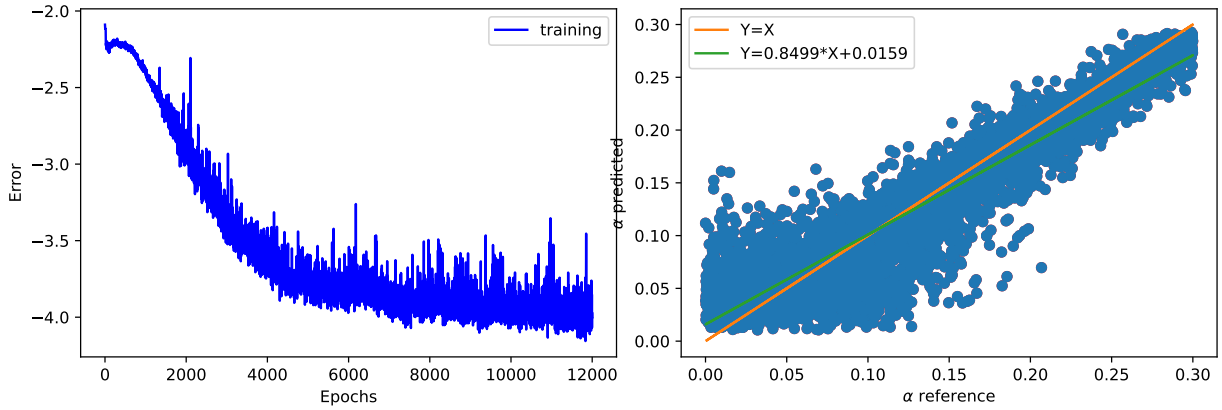


Figure 48: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 18.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 18.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 18.10^{-2} std(\Gamma_6)$.

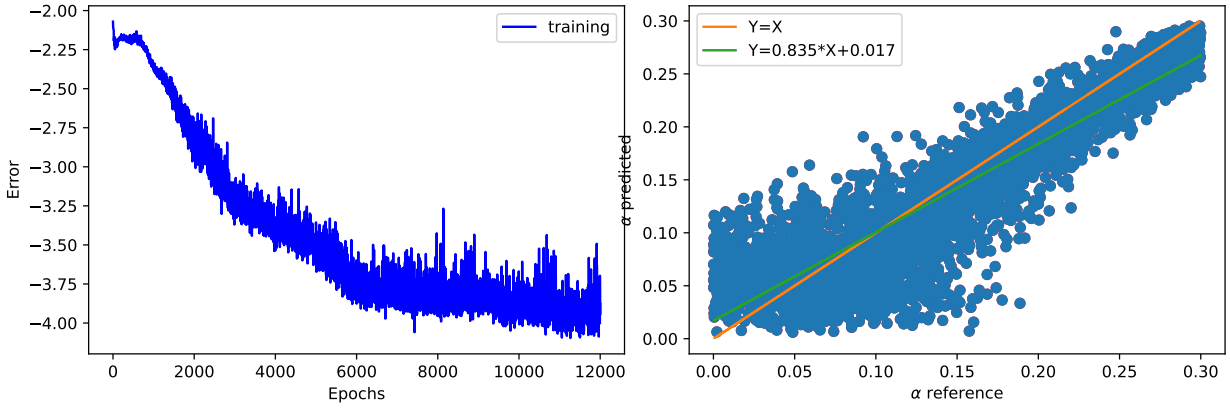


Figure 49: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 20.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 20.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 20.10^{-2} std(\Gamma_6)$.

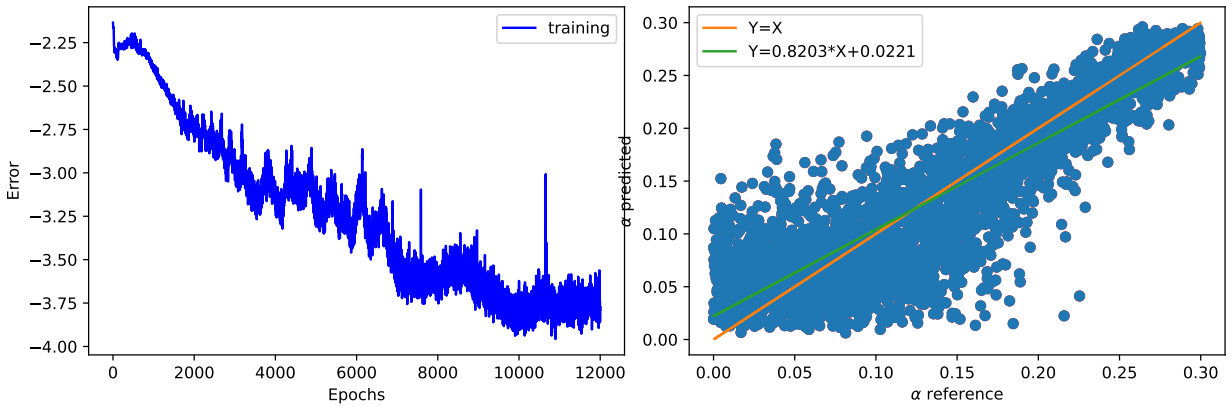


Figure 50: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1, \Gamma_4+\epsilon_2, \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 24.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 24.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 24.10^{-2} std(\Gamma_6)$.

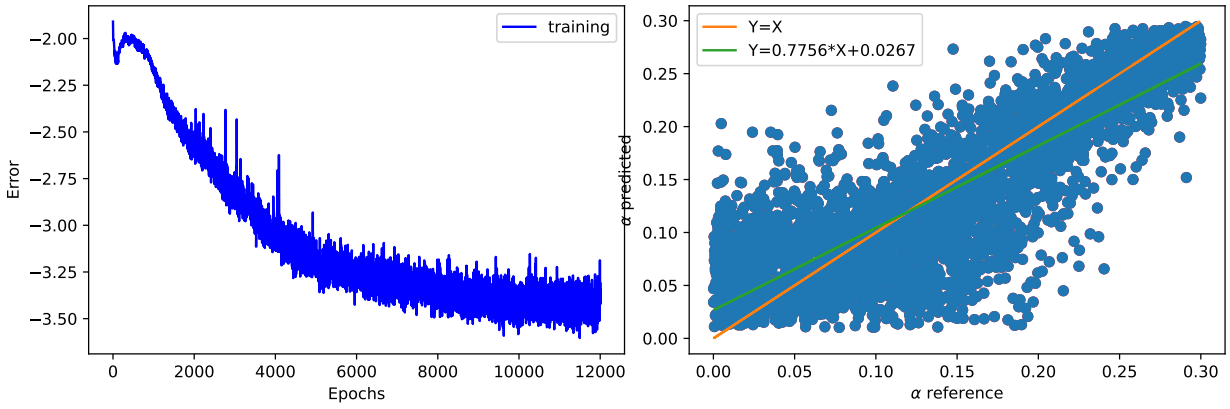


Figure 51: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1, \Gamma_4+\epsilon_2, \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 30.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 30.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 30.10^{-2} std(\Gamma_6)$.

Test case using autocovariance estimator $\hat{\gamma}_n^N$ for the training set:

In this part we investigate a training data set formed by $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \hat{\gamma}_6^N)$ where

$$\hat{\gamma}_n^N = \frac{\gamma_n^N}{\mathbb{E}_N[x^2]}$$

as an estimator of

$$\hat{\gamma}_n = \frac{\gamma_n}{\mathbb{E}[x^2]}$$

where

$$\gamma_n^N = Cov_n^N(x_t, x_{t+n}) = \mathbb{E}_N[xx_n] - \mathbb{E}_N[x]\mathbb{E}_N[x_n]$$

Hence we construct different processes $(x_t)_{t\in[0,T]}$ given by a set of parameters $(\alpha_0, \alpha_1, \beta_1)$ and an independent trajectories $(Z_t)_{t\in[0,T]}$ formed by normal independent increments and using the expression:

$$\sigma_t^2 = \alpha_0 + \alpha_1 x_{t-1}^2 + \beta_1 \sigma_{t-1}^2 \tag{29}$$

where $x_t = \sigma_t Z_t$

For all tests the dynamical learning rate starts from $lr = 0.1$ and decreasing by 10 percent each 100 epochs.

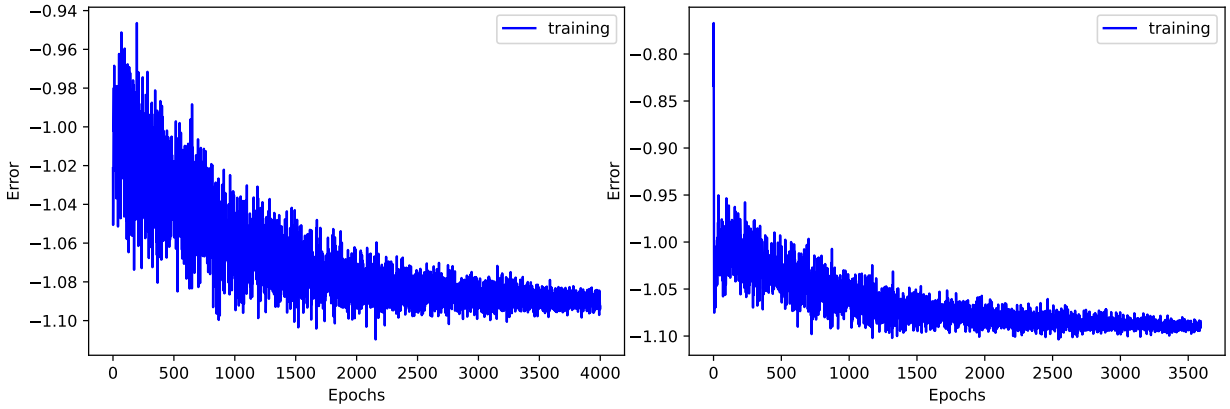


Figure 52: Left: Training error for the training set $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \hat{\gamma}_6^N)$ in logarithmic scale using dynamic learning rate and using 128 neurons in the first layer; Right: Training error for the same set using 1280 neurons in the first layer. Dynamic learning rate.

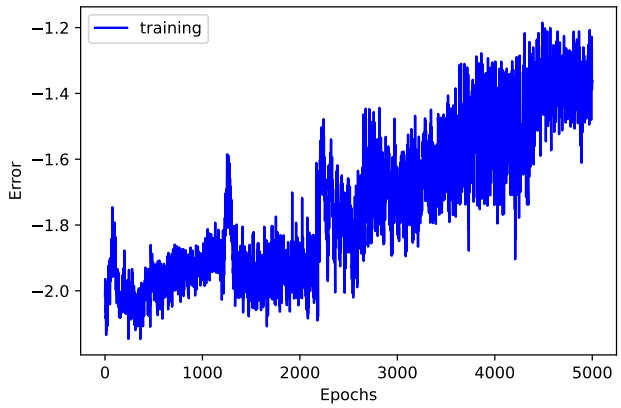


Figure 53: Training error for the training set $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \hat{\gamma}_6^N)$ in logarithmic scale using dynamic learning rate and using 128 neurons in the first layer. Fixed learning rate $lr = 10^{-4}$.

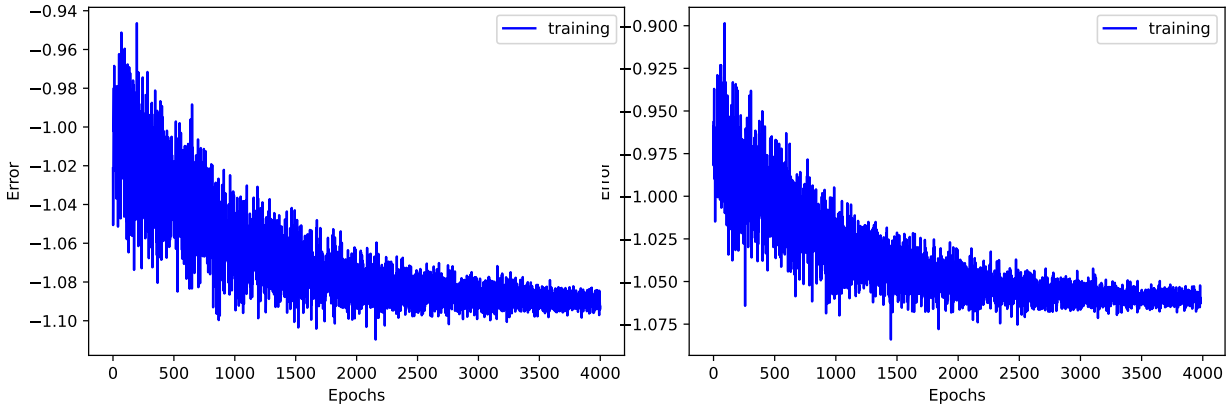


Figure 54: Left: Training error for the training set $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \hat{\gamma}_6^N)$ in logarithmic scale using dynamic learning rate and using 1280 neurons in the first layer; Right: Training error for the same neural network using the training set "Exact" $(\hat{\gamma}_2, \hat{\gamma}_4, \hat{\gamma}_6)$. Dynamic learning rate.

Test case using L-moments and autocovariance estimator $\hat{\gamma}_n^N$ for the training set:

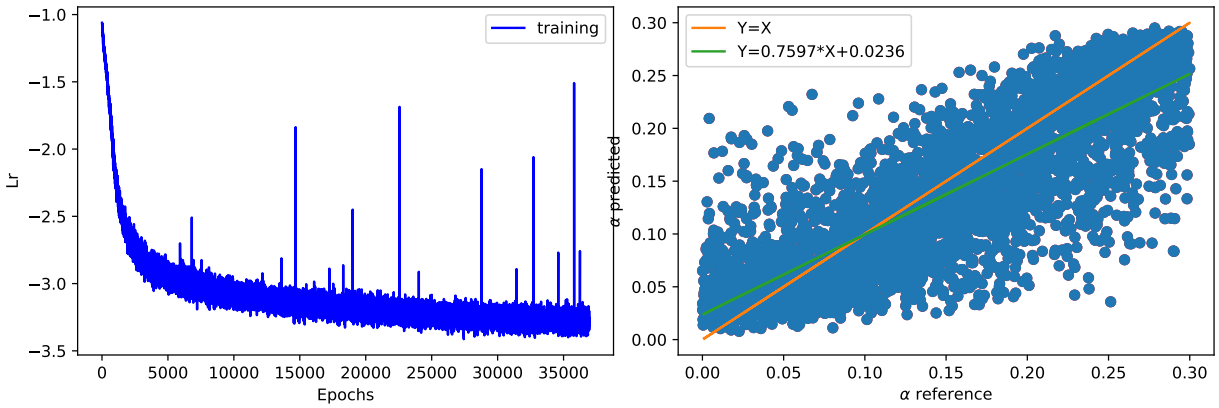


Figure 55: Left: Training error for the training set $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \hat{\gamma}_6^N, \hat{\gamma}_8^N, \lambda_1, \lambda_2, \tau_3, \tau_4, \tau_5)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 compared to α_1 of the reference. We use here 5.10^4 data.

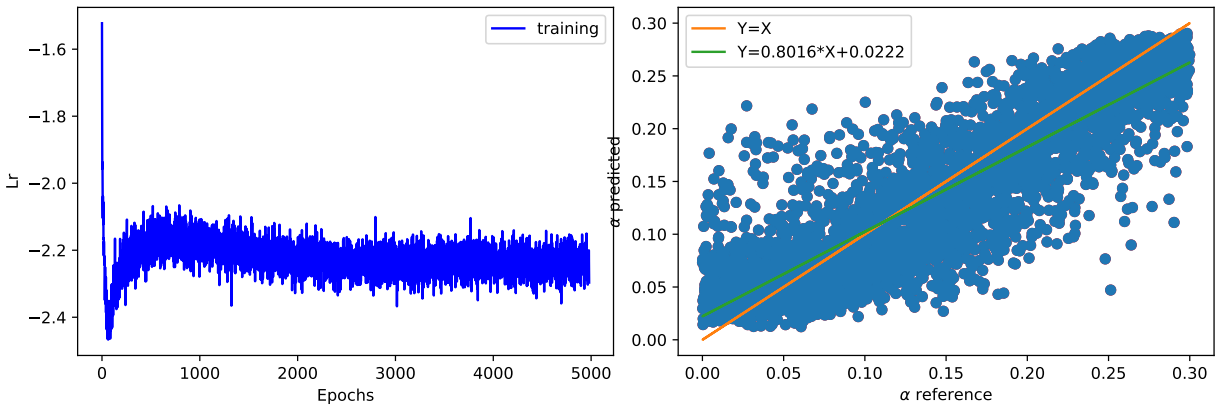


Figure 56: Left: Training error for the training set $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \tau_3, \tau_4, \tau_5)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 compared to α_1 of the reference. We use here 5.10^4 data.

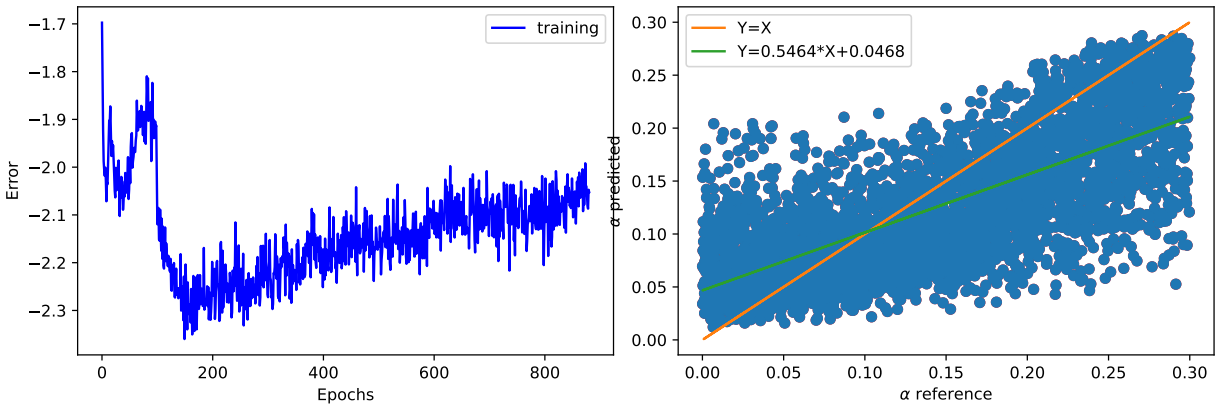


Figure 57: Left: Training error for the training set $(\hat{\gamma}_6^N, \hat{\gamma}_8^N, \tau_3, \tau_4, \tau_5)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 compared to α_1 of the reference. We use here 5.10^4 data.

3.2 A Convolutional Network Using Dynamic Learning Rate

As discussed in section 2.2.3, we propose using a dynamic learning rate for the generator by employing two neural networks, one encapsulated within the other. The first network is the generator that adjusts α_1, while the second network, referred to as the "decision maker," is responsible for learning and selecting the optimal learning rate.

The decision maker is an unsupervised neural network consisting of a linear layer with 4 neurons, followed by a softmax activation function. The softmax output represents a set of probabilities corresponding to each neuron index, where each index is associated with one of four learning rates. The goal is to train this neural network to select the learning rate that has the highest probability of leading to better performance. To achieve this, we define a reward as:

$$reward = -generator_{loss}$$

Thus, the decision maker's loss function is defined as:

$$DecisionMaker_{Loss} = -\log(p) \cdot reward$$

where p represents the probability of the chosen learning rate. The decision maker's loss is computed as the negative expected reward, encouraging the network to select learning rates that lead to higher rewards (better performance).

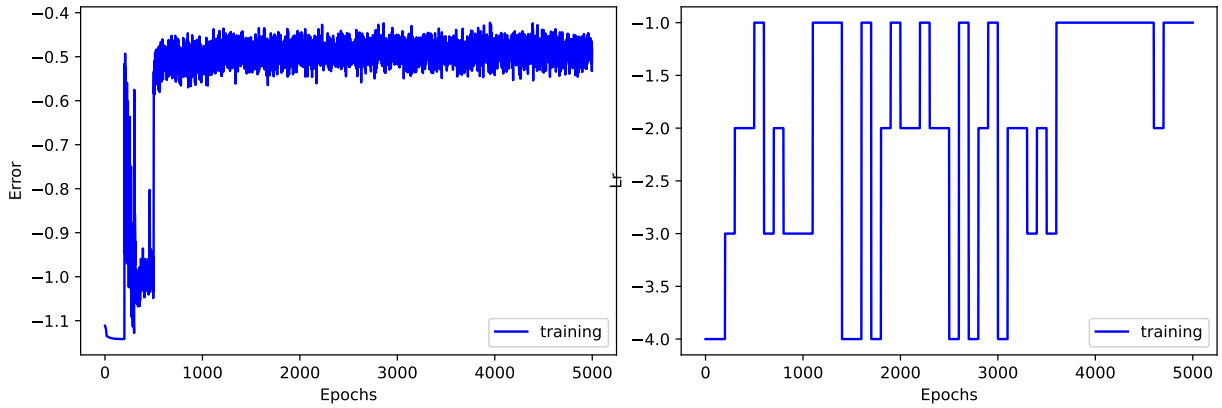


Figure 58: Left: Training error for the training set $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \hat{\gamma}_6^N)$ in logarithmic scale using dynamic learning rate and using 1280 neurons in the first layer; Right: Evolution of the learning rate during the learning phase.

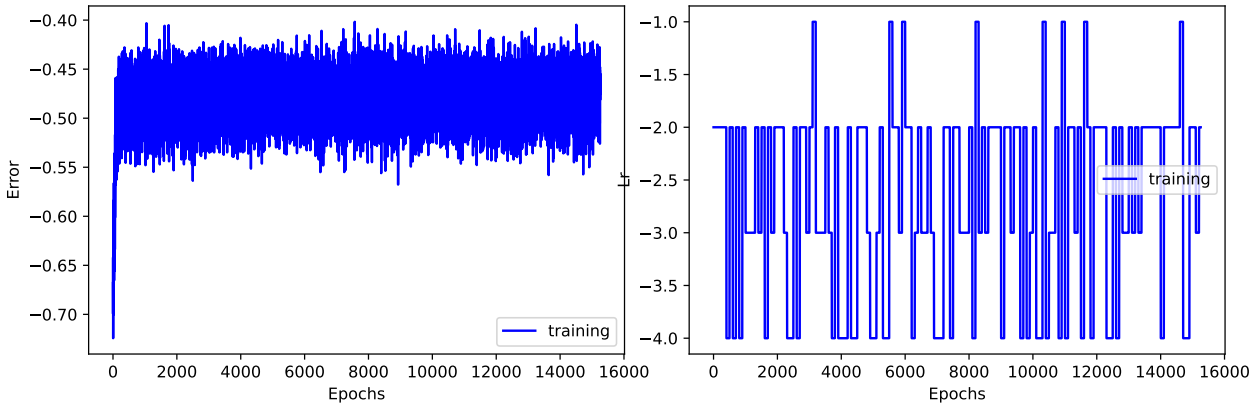


Figure 59: Left: Training error for the training set $(\hat{\gamma}_6^N, \hat{\gamma}_8^N, \tau_3, \tau_4, \tau_5)$ in logarithmic scale using adaptive learning rate and using 1280 neurons in the first layer; Right: Evolution of the learning rate during the learning phase.

4 Conclusion and perspectives

Throughout this project, we have demonstrated favorable outcomes in calibrating both the Black-Scholes and SABR models using CEGEN algorithm. The methodology holds the potential for straightforward extension to high-dimensional problems such as basket portfolios involving multiple correlated spot prices. However, our primary focus is to achieve calibration for the SABR model without relying on the temporal series of volatility, which presents a distinct challenge.

To tackle this objective, we decided to approximate the volatility with a GARCH model but using a neural network model to calibrate the parameters of the GARCH model and not standard methods. Indeed, the reverse problem, with standard methods, finding GARCH parameter values when statistical moments are known, reduces to a set of non-linear equations that in certain circumstances become extremely time consuming to solve and very unstable. Our initial findings have demonstrated the efficiency of our neural network, surpassing the computational speed and performance of even the most advanced direct minimization algorithms. We've elucidated the shortcomings of these algorithms in certain scenarios where our model excelled.

Furthermore, we showcased the efficacy of the neural network in time series, delivering results comparable to the maximum likelihood method but with significantly improved computational efficiency. This shift in approach could has the potential to yield more efficient and effective calibration results.

Another challenge arises when utilizing only a realistic temporal series: the unavailability of independent observations and the requirement for a high number of samples along the trajectory to construct estimators. To overcome this limitation, we propose in future work an innovative solution: data augmentation. By segmenting the main trajectory into multiple sub-trajectories and employing Generative Adversarial Network (GAN) techniques, we can generate additional trajectories with similar distributions, in order to enhance the robustness of our estimators.

References

[1] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986.

[2] Luke De Clerk and Sergey Savl'ev. A machine learning search for optimal garch parameters, 2022.

[3] Christian Francq and jean-michel Zakoian. *GARCH Models: Structure, Statistical Inference and Financial Applications*. 07 2019.