Neural network for Calibration, CEGEN algorithm and machinne learning GARCH method

Mohamed Raed Blel*,[1]

[1]Laboratoire de Probabilités Statistiques et Modélisation, LPSM

January 26, 2024

Contents

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

1 Introduction

In our study, our primary objective is to utilize the CEGEN algorithm [5] for the calibration of various models. The notable advantage of this method lies in its ability to provide theoretical results and exhibit favorable performance in high-dimensional scenarios. In this paper, we present a selection of results obtained from the calibration of two specific models: the Black-Scholes model, which involves one dimension, and the SABR model, which involves two dimensions.

1.1 Completed work

In this paragraph we summarize the different accomplished points:
In section 2:

- An implementation of the CEGEN algorithm [5] using python language.
- We have developed a robust numerical implementation of the Bures metric ([1], [4]), addressing several challenges to ensure convergence. One of the key issues we encountered was the need for a well-defined and compact partition to avoid singularities. To overcome this, we employed a partition by quantiles approach. Another significant aspect of our implementation involved utilizing the singular value decomposition technique to handle matrix square roots and related conditional problems. By tackling these challenges, we have achieved a reliable and effective implementation of the Bures metric.

- We have programmed entirely a parameterized neural network architecture where for certain parameters, we have utilized the sigmoid activation function, while for others, the cotangent activation function has been applied. This combination of activation functions allows us to achieve stability and optimize the performance of the neural network in our specific context.

- The implemented algorithm was tested on both the Black-Scholes (BS) model and the SABR model. While the main authors of the algorithm did not originally test the SABR model, we encountered additional convergence issues due to the direct dependency of the forward rate on the volatility process in this model. To address this challenge, we made adaptations to the algorithm by incorporating a partitioning technique that alternates the partitioning and the minimization function depending on the variable. This approach helped to improve the convergence of the algorithm for the SABR model.

Remark

- The current method is particularly useful when dealing with a basket portfolio that involves dependent noises.

In section 3:

- We propose a novel approach to approximate the volatility of the SABR model by leveraging the GARCH model. Drawing inspiration from the findings presented in [2], we refine certain theoretical aspects and introduce a neural network designed to accurately estimate the parameters governing the GARCH volatility trajectory.

- To gauge the effectiveness of our neural network model, we conduct a comprehensive comparison with various direct minimization algorithms. Our investigation reveals that our model outperforms these algorithms, particularly in the face of topological challenges associated with high-order moments.

- Demonstrating the robustness of our model, we highlight its stability under significant data perturbations, distinguishing it from other direct minimization algorithms that exhibit instability, especially in challenging regions.

- Delving into the practical application of our model on time series data, we showcase its efficiency in predicting GARCH parameters, particularly when dealing with a high number of trajectory samples. Through a comparative analysis with the maximum likelihood method, we establish the equivalence in performance, with the added benefit of superior computational efficiency offered by our neural network model.

Remarks

- Our objective is to calibrate the parameters (nu, beta, and rho) of the SABR model using the temporal series of the forward rate. Notably, we aim to achieve this calibration without relying on the availability of the volatility temporal series, which is not directly observable in practice. By focusing only on the forward rate temporal series, we aim to relate the GARCH volatility to the volatility of the SABR model and use the results obtained in section 2.
- An alternative avenue for calibrating the SABR model involves the utilization of standard methods based on GARCH volatility approximation. An intriguing aspect is the potential comparison between these conventional methods and our proposed approach. Investigating and contrasting both methods could provide valuable insights into the strengths and limitations of each, contributing to a comprehensive understanding of SABR model calibration.

2 The CEGEN Method

The primary objective of this method is to acquire an empirical probability distribution that effectively approximates the data distribution. In pursuit of this goal, we focus our framework on Ito processes due to their ability to provide robust theoretical results, which we will discuss in the subsequent section.

2.1 Context and theoretical results

The context

Let be the following stochastic differential equation:

$$dX_t = b_X(t, X_t)dt + \sigma_X(t, X_t)dW_t, \quad \forall t \in [0, T], \tag{1}$$

where $b_X : \mathbb{R} \times \mathbb{R}^d \longrightarrow \mathbb{R}^d$ is the drift term and $\sigma_X : \mathbb{R} \times \mathbb{R}^d \longrightarrow \mathcal{M}_{d\times d}$ the volatility term and W is a d-dimensional Brownian motion. We suppose that the parameters b_X and σ_X satisfy the usual Lipschitz conditions and thus ensuring existence eand uniqueness of the solution. Let $\{t_i\}_{0\leq i\leq N}$ a discretization of $[0, T]$, then the Euler Maruyama scheme gives,

$$X_{t_i+\Delta t} = X_{t_i} + b_X(t_i, X_{t_i})\Delta t + \sigma_X(t_i, X_{t_i})\Delta W_{t_i}, \tag{2}$$

where $\Delta W_{t_i} \sim \mathcal{N}(0, \Delta t I_d)$ and are iid for all $\{t_i\}_{0\leq i\leq N}$. We aim to replace this procedure by a process defined by,

$$Y^\theta_{t_i+\Delta t} = Y^\theta_{t_i} + b^\theta_Y(t_i, Y^\theta_{t_i})\Delta t + \sigma^\theta_Y(t_i, Y^\theta_{t_i})Z_{t_i}, \tag{3}$$

where $Z_{t_i} \sim \mathcal{N}(0, \Delta t I_d)$ and b_Y^θ and σ_Y^θ are θ-parametrized functions approximated by neural networks, such that,

$$\mathcal{L}(X_{t_i+\Delta t}|_{X_{t_i}}) \simeq \mathcal{L}(Y^\theta_{t_i+\Delta t}|_{Y^\theta_{t_i}}) \quad \text{for all} \quad \{t_i\}_{0\leq i\leq N}.$$

Hence, the objective is to learn b_Y^θ and σ_Y^θ such that the distribution of X and Y^θ are close.

The metric

The loss function used in this problem is based on the Bures metric and is defined by,

$$\mathcal{W}_2^2(\mathcal{L}(X), \mathcal{L}(Y)) = ||\mathbb{E}[X] - \mathbb{E}[Y]||^2 + \mathcal{B}^2(Var(X), Var(Y)), \tag{4}$$

where $\mathcal{B}$ is the Bures metric. This metric is interesting as:

- **If X and Y are gaussian**, the considered metric is the definition of the **Wasserstein 2 distance**.
- We can compute exactly the wasserstein 2 distance from the Bures formulation:
$$\mathcal{B}^2(A, B) = Tr(A) + Tr(B) - 2Tr(\sqrt{A^{\frac{1}{2}} B A^{\frac{1}{2}}}),$$
for positive definite matrices A and B.
- **Whenever $\mathcal{L}(X)$ and $\mathcal{L}(Y)$ coincide in $\mathcal{W}_2$ then the drift and the volatility terms coincide as well.**

Loss function

To build the generator we create at each time t_i a partition $(I_k)_{k\leq N_K}$ of the union of supports of X_{t_i} and $Y^\theta_{t_i}$. For a given batch of samples, $\mathcal{L}(Y^\theta_{t_{i+1}}|Y^\theta_{t_i} \in I_k)$ is then approximated by extracting $Y^\theta_{t_i}$ such that $Y^\theta_{t_i} \in I_k$ and evaluated with the $\mathcal{W}_2$ metric in order to approach $\mathcal{L}(X_{t_{i+1}}|X_{t_i} \in I_k)$ where X_{t_i} is extracted also such that $X_{t_i} \in I_k$. Hence the loss function is defined as,

$$l(X, Y^\theta) = \sum_{i=0}^{N-1} \sum_{k=1}^{N_K} \mathcal{W}_2^2(\mathcal{L}(X_{t_{i+1}}|X_{t_i} \in I_k), \mathcal{L}(Y^\theta_{t_{i+1}}|Y^\theta_{t_i} \in I_k)). \tag{5}$$

Theoretical result

The expression of the metric yields the following results in terms of controlling the error between the predicted drift and the actual drift, as well as the error between the predicted volatility and the actual volatility.

Proposition: Assume that $\sigma_X^2(t_i,.)$ and $\sigma_{Y^\theta}^2(t_i,.)$ are strictly positive and, together with $b_X(t_i,.)$ and $b_{Y^\theta}(t_i,.)$, K-Lipschitz in their second coordinate. For $t_i \in \mathcal{T}$, let $(I_k)_k$ be a regular partition covering $Supp(X_{t_i}) \cup Supp(Y_{t_i})$ with mesh size Δx and let $\epsilon > 0$.
If $\mathcal{W}_2^2(\mathcal{L}(X_{t_i+\Delta t}|X_{t_i} \in I_k), \mathcal{L}(Y^\theta_{t_i+\Delta t}|Y^\theta_{t_i} \in I_k)) \leq \epsilon^2$ for any k, then, for z in the partition we have,

$$||b_X(t_i,z) - b_{Y^\theta}(t_i,z)||_2 \leq \frac{\epsilon + \Delta x}{\Delta t} + 2K\Delta x, \tag{6}$$

Furthermore if d=1,

$$||\sigma_X(t_i,z) - \sigma_{Y^\theta}(t_i,z)||_2 \leq \frac{\epsilon}{\sqrt{\Delta t}} + 2K\Delta x. \tag{7}$$

and, when $d \geq 1$ and $Tr(\sigma_X^2(t_i,z)) = Tr(\sigma_{Y^\theta}^2(t_i,z)) = \alpha$, we have,

$$||\sigma_X(t_i,z) - \sigma_{Y^\theta}(t_i,z)||_2 \leq \sqrt{\frac{2\alpha}{\Delta t}}\epsilon + 2K\Delta x. \tag{8}$$

Key elements of the proof
To obtain these results we only need to observe that

$$X_{t_{i+1}}|(X_{t_i} = z) \sim \mathcal{N}(z + b_X(t_i,z)\Delta t, \sigma_X^2(t_i,z)\Delta t)$$

and,

$$Y^\theta_{t_{i+1}}|(Y^\theta_{t_i} = z) \sim \mathcal{N}(z + b_{Y^\theta}(t_i,z)\Delta t, \sigma_{Y^\theta}^2(t_i,z)\Delta t)$$

then considering the K-Lipschitz condition and applying the condition

$$\mathcal{W}_2^2(\mathcal{L}(X_{t_i+\Delta t}|_{X_{t_i} \in I_k}), \mathcal{L}(Y^\theta_{t_i+\Delta t}|_{Y^\theta_{t_i} \in I_k})) \leq \epsilon^2$$

with the reverse triangular inequality we end up with the required results.

- The proof is checked and the inequalities are correct.
- We need to control the ratio by considering a fine partition, which increases the computational cost. In the same time we are constrained to the Wasserstein error ϵ that needs an enough high sampling number to estimate correctly.

- In order to ensure accurate control over the ratio $\frac{\Delta x}{\Delta t}$, we must employ a fine partition, which inevitably increases the computational cost. However, we are simultaneously faced with the constraint of the Wasserstein error ϵ, which requires a sufficiently large number of samples for accurate estimation. Balancing these considerations becomes crucial as we strive to achieve both precise control over the ratio $\frac{\Delta x}{\Delta t}$ and accurate estimation of the Wasserstein error ϵ, while being mindful of the associated computational demands.

The algorithm

In this paragraph, we introduce the CEGEN algorithm 1. At each time step t_i, we sample m observations of $X_{t_{i+1}}$ and $Y^\theta_{t_{i+1}}$ according to the corresponding equation. Subsequently, a partition function is applied using the quantiles method to create N_K subdivisions, from which the mesh size Δx is determined. For each subdivision I_k, we estimate the Wasserstein distance $\mathcal{W}_2^2(\mathcal{L}(X_{t_i+\Delta t}|X_{t_i} \in I_k), \mathcal{L}(Y^\theta_{t_i+\Delta t}|Y^\theta_{t_i} \in I_k))$. Finally, the backpropagation is performed on the overall loss function given by

$$l(X, Y^\theta) = \sum_{i=0}^{N-1} \sum_{k=1}^{N_K} \mathcal{W}_2^2(\mathcal{L}(X_{t_{i+1}}|X_{t_i} \in I_k), \mathcal{L}(Y^\theta_{t_{i+1}}|Y^\theta_{t_i} \in I_k)).$$

Algorithm 1 Algorithm CEGEN.

Input: $\mathcal{D}$ samples of X, m batch size, K Nb of subdivisions, γ learning rate
Initialize: θ (randomly picked)
while Not converged **do**
  for $t_i = 0...T$ **do**
    Sample m observations (x_{t_i+1}) from of X_{t_i+1}
    Sample $z \sim \mathcal{N}(0, I_D \Delta t)$
    $y_{t_i+1} \leftarrow y_{t_i} + g^b_\theta(t_i, y_{t_i})\Delta t + g^\Sigma_\theta(t_i, y_{t_i})z$
    $I_K \leftarrow K$ subdivisions of Supp$(X_{t_i})\cup$ Supp(Y_{t_i})
    for $k = 0...K$ **do**
      $\ell_{t_i+1,k} \leftarrow \mathcal{W}_2^2(\mathcal{L}(x_{t_i+1}|x_{t_i} \in I_k), \mathcal{L}(y_{t_i+1}|y_{t_i} \in I_k))$
    end for
  end for
  $\theta = \theta - \gamma \nabla_\theta \sum_{t_i=1}^{T-1} \sum_{k=1}^{K} \ell_{k,t_i+1}$
end while
Output: y

Figure 1: CEGEN algorithm

Neural Network architecture

We employ a neural network as the generator, which have entirely programmed and which consists of three layers with four times the dimension of the data for each layer. To ensure appropriate output ranges, we apply the sigmoid function at the end of the network for the parameters ν and β, while the cotangent function is utilized for ρ. The Adam optimizer is employed for training the neural network, with a learning rate (l_r) set to 10^{-4}. The factors β_1 and β_2, which correspond to gradients and second moments of gradients, are respectively assigned values of 0.5 and 0.99.

2.2 Application to a fully observed one-dimensional Black Scholes model

2.2.1 The model

The Black Scholes model is described by the following equation,

$$dS_t = \mu S_t dt + \sigma S_t dW_t, \tag{9}$$

with initial values S_0. In this equation, S_t is the spot price, σ is the volatility of the return, μ the drift term of the return and W_t Brownian motion. We are looking for calibrating the following parameters σ and μ using the temporal series of the spot price.

Predictive model

The predictive model is given by the following discretized Euler Maruyama equation:

$$S_{t_{i+1}} = S_{t_i} + b^\theta(t_i, S_{t_i})\Delta t + \sigma^\theta(t_i, S_{t_i})\Delta W_{t_i}, \tag{10}$$

Such that the predictive drift term $b^\theta(t, S_t)$ tries to approximates the term μS_t and the predictive volatility term $\sigma^\theta(t, S_t)$ tries to approximate the term σS_t for each t.

2.2.2 Numerical Results

Calibration phase for the Black and Scholes model

During the calibration phase, we simulate trajectories over the interval $[0, T] = [0, 1/4]$ using a time step of $\Delta t = 2.2510^{-2}$. We utilize a total of $M = 10000$ trajectories, with an initial condition of $S_0 = 1$. The partitioning of the space is performed by dividing it into four parts using the quantile method.
For the training process, we set a batch size of 300 and conduct 10 epochs.

The obtained results include the Wasserstein error, as well as the estimated values for the volatility σ and the drift term μ. In figure 2, we observe a decrease of the loss function during the training phase to achieve convergence at 10^{-6}.

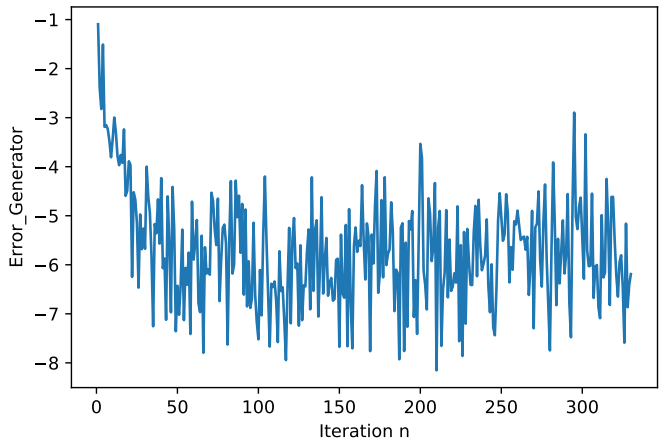

Figure 2: Generator error

In figure 3, we observe that the volatility term converges to a predicted volatility around 0.28 while the actual volatility is equal to 0.3.

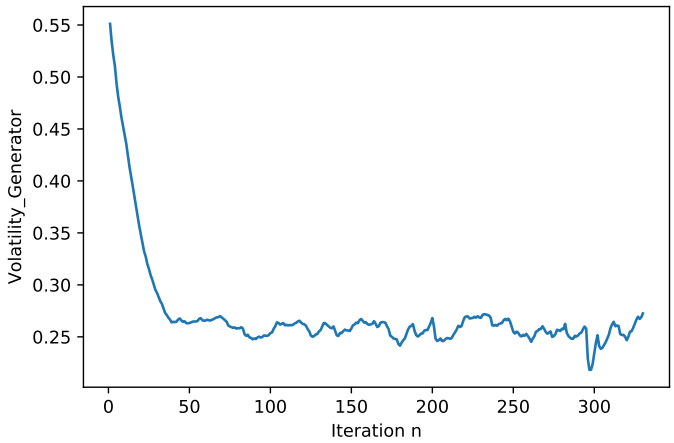

Figure 3: Prediction of the volatility

In figure 4, we observe that the drift term converges to a predicted drift around 0.78 while the actual drift is equal to 0.8.

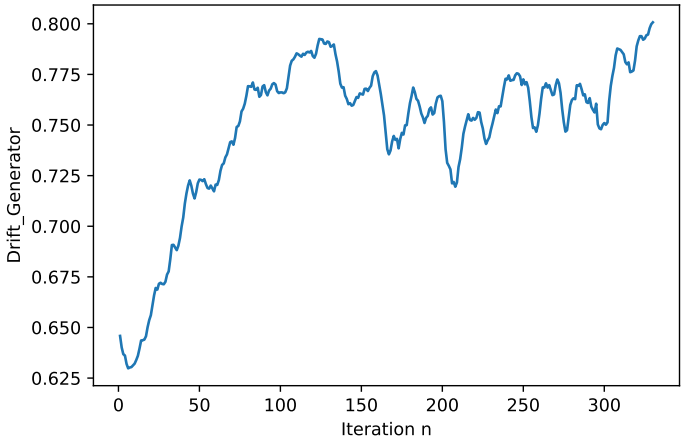

Figure 4: Prediction of the drift

2.3 Application to a fully observed two-dimensional SABR model

This section serves as an intermediate modelization between the one-dimensional model and the partial two-dimensional model where only one dimension is observed while the second dimension remains unknown. However it useful for basket portfolio problems.

2.3.1 The model

The SABR model is described by the following three equations,

$$dF_t = \alpha_t F_t^{\beta} dW_t^1, \tag{11}$$

$$d\alpha_t = \nu \alpha_t dW_t^2, \tag{12}$$

$$\mathbb{E}[dW_t^1 dW_t^2] = \rho dt, \tag{13}$$

with initial values F_0 and α_0. In these equations, F_t is the forward rate, α_t is the volatility, and W_t^1 and W_t^2 are correlated Brownian motions, with correlation ρ. We are looking for calibrating the following parameters,

- ν the volatility of the variance,
- β the exponent for the forward rate,
- ρ the correlation between the Brownian motions.

Predictive model

The predictive model is given by the following discretized Euler Maruyama equation:

$$F_{t_{i+1}} = F_{t_i} + \sigma_F^\theta(t_i, \alpha_{t_i}, F_{t_i})\Delta W_{t_i}^1 \tag{14}$$

$$\alpha_{t_{i+1}} = \alpha_{t_i} + \sigma_\alpha^\theta(t_i, \alpha_{t_i})\Delta W_{t_i}^2 \tag{15}$$

$$\Delta W_{t_i}^2 = \rho^\theta \Delta W_{t_i}^1 + \sqrt{1-(\rho^\theta)^2}\Delta W_{t_i}^1, \tag{16}$$

such that the predictive volatility term $\sigma_F^\theta(t, \alpha_t, F_t)$ approximates $\alpha_t F_t^\beta$ and the predictive volatility term of the second equation $\sigma_\alpha^\theta(t, \alpha_t)$ approximates $\nu\alpha_t$ for each t where the Brownian increment $\Delta W_{t_i}^2$ is taken under the equation (16). Hence we look to calibrate the parameters β, ν and ρ by β^θ, ν^θ and ρ^θ following the method given in algorithm 1 such that,

$$\beta^\theta = \frac{ln(\frac{\sigma_F^\theta(t_i, \alpha_{t_i}, F_{t_i})}{\alpha_t})}{ln(F_{t_i})},$$

$$\nu^\theta = \frac{\sigma_\alpha^\theta(t_i, \alpha_{t_i})}{\alpha_{t_i}}.$$

2.3.2 Numerical results

Calibration phase for the SABR model

In the calibration phase we run our trajectories on $[0, T] = [0, 1/4]$ with time step Δt equal $\Delta t = 2.2510^{-2}$, the number of trajectories used is equal to $M = 10000$, the parameters $\alpha_0 = 0.36$, and $F_0 = 100$ and the partition space is divided in 4 parts using the quantile method.
We set a number of batch size equal to 300 and a number of epochs equal to 22. We obtain the following results for respectively the parameters β, ν and ρ. In figure 5, we observe that the parameter β^θ converges to a predicted β around 0.8.

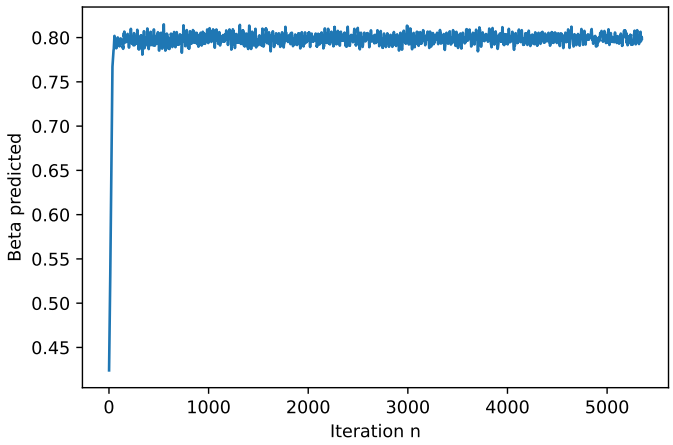

Figure 5: Prediction of Beta

In figure 6, we observe that the parameter ν converges to a predicted ν around 0.21 while the actual ν is equal to 0.2.

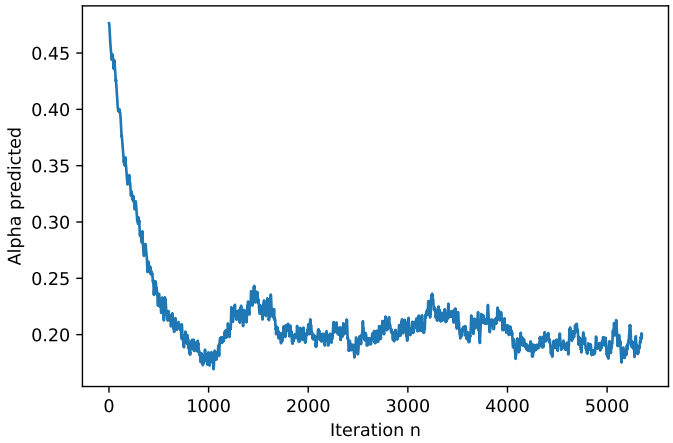

Figure 6: Prediction of ν

In figure 7, we observe that the parameter ρ is quite unstable and converges to a predicted rho around -0.2 while the actual rho parameter is equal to -0.3.

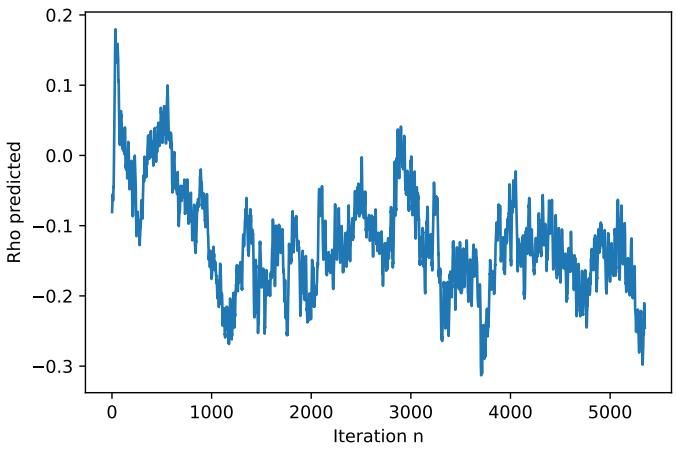

Figure 7: Prediction of Rho

Hence we succeeded to calibrate the SABR model in two dimensions and the Black and Scholes model in one dimension using a neural network method.
Remark:

- The number of trajectories $M = 10000$ is huge and one can cut the main temporal series in several trajectories to get the calibration.
- The code uses some hyper parameters crucial to the convergence of the method, one is the number of partitions used in the quantile method in our case we take it equal to 4.
- Due to the restricted time, we were unable to deeply analyze and compare the sensitivity of convergence when modifying various hyperparameters such as the number of observations or time step.

3 From two dimensional to one dimensional model using ML GARCH method

3.1 Estimating GARCH parameters by Neural Networks method

In this part we investigate the work developped by L.DeCLerk and S.Savlev, in their article [3] on estimating GARCH parameters using a Neural Networks method. The idea is to suppose that the volatility of our time serie could be written as a function of the previous variance along with a stochastic variable x_t to

forecast the future variance. The x_t is generally supposed to be the log return of the price between t and $t-1$ and supposed to be conditionally gaussian. Hence a GARCH(1,1) model is then defined by:

$$\sigma_t^2 = \alpha_0 + \alpha_1 x_{t-1}^2 + \beta_1 \sigma_{t-1}^2 \tag{17}$$

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

To fit the parameters of the GARCH in order to describe the market stock data, we can use maximum likelihood method but these techniques become time consuming especially when we wish to fit parameters in a high frequency setting. To overcome this problem, a neural network is learned to fit the parameters of the GARCH model using the explicit equations given for the second order moment $\mathbb{E}[x^2]$, the fourth order standardized moment Γ_4 and the sixth order standardized moment Γ_6 as following:

$$\mathbb{E}[x^2] = \sigma^2 = \frac{\alpha_0}{1-\alpha_1-\beta 1}, \tag{18}$$

$$\Gamma_4 = \frac{\mathbb{E}[x^4]}{\mathbb{E}[x^2]^2} = 3 + \frac{6\alpha_1^2}{1-3\alpha_1^2-2\alpha_1\beta_1-\beta_1^2}, \tag{19}$$

$$\Gamma_6 = \frac{E(x^6)}{(E(x^2))^3} = \frac{15(1-\alpha_1-\beta_1)^3(1+\frac{3(\alpha_1+\beta_1)}{1-\alpha_1-\beta_1}+\frac{3(1+\frac{2(\alpha_1+\beta_1)}{1-\alpha_1-\beta_1})(\beta_1^2+2\alpha_1\beta_1+3\alpha_1^2)}{1-3\alpha_1^2-2\alpha_1\beta_1-\beta_1^2})}{1-15\alpha_1^3-9\alpha_1^2\beta_1-3\alpha_1\beta_1^2-\beta_1^3}, \tag{20}$$

The autors propose to use also the autocovariance function with lag n as input to the neural network. The reason for this is that the autocovariance function includes more information on a time period rather than the moments which give the information for a specific time. Bollersev in [2] derives an equation for the autocovariance of the square of the return, $Cov(x_t^2, x_{t+n}^2)$ with lag n, γ_n by:

$$\hat{\gamma}_n = \frac{\gamma_n}{\mathbb{E}[x^2]} = \frac{2\alpha_1(1-\alpha_1\beta_1-\beta_1^2)}{1-3\alpha_1^2-2\alpha_1\beta_1-\beta_1^2}(\alpha_1+\beta_1)^{n-1}. \tag{21}$$

This equation of $\hat{\gamma}_n$ above is false and we correct it in the following paragraph:

Correction of $\hat{\gamma}_n$ formula:

Let us start from the equation (A1) of the article: we can write the Garch(1,1) normal equation as following:

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

where $\nu_t = x_t^2 - \sigma_t^2$, then,

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

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

the term $Cov(x_t^2, \nu_{t+n-1})$ gives:

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

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

$$\gamma_n = (\alpha_1 + \beta_1)^n\gamma_0,$$

using $\gamma_0 = \mathbb{E}[x^4] - (\mathbb{E}[x^2])^2$ and replacing wit the expressions of $\mathbb{E}[x^4]$ and $\mathbb{E}[x^2]$ we obtain:

$$\gamma_n = (\mathbb{E}[x^2])^2[2 + \frac{6\alpha_1^2}{1 - 3\alpha_1^2 - 2\alpha_1\beta_1 - \beta_1^2}](\alpha_1 + \beta_1)^n,$$

finally,

$$\hat{\gamma}_n = (\mathbb{E}[x^2])[2 + \frac{6\alpha_1^2}{1 - 3\alpha_1^2 - 2\alpha_1\beta_1 - \beta_1^2}](\alpha_1 + \beta_1)^n. \tag{22}$$

The authors propose to train the neural network on a data set formed by $(\mathbb{E}[x^2], \Gamma_4, \Gamma_6)$ in order to learn α_1 and then using the following equations we deduce the values of β_1:

$$\beta_1 = \sqrt{1 - 2\alpha_1^2 - \frac{6\alpha_1^2}{\Gamma_{4,emp} - 3}}, \tag{23}$$

and α_0 by:

$$\alpha_0 = \sigma^2_{emp}(1 - \alpha_1 - \beta_1), \tag{24}$$

where $\Gamma_{4,emp}$ and σ_{emp} are given empirically from the time serie. Following the same procedure as in the article, i found numerical instability to obtain a good results. I changed the Relu activation function by the sinus function.

The neural network is trained on the same training data set $(\mathbb{E}[x^2], \Gamma_4, \Gamma_6)$ to learn α_1, we tryed also to train it on other data sets using the autocovariance of the square of the return process $(\mathbb{E}[x^2], \Gamma_4, \hat{\gamma}_n)$ for different values of n, we present the results in the next sections.

Neural Network architecture

The neural network build in this work is quite different from the one used by the authors, i used a neural network with four layers containing $(1280, 2048, 2048, 1280)$ nodes respectively however the authors have used a neural network with four layers containing $(1280, 2048, 2048, 1280)$ with a ReLu function as an activation function while i have used a sinus function, additionally i have added a sigmoid function as an output activation function to ensure results parameters in $[0, 1]$. We both used a scaling of the data by using the MinMaxScale function. We both used Adam algorithm as optimizer but with a learning rate equal to 10^{-4} while in the article it's equal to 10^{-2}. We will see that i have found better results and more stable performance.

Data set

The training set is generated by sampling uniformly each variable α_0, α_1 and β_1 in $[10^{-2}, 10^{-3}]$, $[0, 0.3]$ and $[0, 1]$ respectively. Note that we need to insure the propriety $\alpha_1 + \beta_1 < 1$ to insure existence of the GARCH solution. I sampled $M = 5.10^4$ data of $(\alpha_0, \alpha_1, \beta_1)$ that give using equations (18), (19) and (20) the training data set $(\mathbb{E}[x^2], \Gamma_4, \Gamma_6)$, or equations (18), (19) and (22) to provide the training data set $(\mathbb{E}[x^2], \Gamma_4, \hat{\gamma}_n)$. We use the mean square difference as a distance between the predicted α_1^θ (output of the NN) and the model value α_1. Hence the loss function used is,

$$MSD = \frac{1}{M}\sum_{i=1}^{M}(\alpha_{1i}^\theta - \alpha_{1i})^2. \tag{25}$$

3.2 Numerical results using different architectural network

In this section, we present the outcomes of the training and validation phases, employing the same architecture and hyper parameters as specified by the authors

in the initial case. We introduce in the following sections our own results obtained after examination of many parameters.

3.2.1 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 8 provides an illustrative example of fitting the parameter α_1 using the neural network model, with a promise of further refinement in subsequent sections. Moving from Figure 9 to 12, we showcase results obtained from distinct training sets, namely $(\mathbb{E}[x^2], \Gamma_4, \hat{\gamma}_1)$, $(\mathbb{E}[x^2], \Gamma_4, \hat{\gamma}_2)$, $(\mathbb{E}[x^2], \Gamma_4, \hat{\gamma}_3)$, and $(\mathbb{E}[x^2], \Gamma_4, \hat{\gamma}_4)$. While these results exhibit favorable shapes, there is an ongoing effort to enhance their stability. Consequently, we delve into an investigation to assess the impact of certain hyperparameters, such as the learning rate, and also explore alternative neural network models, including convolutional networks.

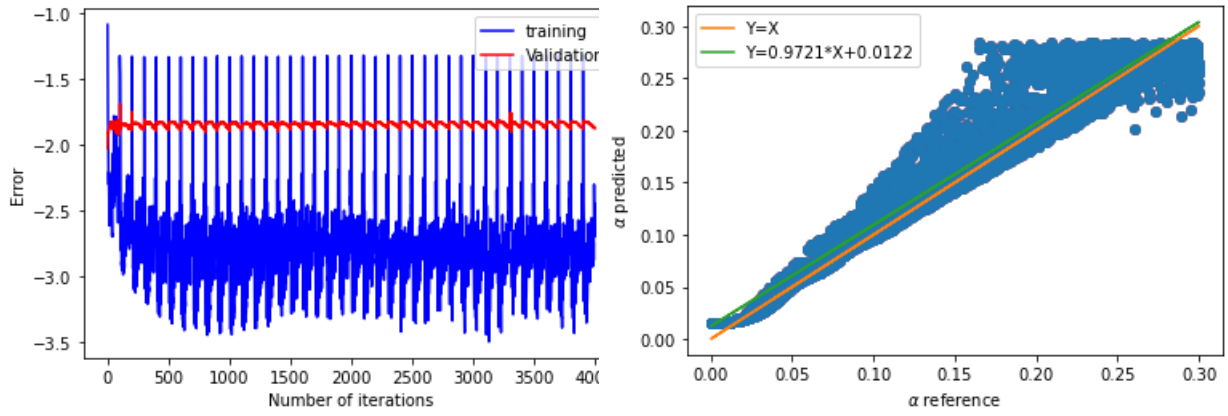

Figure 8: Left: Training error and validation error for the training set $(\mathbb{E}[x^2], \Gamma_4, \Gamma_6)$ in logarithmic scale; Right: Prediction of α_1 for the training set $(\mathbb{E}[x^2], \Gamma_4, \Gamma_6)$.

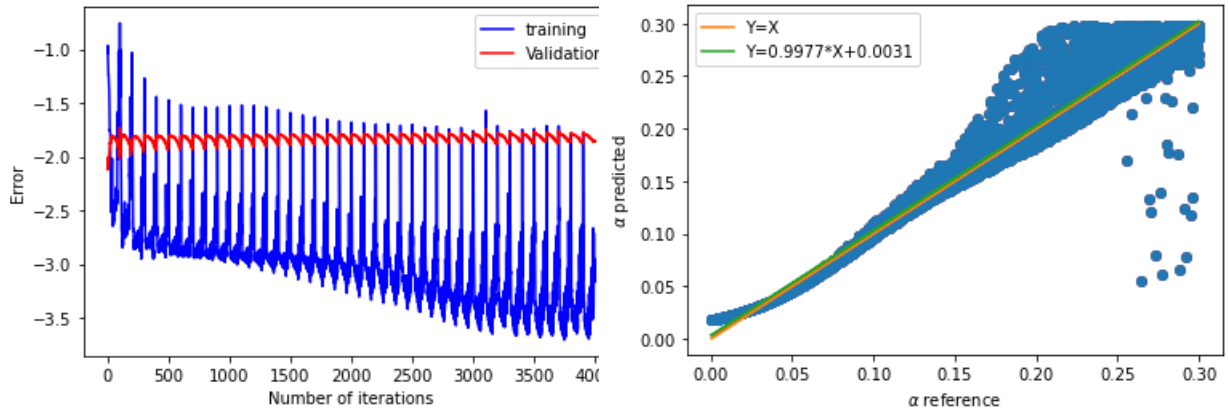

Figure 9: Left: Training error and validation error for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_1$) in logarithmic scale; Right: Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_1$).

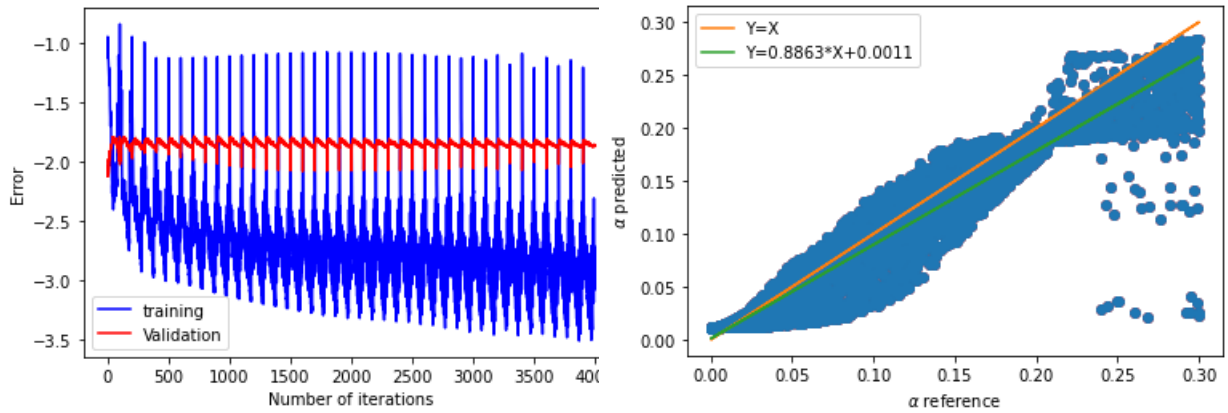

Figure 10: Left: Training error and validation error for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_2$) in logarithmic scale; Right:Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_2$).

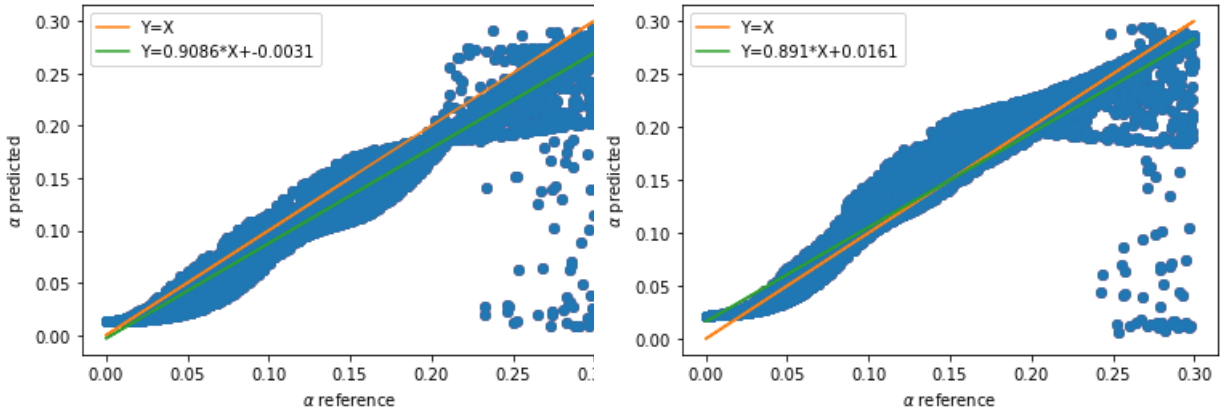

Figure 11: Left: Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_3$); Right: Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_4$).

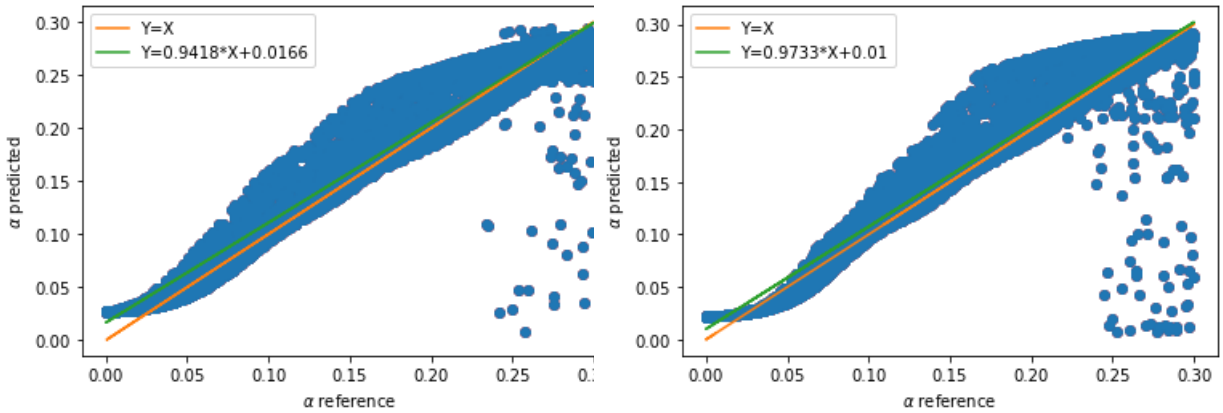

Figure 12: Left: Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_5$).; Right: Prediction of α_1 for the training set ($\mathbb{E}[x^2]$, Γ_4, $\hat{\gamma}_7$).

Impact of some hyper parameters

In this section, we delve into a comprehensive examination of the impact of both the learning rate and the number of neurons in the first layer on our model's performance. While numerous studies have been conducted, we present a concise set of results for clarity. In Figure 13, we employ a fixed learning rate alongside two different configurations for the number of neurons: 128 on the left and 1280 on the right. The results indicate that utilizing 1280 neurons leads to superior outcomes. Moreover, it is worth noting that increasing the number of neurons

contributes to the model's ability to capture more metastable regions, offering a plausible explanation for the observed improvement.

Figure 14 introduces a dynamic learning rate approach, where the learning rate varies across $(10^{-1}, 10^{-2}, 10^{-3}, 10^{-4})$. This dynamic learning rate is achieved through a secondary neural network tasked with adapting the learning rate. The results suggest that this technique is less effective. It appears that the neural network tends to jump between wells without delving deeply into any particular well, indicating a potential challenge in achieving a stable and meaningful adaptation of the learning rate. Further exploration and refinement may be needed to enhance the efficacy of this adaptive learning rate strategy.

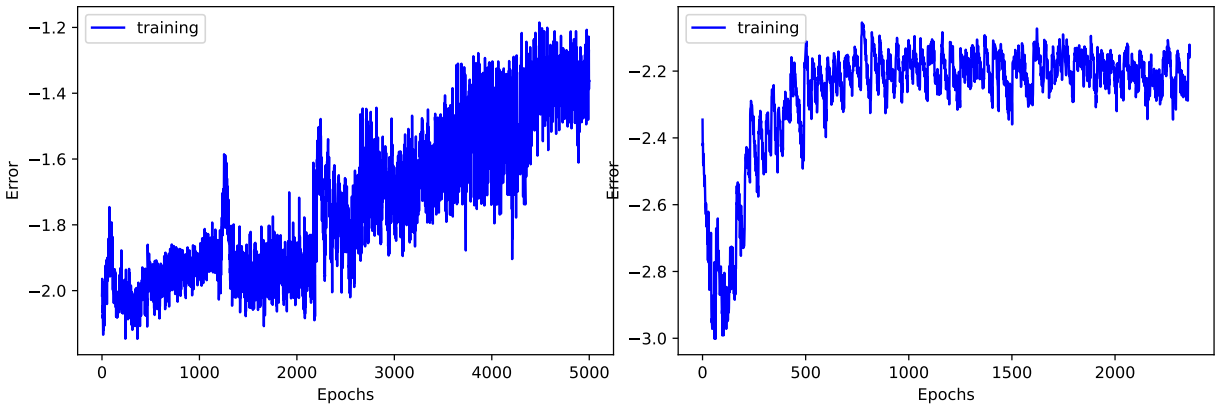

Figure 13: Left: Training error for the training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) in logarithmic scale using 128 neurons in the first layer; Right: Training error for the same set using 1280 neurons in the first layer. Fixed learning rate $lr = 10^{-4}$.

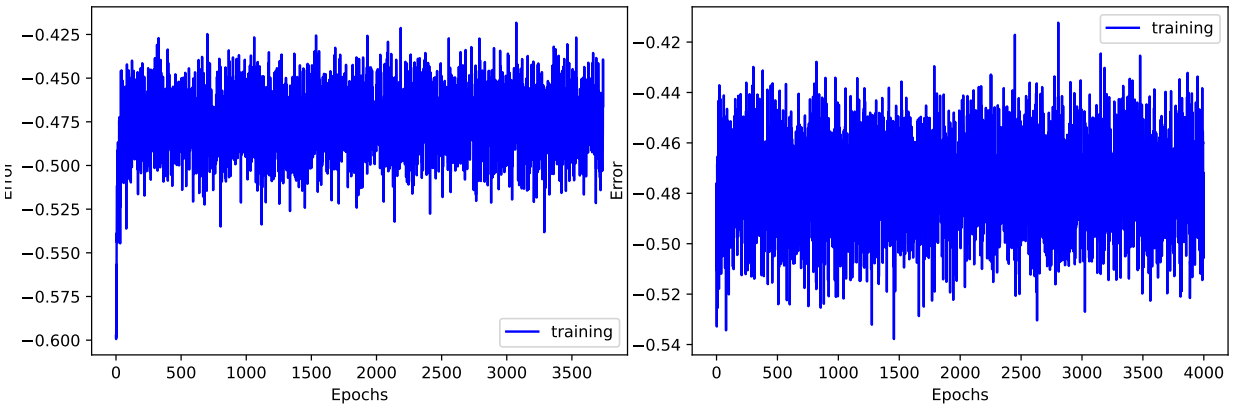

Figure 14: Left: Training error for the training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) in logarithmic scale using dynamic learning rate and using 640 neurons in the first layer; Right: Training error for the same set using 1280 neurons in the first layer. Dynamic learning rate.

3.2.2 A Convolutional network with fixed learning rate

We use in this part a convolutional neural network composed by 4 convolutional layers containing respectively $1280, 2048, 2048, 1280$ neurons each one. We use as activation function the sinus function, we also tried the Relu function and other actvation functions. For the last activation function we use the sigmoid function to insure values in $[0, 1]$.

Using L-moments for the training set:

We propose here to use a fixed learning rate $lr = 10^{-4}$ for the generator and using a training data set composed by the first five L-moments of the time series defined by:

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

Note that we use the L moments ratio for the third, fourth and fifth L-moments defined by $\tau_i = \frac{\lambda_i}{\lambda_2}$ for $i \in \{3, 4, 5..\}$.

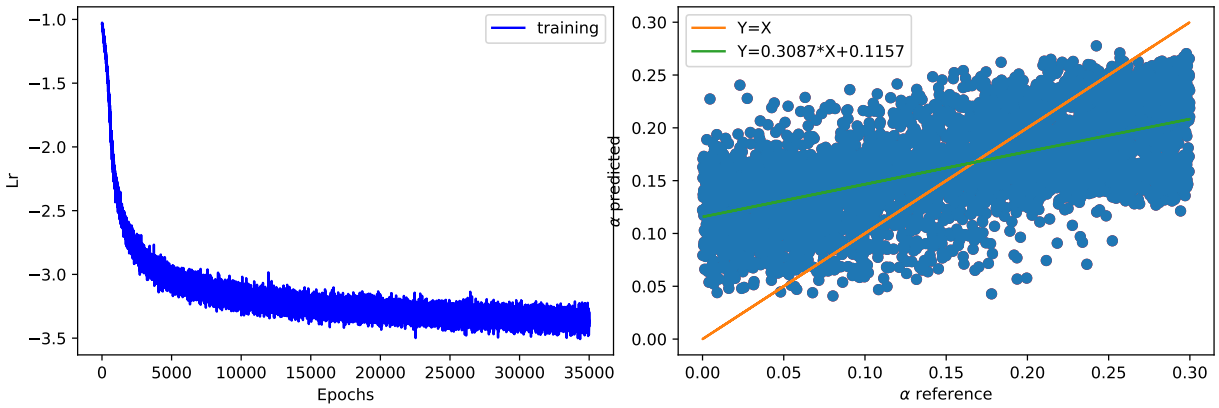

Figure 15: Left: Training error for the training set $(\lambda_1, \lambda_2, \tau_3, \tau_4, \tau_5)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 compared to α_1 of the reference. We use here 5.10^4 data.

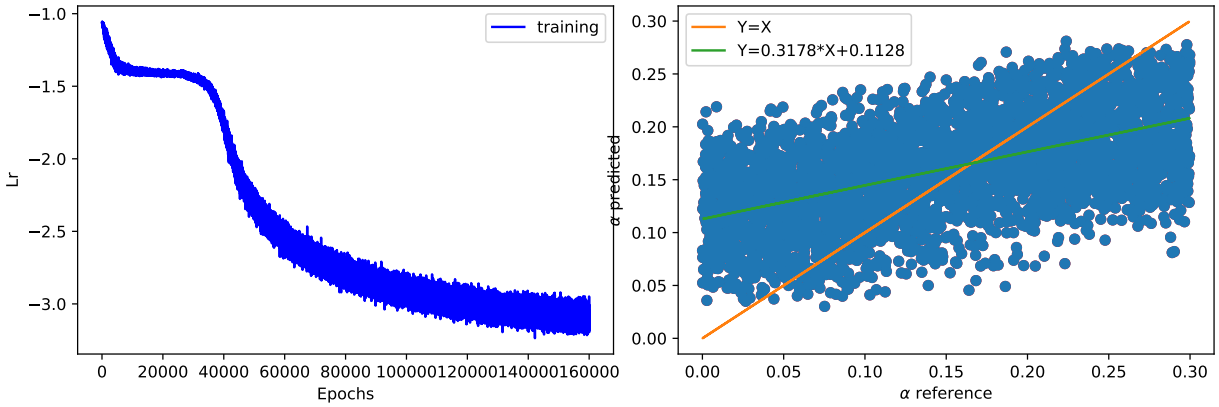

Figure 16: Left: Training error for the training set $(\lambda_1, \lambda_2, \tau_3, \tau_4, \tau_5)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 compared to α_1 of the reference. We use here 5.10^5 data.

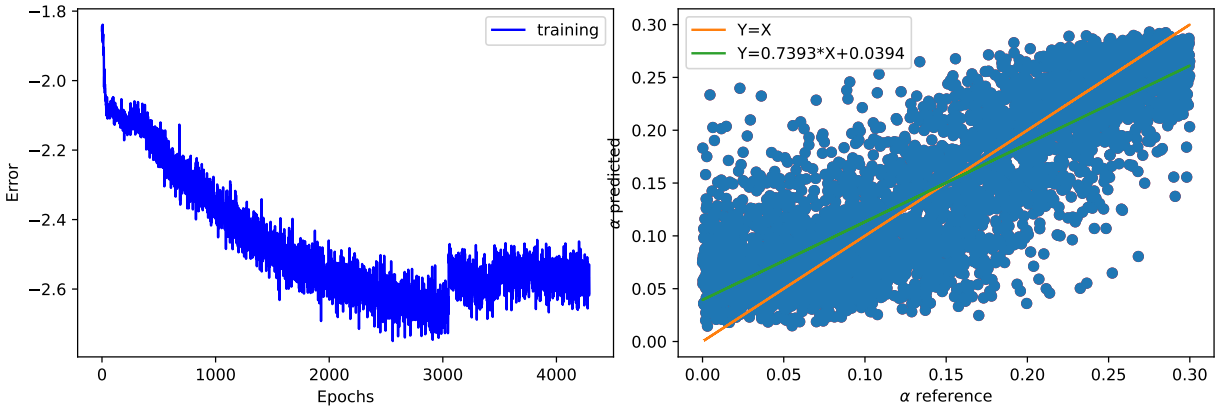

Figure 17: Left: Training error for the training set (τ_3, τ_4, τ_5) in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 compared to α_1 of the reference. We use here 5.10^4 data.

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

3.2.3 A Convolutional network using dynamic learning rate

We propose here to use a dynamic learning rate for the generator, by using two neural networks, one encapsulated in the second, where the first is the generator that calibrates α_1 and the second is the "decision maker", a neural network that learns to choose the best learning rate value.

For the decision maker, the neural network is unsupervised, composed by a linear layer with 4 neurons and a softmax activation function in order to consider the output as a set of probabilities on the indexed neuron. Each index will be linked to one among 4 learning rates. We aim to learn this neural network to choose the learning rate that has the better probability to get a better performance. To do so, we use the reward notion, defined in this case as equal to $reward = -generator_{loss}$ and thus the loss of decision maker is taken as

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

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

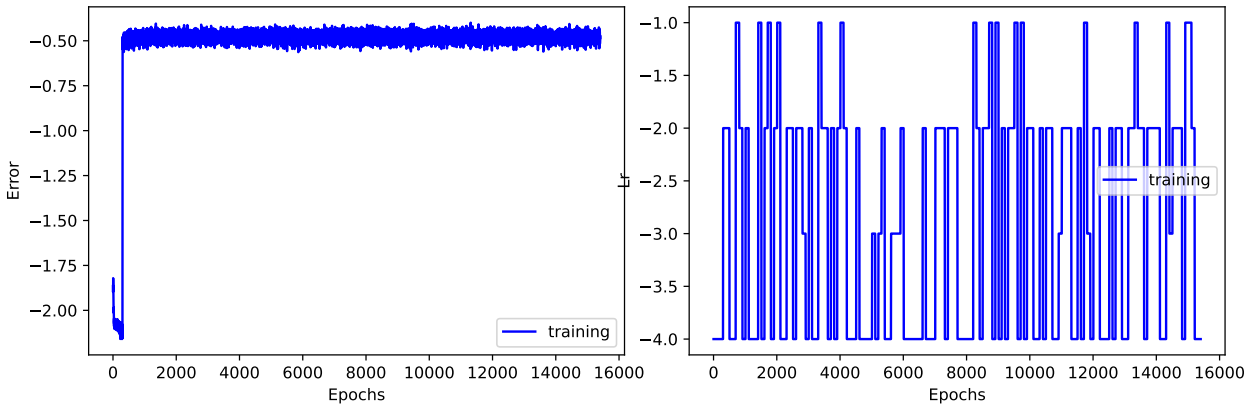

Figure 18: Left: Left: Training error for the training set (τ_3, τ_4, τ_5) in logarithmic scale using adaptive learning rate and using 1280 neurons in the first layer; Right: Evolution of the learning rate during the learning phase.

Message 1: In these first results we observe that we encounter several problems to predict the GARCH parameters even when we try to enhance the training set by other observables such the L-moments or auto-covariance factor of the square of the return with lag n. We tried also different neural network architectures with adaptive learning rate but the results are not enough good. In the next section we explain that this is due to the geometry of the problem, indeed we are faced to a geometry with many wells and data should be chosed such that several constrains are satisfied especially on the first moments.

3.3 Comparison between direct minimization algorithm and neural network model to solve Garch parameters moment equations

To solve the constrained problem given by the system of equations (26), (27), (28) under the constrains (29), (30) and (31) for each triplet $(\mathbb{E}[x^2], \Gamma_4, \Gamma_6)$ we have used several minimization methods (SLSQP: Sequential Least Squares Programming, COBYLA: Constrained optimization by linear approximation, trust-constrain, nelder-Mead, CG, L-BFGS-B, Trust-Krylov, Differential equation, RandomSearch, Couenne...), the best results are given by the SLSQP, Differential equation, Random Search and Couenne algorithms as shown in the next part.
Garch moment equations to be solved:

$$\mathbb{E}[x^2] = \sigma^2 = \frac{\alpha_0}{1-\alpha_1-\beta_1}, \tag{26}$$

$$\Gamma_4 = \frac{\mathbb{E}[x^4]}{\mathbb{E}[x^2]^2} = 3 + \frac{6\alpha_1^2}{1-3\alpha_1^2-2\alpha_1\beta_1-\beta_1^2}, \tag{27}$$

$$\Gamma_6 = \frac{E(x^6)}{(E(x^2))^3} = \frac{15(1-\alpha_1-\beta_1)^3(1+\frac{3(\alpha_1+\beta_1)}{1-\alpha_1-\beta_1}+\frac{3(1+\frac{2(\alpha_1+\beta_1)}{1-\alpha_1-\beta_1})(\beta_1^2+2\alpha_1\beta_1+3\alpha_1^2)}{1-3\alpha_1^2-2\alpha_1\beta_1-\beta_1^2})}{1-15\alpha_1^3-9\alpha_1^2\beta_1-3\alpha_1\beta_1^2-\beta_1^3}, \tag{28}$$

under the constrains:

$$1-\alpha_1-\beta_1 > 0, \tag{29}$$

$$1-3\alpha_1^2-2\alpha_1\beta_1-\beta_1^2 > 0, \tag{30}$$

$$1-15\alpha_1^3-9\alpha_1^2\beta_1-3\alpha_1\beta_1^2-\beta_1^3 > 0. \tag{31}$$

First we present the geometrical problem that will face any algorithm trying to solve the constrained problem given above.

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

The following figures show the irregular geometry of the functions Γ_4 and Γ_6 as function of α_1 and β_1, note that α_1 is taken in $[0, 3]$ to insure finite high moments (until eighth moment) and β_1 in $[0, 1]$. We observe in figure 19 and 23 several irregular zones with many pics and where Γ_4 and Γ_6 could change signs from high positive values to high negative values with a variation smaller than 10^{-14} which is very brutal and which make several problems for minimization solvers, one can verify the values of Γ_4 and Γ_6 for $(\alpha_1 = 0.29, \beta_1 = 0.6220307012376284)$ and for

$(\alpha_1 = 0.29, \beta_1 = 0.622030701237628)$. Additional to the problem of pics, we are faced to a problem of flat curve for some zone as shown in figure 21, in these zones, direct minimization algorithms will find several problem as the gradient vanishes, hence the gradient matrice will be ill-conditioned and the inverse of the Hessian matrice will have several problem to be approximated. In figure 20 the problem is convex and quite far from the flat zone, all algorithms find great results.

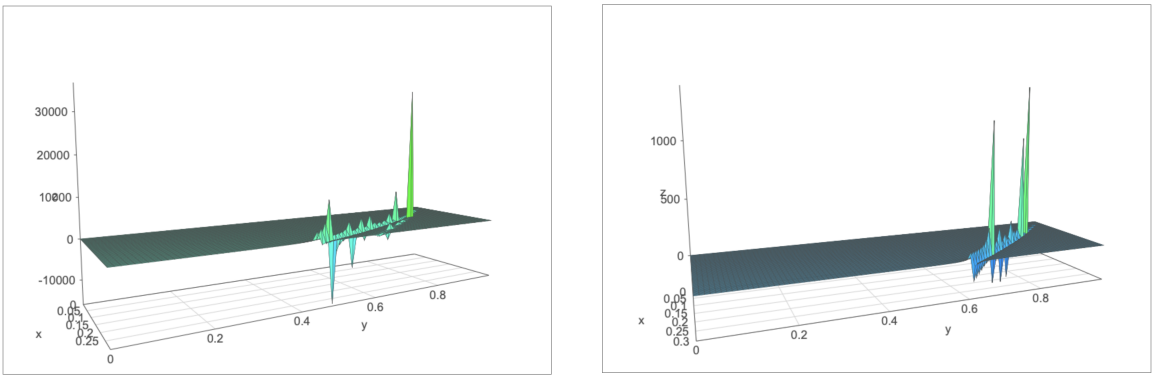

Figure 19: Left: Γ_4 as a function of $\alpha_1 \in [0, 0.3]$ and $\beta_1 \in [0, 1]$; Right: Γ_6 as a function of $\alpha_1 \in [0, 0.3]$ and $\beta_1 \in [0, 1]$.

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

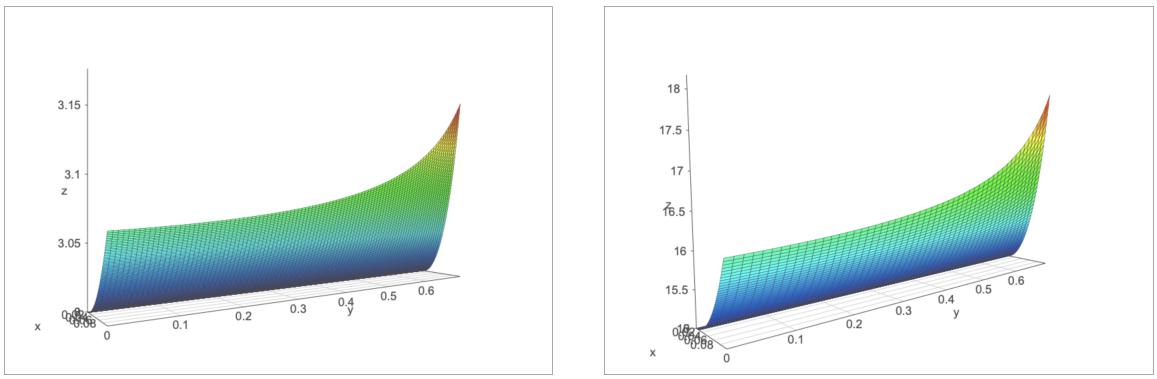

Figure 20: Left: Γ_4 as a function of $\alpha_1 \in [0, 0.1]$ and $\beta_1 \in [0, 0.7]$; Right: Γ_6 as a function of $\alpha_1 \in [0, 0.1]$ and $\beta_1 \in [0, 0.7]$.

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 $\beta_1 \in [0, 0.7]$.

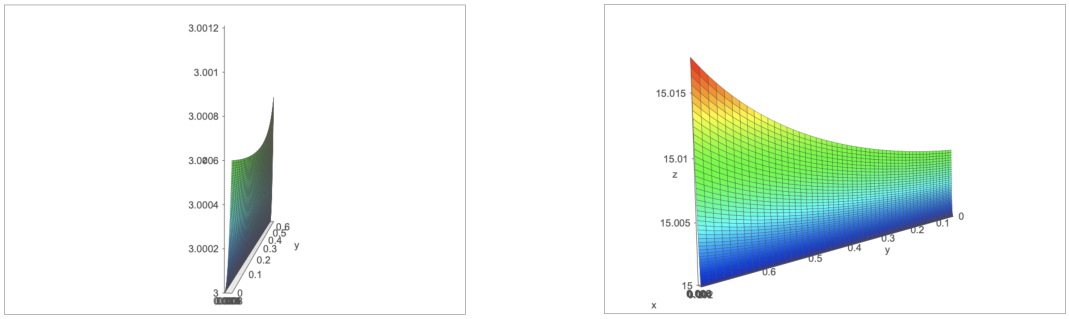

Figure 21: Left: Γ_4 as a function of $\alpha_1 \in [0, 0.01]$ and $\beta_1 \in [0, 0.7]$; Right: Γ_6 as a function of $\alpha_1 \in [0, 0.01]$ and $\beta_1 \in [0, 0.7]$.

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

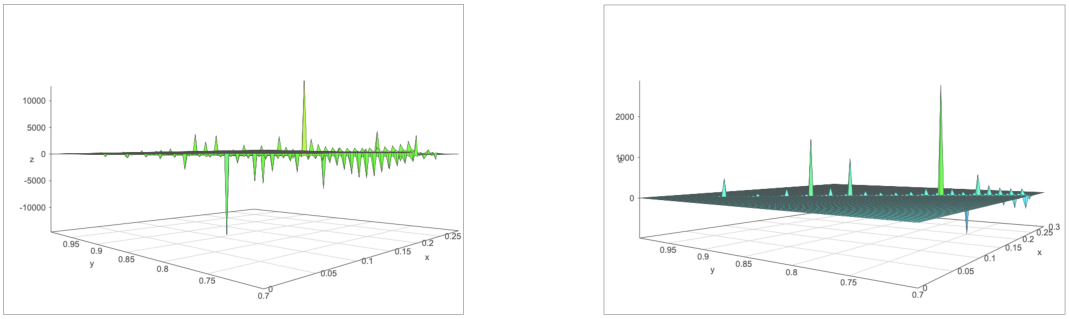

Figure 22: Γ_4 as a function of $\alpha_1 \in [0, 0.3]$ and $\beta_1 \in [0.7, 1]$; Right: Γ_6 as a function of $\alpha_1 \in [0, 0.3]$ and $\beta_1 \in [0.7, 1]$.

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

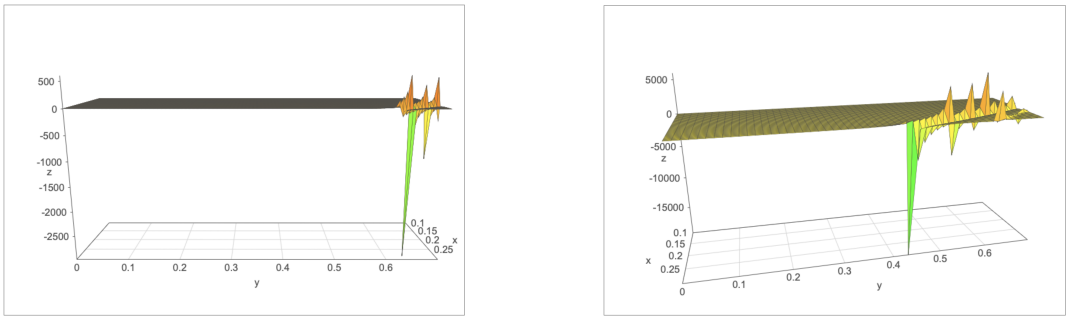

Figure 23: Γ_4 as a function of $\alpha_1 \in [0.1, 0.3]$ and $\beta_1 \in [0, 0.7]$; Right: Γ_6 as a function of $\alpha_1 \in [0.1, 0.3]$ and $\beta_1 \in [0, 0.7]$.

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

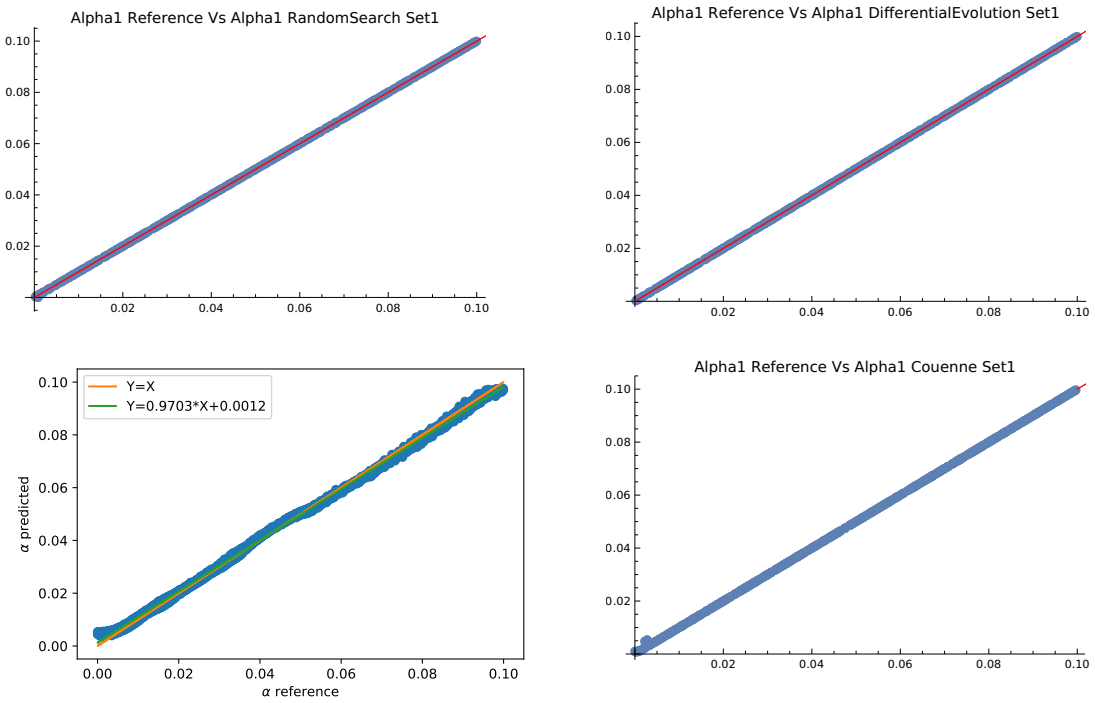

Figure 24: Up and Left:Prediction of α_1 using Random Search algorithm on mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) ; Up and Right: Prediction of α_1 using Differential Evolution algorithm on mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) ;down and left: 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 neural network have succeeded to solve/predict with great performance the Garch constrained problem. The neural network stay 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 around 3 minutes to converge for only 500 parameters (this is was the maximum of points we were allowed to enter for the resolution on mathematica).

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

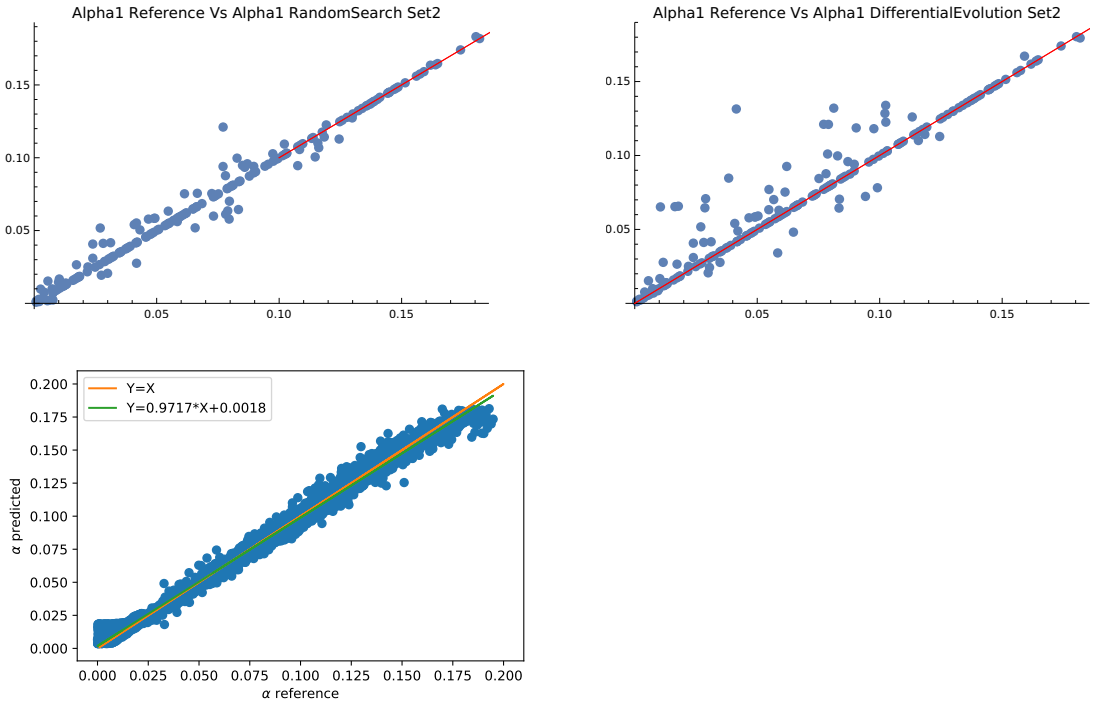

Figure 25: Up and Left:Prediction of α_1 using Random Search algorithm on mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) ; Up and Right: Prediction of α_1 using Differential Evolution algorithm on mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) ;down and left: 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 $set1 = [0, 0.3] \times [0.7, 1]$.

Message 3: In the irregular zone set2, the algorithms Random Search, Differential Evolution haven't succeeded to solve the Garch constrained problem. However, the neural network succeeded to predict with good performance the the Garch constrained problem and stays 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 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:

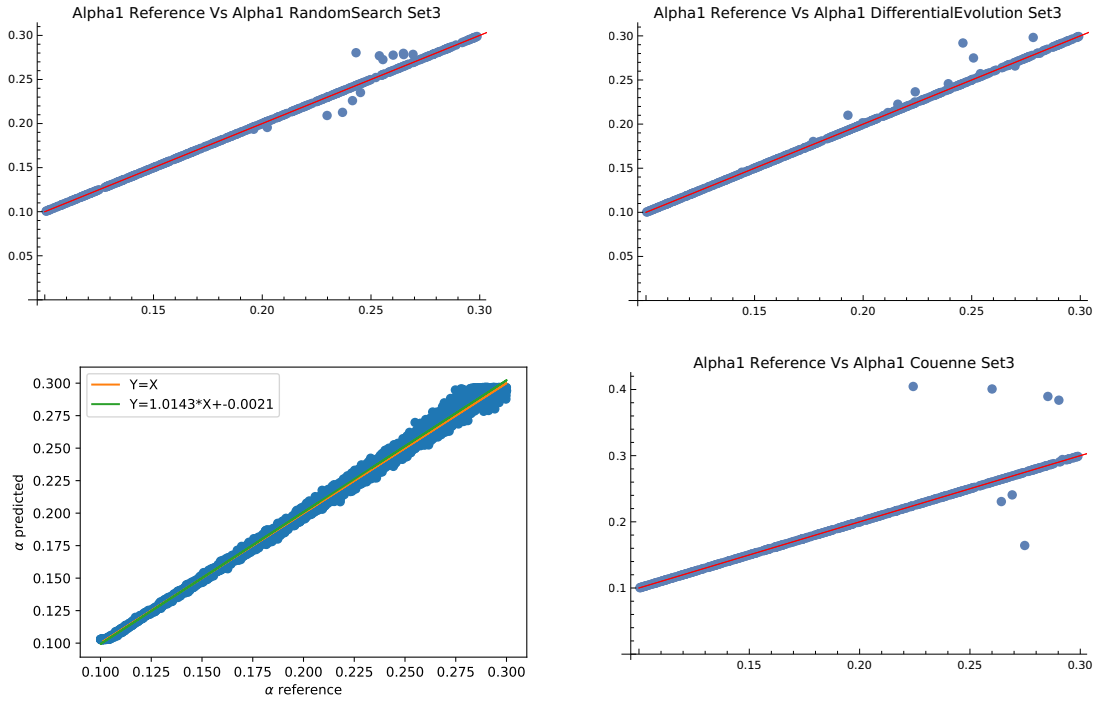

Figure 26: Up and Left:Prediction of α_1 using Random Search algorithm on mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) ; Up and Right: Prediction of α_1 using Differential Evolution algorithm on mathematica for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) ;down and left: 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 algorithms Random Search, Differential Evolution, Couenne have succeeded to solve with good performance the Garch constrained problem but fail on several points, note that the resolution is done on a number of parameters equal to 500, hence the error should raise as we increase the number of parameters.
The neural network show better prediction in this zone especially that the number of parameter for the neural network is equal to 5000. It stays also better in computational time as it takes less than 1 second to compute the result for 5000 parameters on GPU. The direct minimization algorithms take in this zone around 10 minutes to converge for only 500 parameters.

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

In this part we investigate the performance of the direct minimization algorithms compared to the performance of the neural network model to catch the parameter α_1 by adding a noise to the data. We plot some several figures for different magnitude of the standard deviation of the noise.

Let $\epsilon_1, \epsilon_2, \epsilon_3$ random normal variables with mean zero and standard deviation that we change from a test to another as shown in the following figures.

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 $set1 = [0, 0.1] \times [0, 0.7]$.

In figure 27 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 $std(\epsilon_3) \sim 10^{-2} std(\Gamma_6)$.

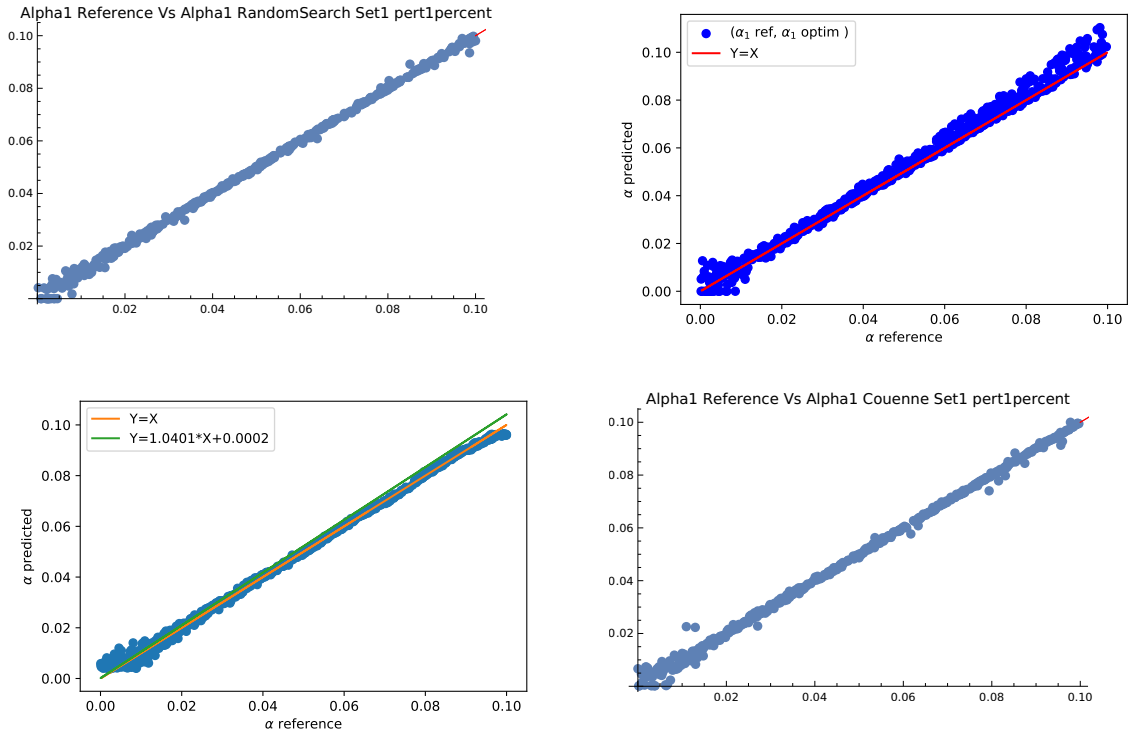

Figure 27: Up and left: Prediction of α_1 using Random Search algorithm; Up and right: Prediction of α_1 using SLSQP algorithm; Down and left: Prediction of α_1 using Neural Network algorithm; Down and right: Prediction of α_1 using Couenne algorithm for the perturbed training set $(\mathbb{E}[x^2] + \epsilon_1, \Gamma_4 + \epsilon_2, \Gamma_6 + \epsilon_3)$ on the set of parameters $set1 = [0, 0.1] \times [0, 0.7]$.

In figure 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 $set1 = [0, 0.1] \times [0, 0.7]$, 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 $std(\epsilon_3) \sim 2.10^{-2} std(\Gamma_6)$.

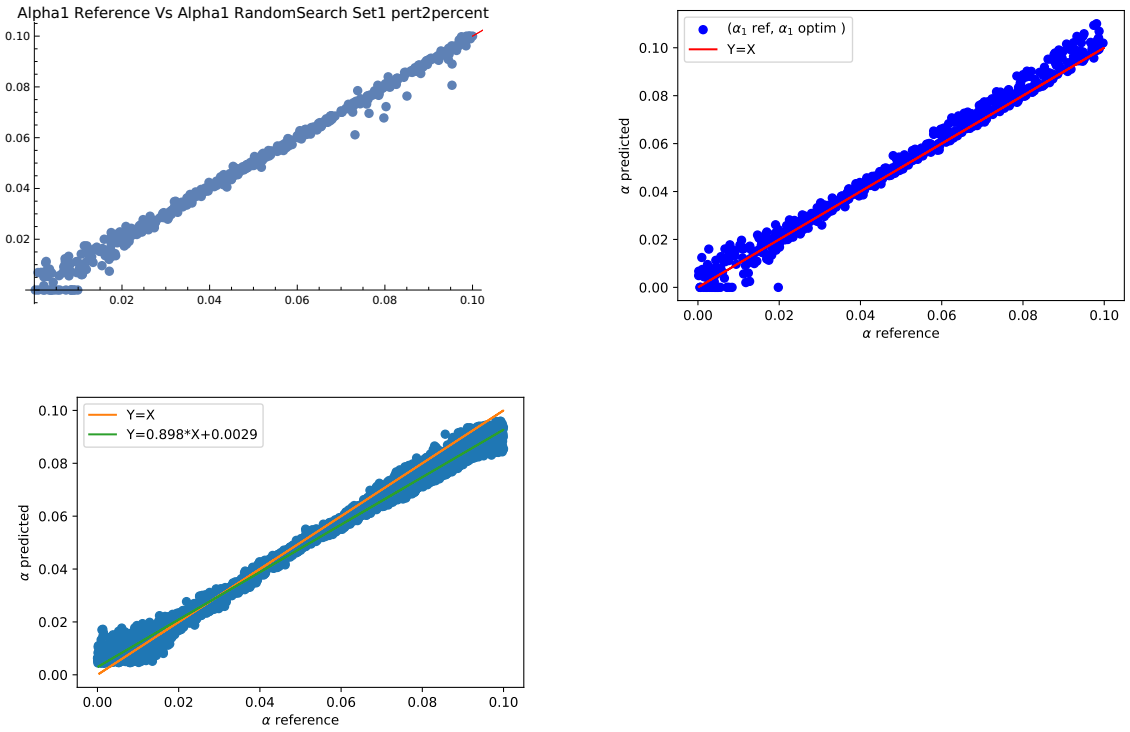

Figure 28: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using SLSQP algorithm on python; Down and left: Prediction of α_1 using Neural Network algorithm on python for the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) 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 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 $set1 = [0, 0.1] \times [0, 0.7]$, 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 $std(\epsilon_3) \sim 3.10^{-2} std(\Gamma_6)$.

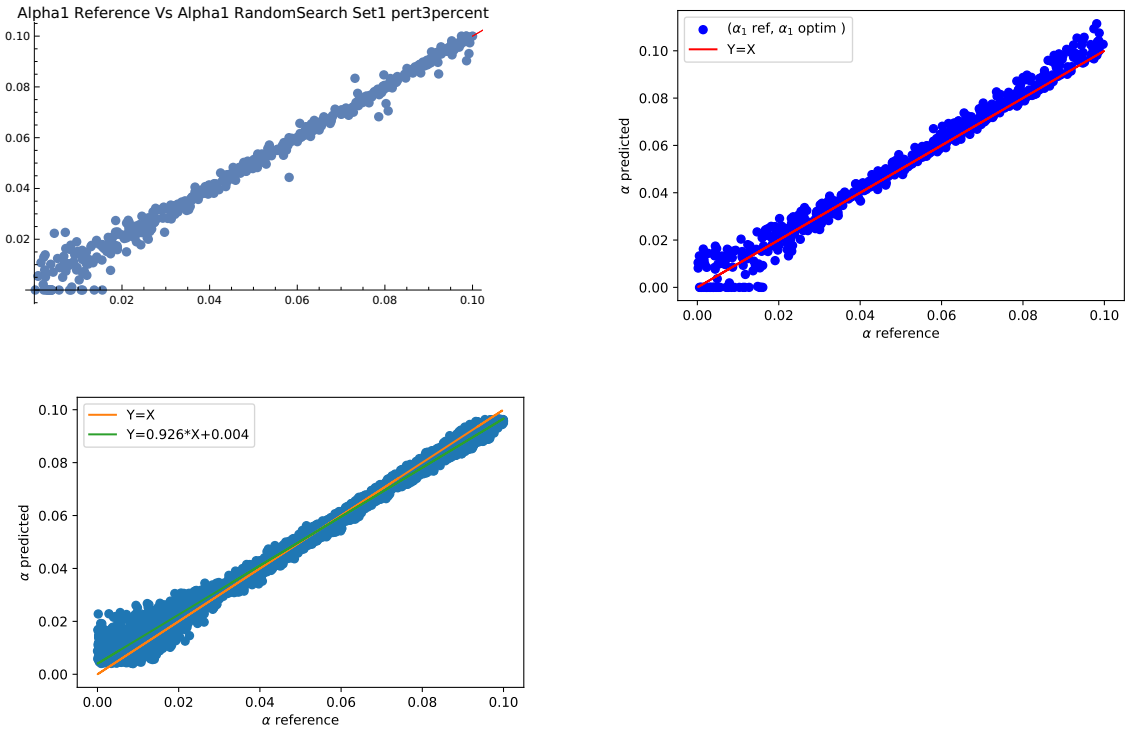

Figure 29: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using SLSQP algorithm on python; Down and left: Prediction of α_1 using Neural Network algorithm on python for the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) 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 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 $set1 = [0, 0.1] \times [0, 0.7]$, 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 $std(\epsilon_3) \sim 5.10^{-2} std(\Gamma_6)$.

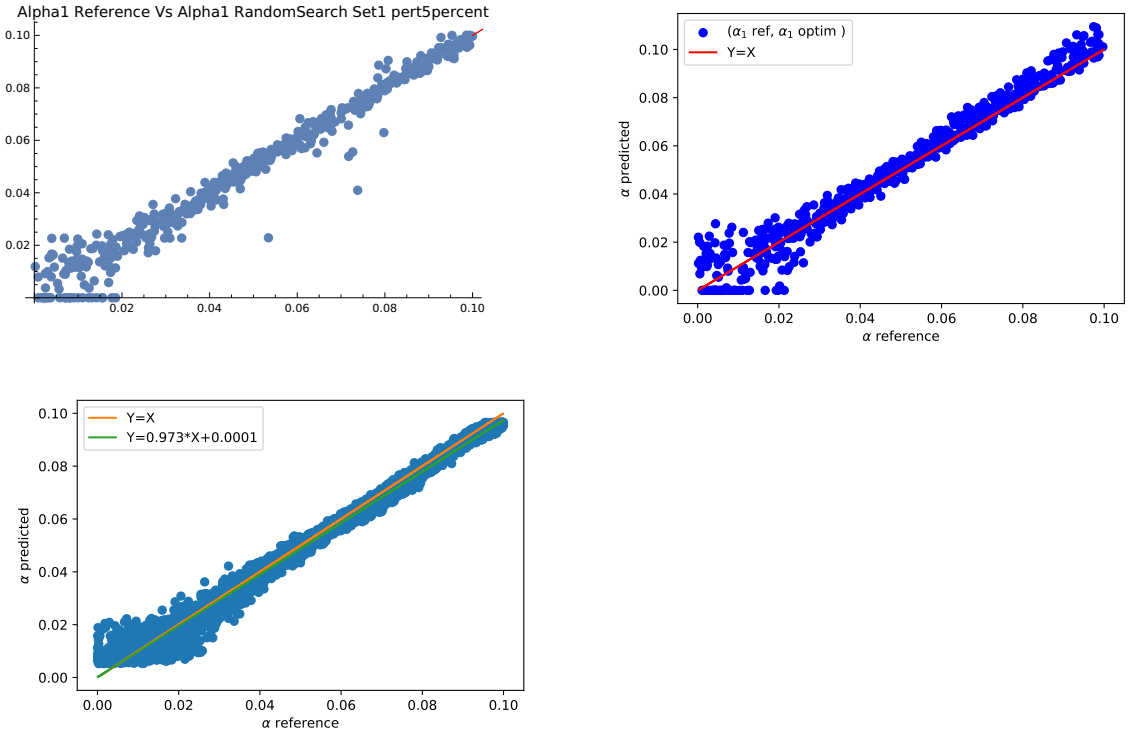

Figure 30: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using SLSQP algorithm on python; Down and left: Prediction of α_1 using Neural Network algorithm on python for the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) 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 31, 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 $std(\epsilon_3) \sim 10^{-1} std(\Gamma_6)$.

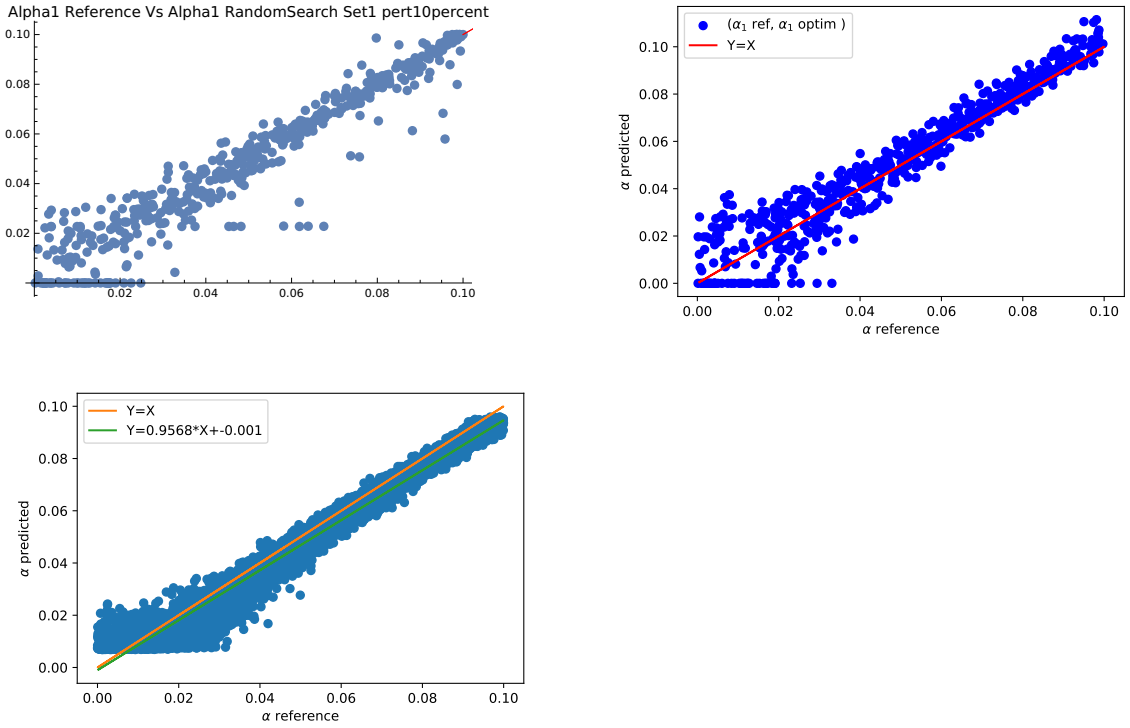

Figure 31: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using SLSQP algorithm on python; Down and left: Prediction of α_1 using Neural Network algorithm on python for the perturbed training set ($\mathbb{E}[x^2]+\epsilon_1$,$\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) 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 part we investigate the performance of the direct minimization algorithms compared to the performance of the neural network model to catch the parameter α_1 by adding a noise to the data. We plot some several figures for different magnitude of the standard deviation of the noise.

Let $\epsilon_1, \epsilon_2, \epsilon_3$ random normal variables with mean zero and standard deviation that we change from a test to another as shown in the following figures.

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 $set1 = [0, 0.3] \times [0.7, 1]$.

In figure 32 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 $std(\epsilon_3) \sim 10^{-2} std(\Gamma_6)$.

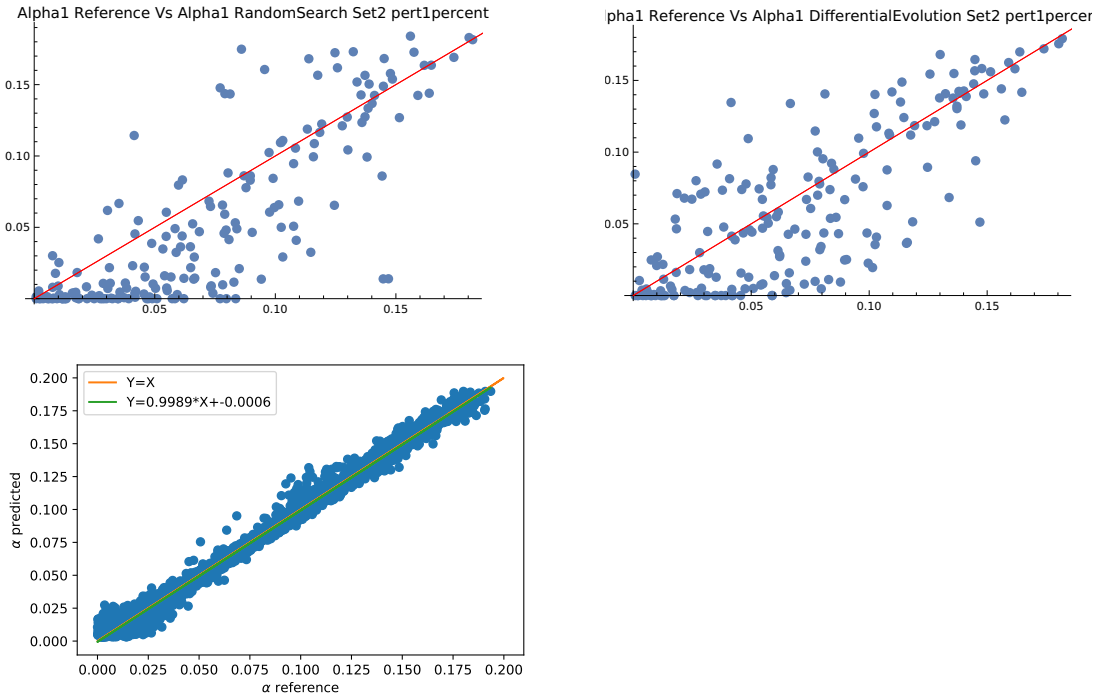

Figure 32: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using SLSQP algorithm on python; Down and left: Prediction of α_1 using Neural Network algorithm on python for the perturbed training set $(\mathbb{E}[x^2] + \epsilon_1, \Gamma_4 + \epsilon_2, \Gamma_6 + \epsilon_3)$ 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 33, 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 $std(\epsilon_3) \sim 2.10^{-2} std(\Gamma_6)$.

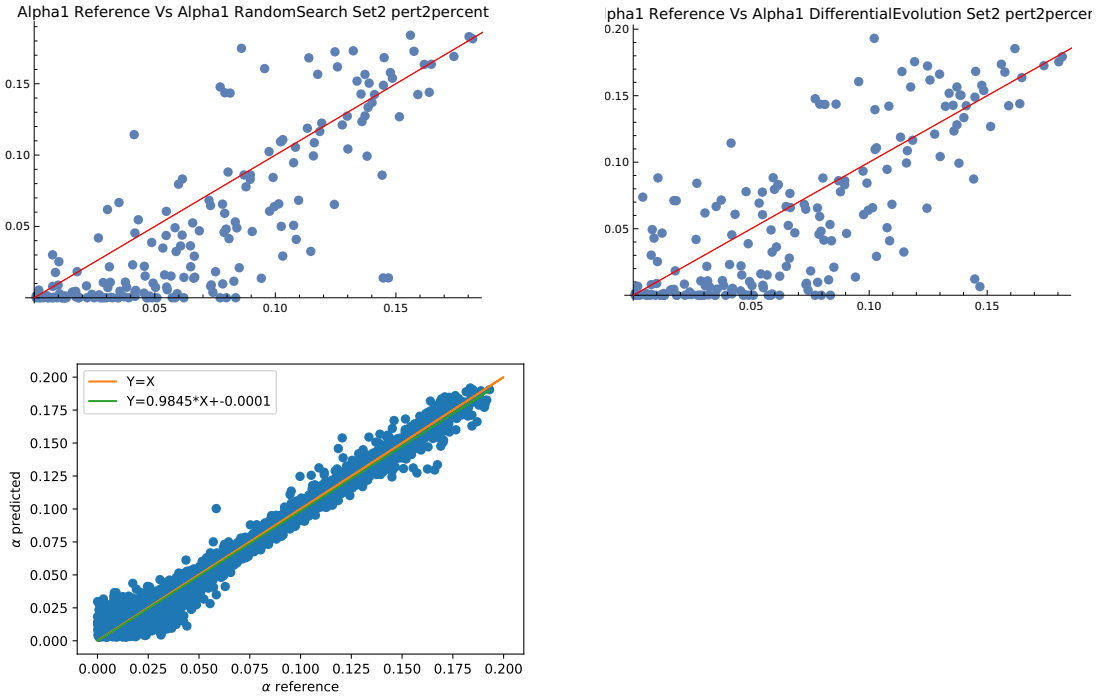

Figure 33: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using Differential Evolution algorithm on mathematica; Down and left: Prediction of α_1 using Neural Network algorithm on python for the perturbed training set ($\mathbb{E}[x^2] + \epsilon_1$,$\Gamma_4 + \epsilon_2$, $\Gamma_6 + \epsilon_3$) 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 algorithms Random Search, Differential Evolution 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 third Set $set3 = [0.1, 0.3] \times [0, 0.7]$ using direct minimization and Neural network comparison:

In figure 34, 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 $std(\epsilon_3) \sim 10^{-2} std(\Gamma_6)$.

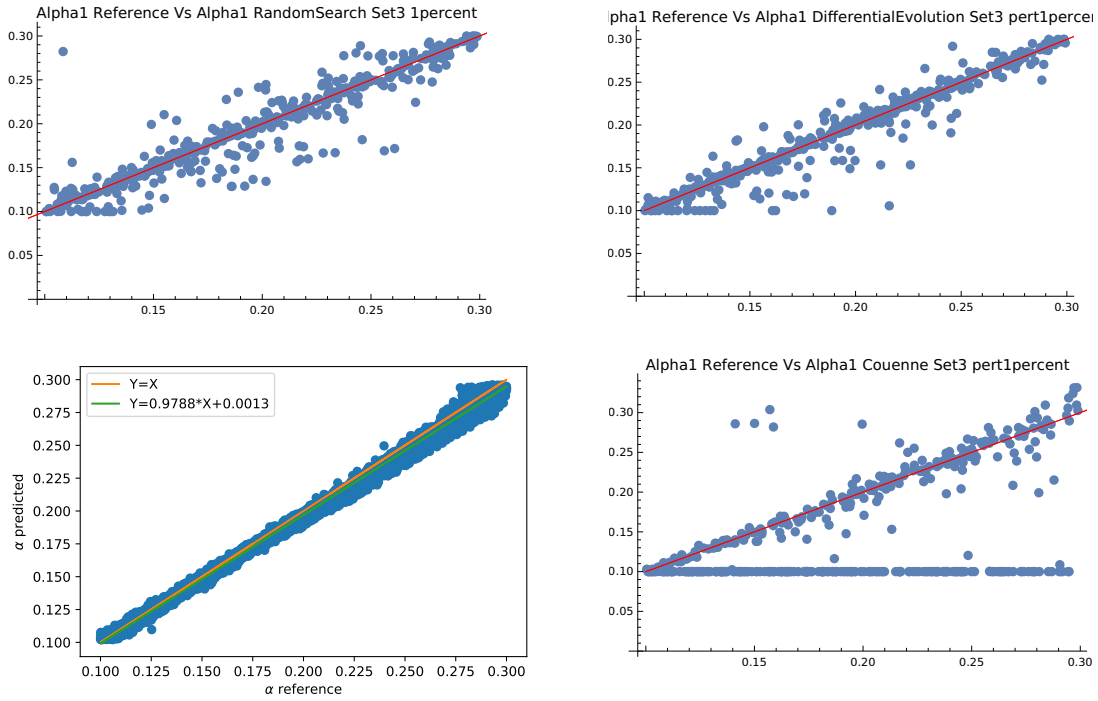

Figure 34: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using Differential Evolution algorithm on mathematica; Down and left: Prediction of α_1 using Neural Network algorithm on python; Down and right: Prediction of α_1 using Couenne algorithm 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 35, 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 $std(\epsilon_3) \sim 2.10^{-2} std(\Gamma_6)$.

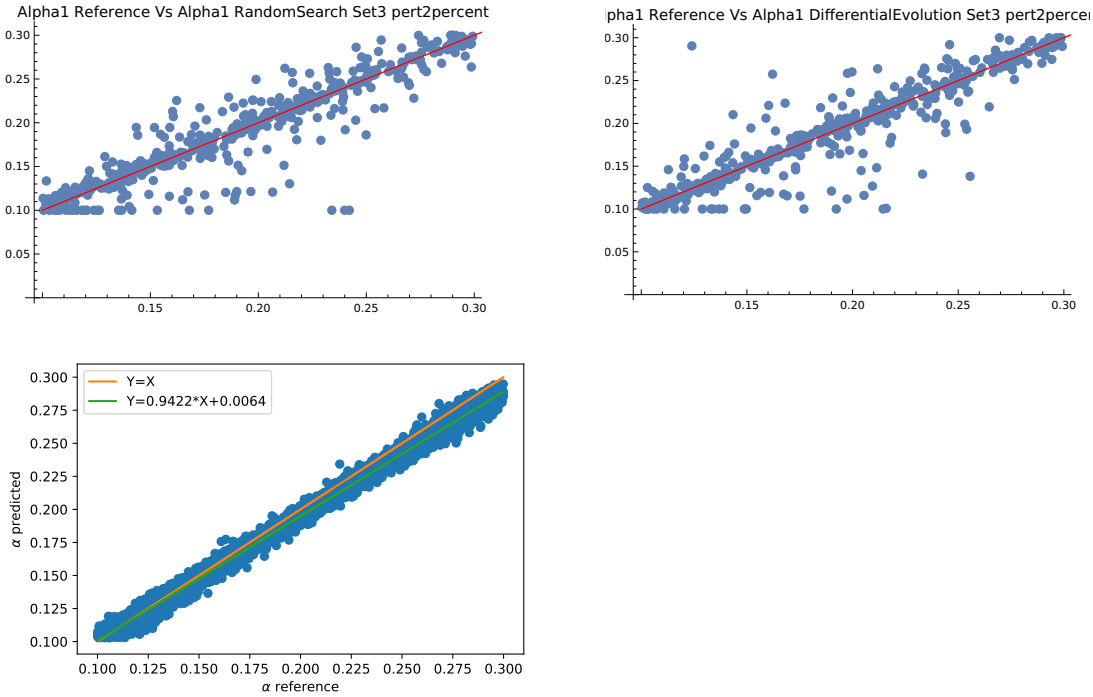

Figure 35: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using Differential Evolution algorithm on mathematica; Down and left: Prediction of α_1 using Neural Network algorithm on python; Down and right: Prediction of α_1 using Couenne algorithm 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 36, 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 $std(\epsilon_3) \sim 3.10^{-2} std(\Gamma_6)$.

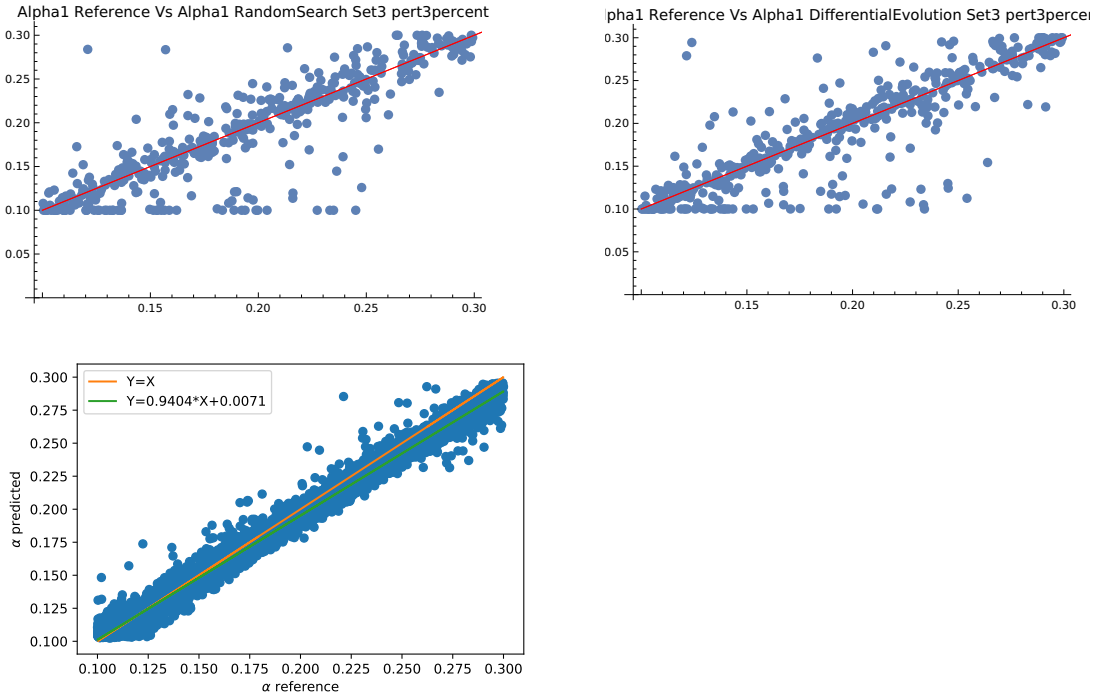

Figure 36: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using Differential Evolution algorithm on mathematica; Down and left: Prediction of α_1 using Neural Network algorithm on python; Down and right: Prediction of α_1 using Couenne algorithm 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 37, 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 $std(\epsilon_3) \sim 5.10^{-2} std(\Gamma_6)$.

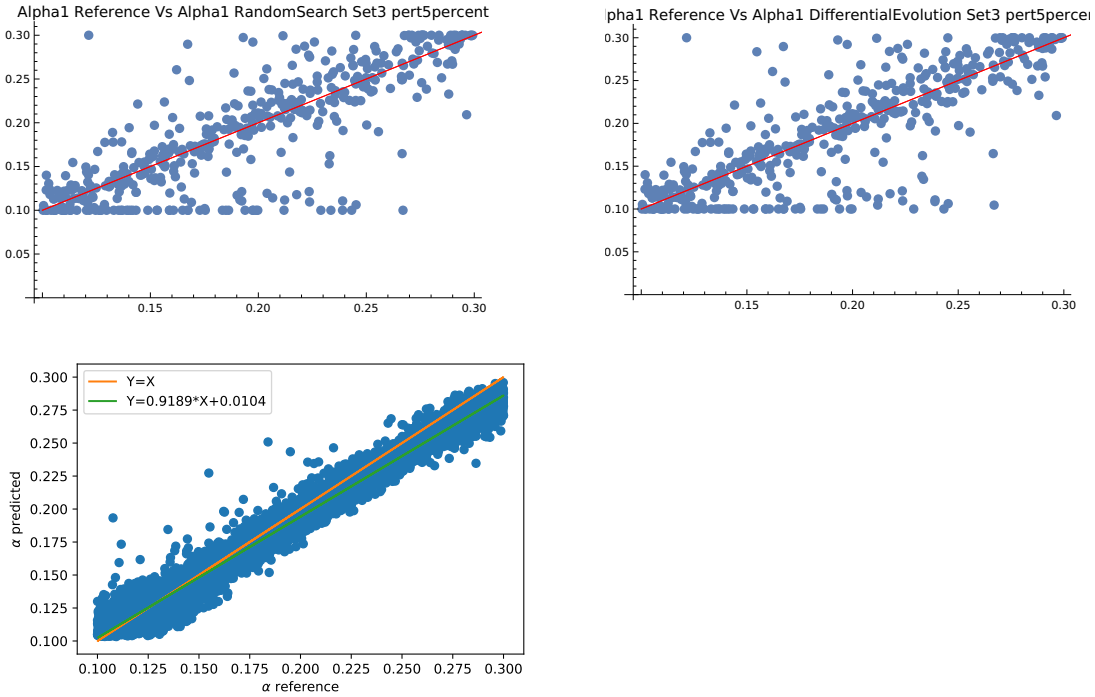

Figure 37: Up and left: Prediction of α_1 using Random Search algorithm on mathematica; Up and right: Prediction of α_1 using Differential Evolution algorithm on mathematica; Down and left: Prediction of α_1 using Neural Network algorithm on python; Down and right: Prediction of α_1 using Couenne algorithm 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 algorithms Random Search, Differential Evolution and Couenne didn't succeed to solve the Garch constrained problem for the different magnitude of the noise.
However, the neural network has shown good performance were 95 percent of the predicted data belong to the domain limited by the lines $y = x+0.1$ and $y = x-0.1$ for a standard deviation of the noise equal to 5 percent. 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.

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

3.4 Statistical data to train the network for the GARCH parameters

As the method neural network has shown good results on perturbed data, we investigate in this part the performance of the neural network method on time series, by considering estimators of the moments, this will introduce a statistical error if we suppose that our model is ergodic. Let us see the performance of a standard method the maximum likelihood method on determining the parameters of the GARCH then we will compare it to neural network results.

3.4.1 Test case using maximum likelihood method

Let us found the probability function that we want to maximise we have that

$$\sigma_t^2 = \alpha_0 + \alpha_1 x_{t-1}^2 + \beta_1 \sigma_{t-1}^2 \tag{32}$$

where the return is defined such that,

$$x_t = z_t \sigma_t \tag{33}$$

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

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

. As we want to perform a 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 parameter. Now we can write,

$$f(x_t, x_{t-1}, x_{t-2}, \ldots, x_0, \theta) = f(x_t, x_{t-1}, x_{t-2}, \ldots, x_1, \theta|_{x_0}) f(x_0, \theta) \tag{34}$$

Iterating this we obtain,

$$f(x_t, x_{t-1}, x_{t-2}, \ldots, x_0, \theta) = \Pi_{i=1}^{t} f(x_i, \theta|_{x_{i-1}, x_{i-2}, \ldots, x_1, x_0}) f(x_0, \theta) \tag{35}$$

As $f(x_i, \theta|_{x_{i-1}, x_{i-2}, \ldots, x_1, x_0})$ is the density of normal variable with variance σ_i^2 we obtain:

$$f(x_t, x_{t-1}, x_{t-2}, \ldots, x_0, \theta) = f(x_0, \theta) \Pi_{i=1}^{t} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp^{-\frac{x_i^2}{2\sigma_i^2}} \tag{36}$$

Finally we take the log to get the function to maximize:

$$log(f(x_t, x_{t-1}, x_{t-2}, \ldots, x_0, \theta)) = \sum_{i=1}^{t} \frac{1}{2}(-log(2\pi) - log(\sigma_i^2) - \frac{x_i^2}{2\sigma_i^2}) \tag{37}$$

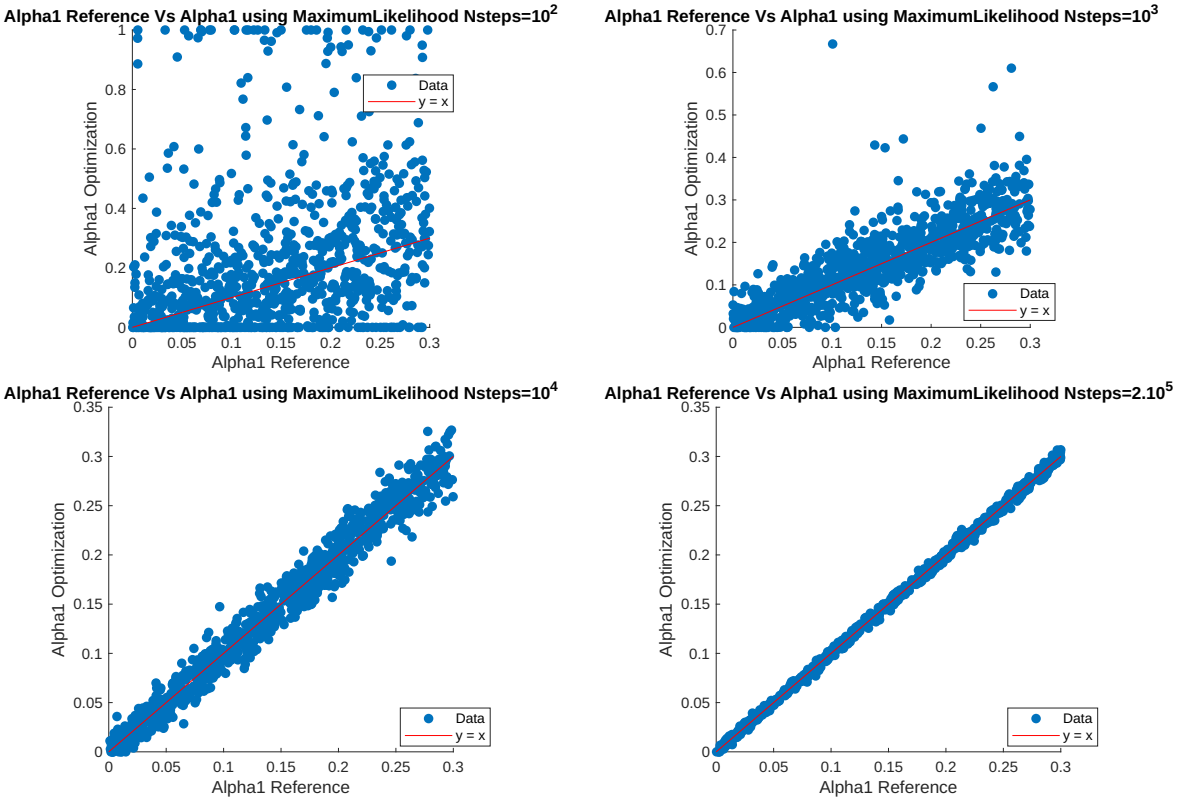

Figure 38: Up and left: Prediction of α_1 on a trajectory with total steps number $N = 10^2$; Up and right: Prediction of α_1 on a trajectory with total steps number $N = 10^3$; Down and left: Prediction of α_1 on a trajectory with total steps number $N = 10^4$; down and Right: Prediction of α_1 on a 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.

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

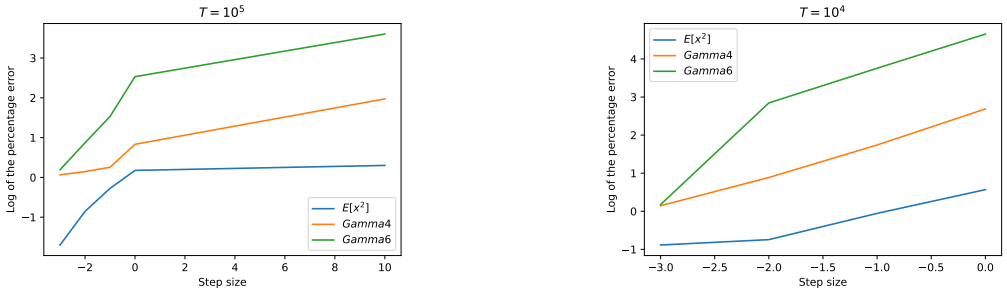

Figure 39: On the left: the logarithmic error of the percentage error on ($\mathbb{E}[x^2]$, Γ_4, Γ_6) for each time step with final time $T = 10^5$; On the right: same thing but with final time $T = 10^4$.

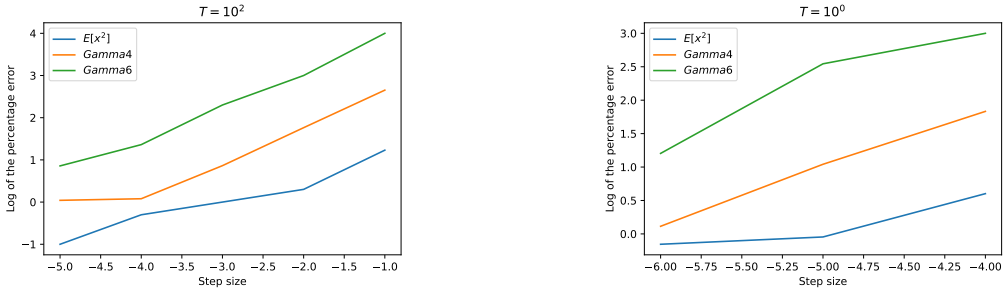

Figure 40: On the left: the logarithmic error of the percentage error on ($\mathbb{E}[x^2]$, Γ_4, Γ_6) for each time step with final time $T = 10^2$; On the right: same thing but with final time $T = 10^0$.

Message 9: The previous results on the statistical error on the training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6) show that we need a number of sampling on the trajectory higher than 10^6 to validate the ergodicity at least for the first moments and to ensure a statistical relative error less than 10 percent 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

3.4.3 Test case using a number of steps N under observed ergodicity for $\mathbb{E}[x^2]$, $\mathbb{E}[x^4]$ and $\mathbb{E}[x^6]$

In this part we investigate the performance of our neural network on the time serie, to this end, we construct estimators ($\mathbb{E}_N[x^2]$, Γ_4^N, Γ_6^N) of the quantities ($\mathbb{E}[x^2]$, Γ_4, Γ_6). As we have seen in the previous section, we can suppose that our trajectory is ergodic for certain test functions, the function x^2, x^4 and x^6, so we leverage the ergodicity principle on the trajectory to craft our estimators. This approach allows us to gain insights into the neural network's effectiveness in capturing and replicating statistical properties over the course of the time series.

We recall that in the case of an ergodic system with invariant measure p_∞ we have:

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

that we can approximate using the trajectory of $(x_t)_{t\in[0,T]}$ by:

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

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

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

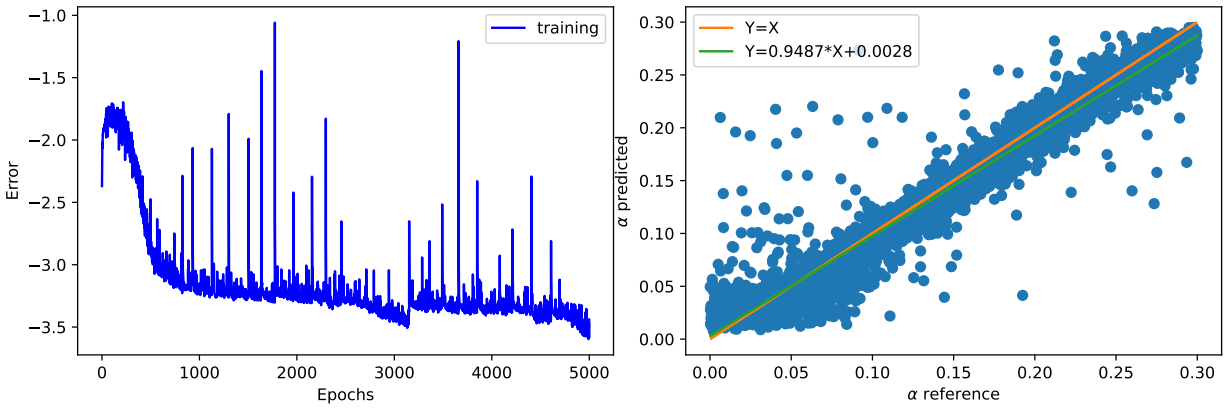

Figure 41: Left: Training error for the training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ using $N = 10^5$.

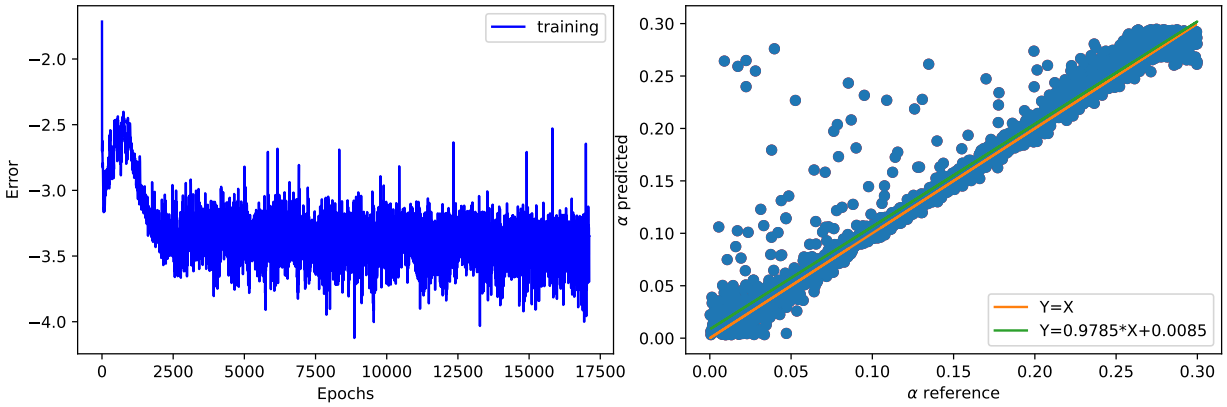

Figure 42: Left: Training error for the training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ using $N = 10^6$.

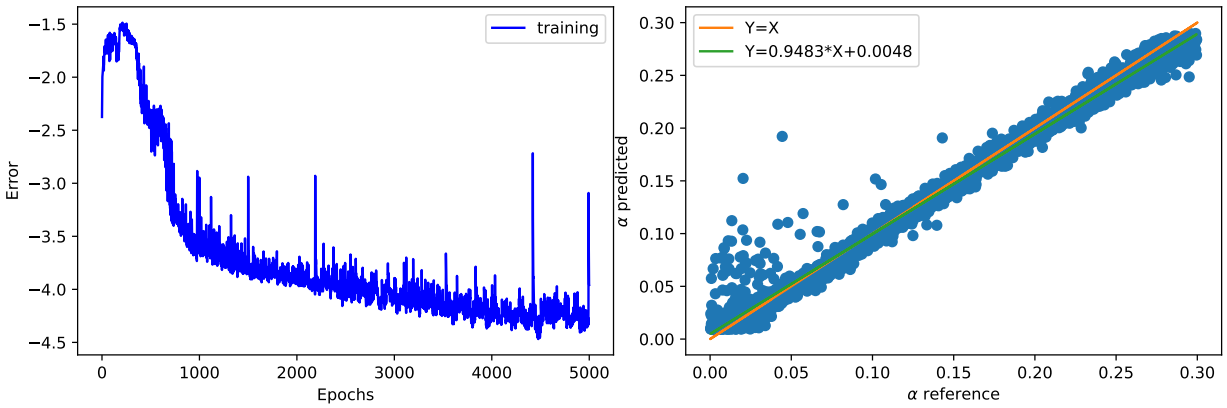

Figure 43: Left: Training error for the training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the training set $(\mathbb{E}_N[x^2], \Gamma_4^N, \Gamma_6^N)$ using $N = 10^7$.

Message 10: The preceding 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 anticipated, the augmentation of the sampling number along the trajectory leads to a reduction in statistical error, consequently yielding superior outcomes. The neural network model exhibits remarkable efficacy on the trajectory, comparable to the performance achieved by the maximum likelihood method. Notably, the computational time required for the neural network model is notably more efficient, emphasizing its practical advantages in terms of efficiency.

4 Appendix

4.1 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):

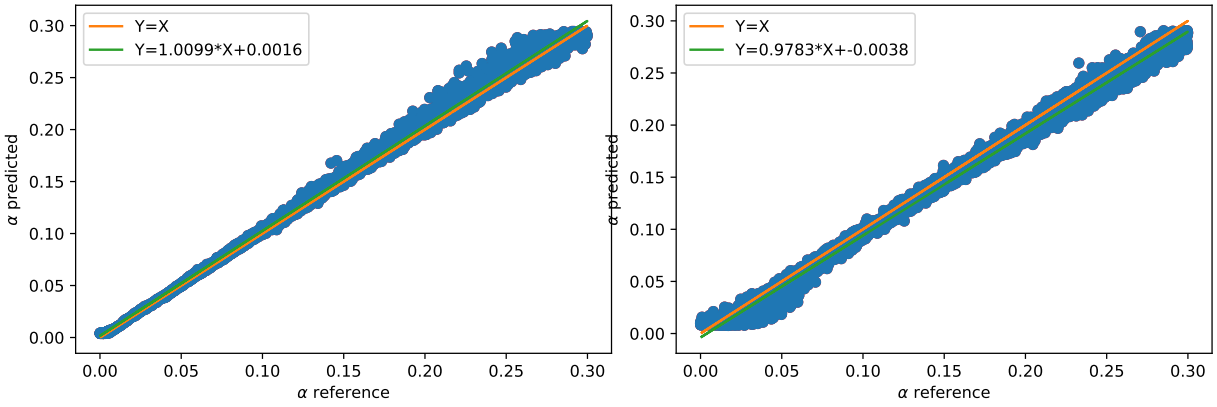

Figure 44: Left:Prediction of α_1 for the exact training set ($\mathbb{E}[x^2]$, Γ_4, Γ_6); Right: Prediction of α_1 for the perturbated training set ($\mathbb{E}[x^2]+\epsilon_1$, $\Gamma_4+\epsilon_2$, $\Gamma_6+\epsilon_3$) using $std(\epsilon_1) \sim 10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 10^{-2} std(\Gamma_6)$.

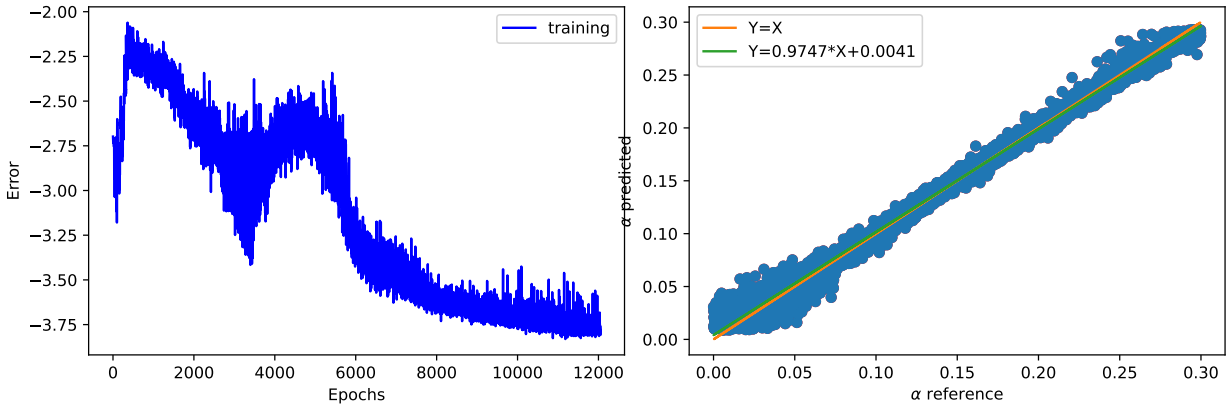

Figure 45: Left:Training error for the training set $(\mathbb{E}[x^2] + \epsilon_1,\ \Gamma_4 + \epsilon_2,\ \Gamma_6 + \epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbed training set $(\mathbb{E}[x^2] + \epsilon_1,\ \Gamma_4 + \epsilon_2,\ \Gamma_6 + \epsilon_3)$ using $std(\epsilon_1) \sim 2.10{-}2std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 2.10{-}2std(\Gamma_4)$ and $std(\epsilon_3) \sim 2.10{-}2std(\Gamma_6)$.

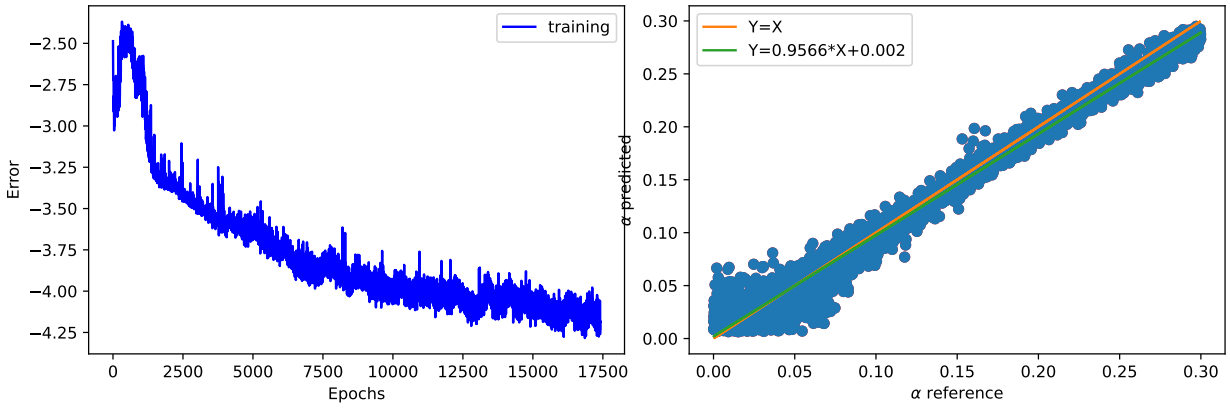

Figure 46: Left:Training error for the training set $(\mathbb{E}[x^2] + \epsilon_1,\ \Gamma_4 + \epsilon_2,\ \Gamma_6 + \epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2] + \epsilon_1,\ \Gamma_4 + \epsilon_2,\ \Gamma_6 + \epsilon_3)$ using $std(\epsilon_1) \sim 3.10^{-2}std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 3.10^{-2}std(\Gamma_4)$ and $std(\epsilon_3) \sim 3.10^{-2}std(\Gamma_6)$.

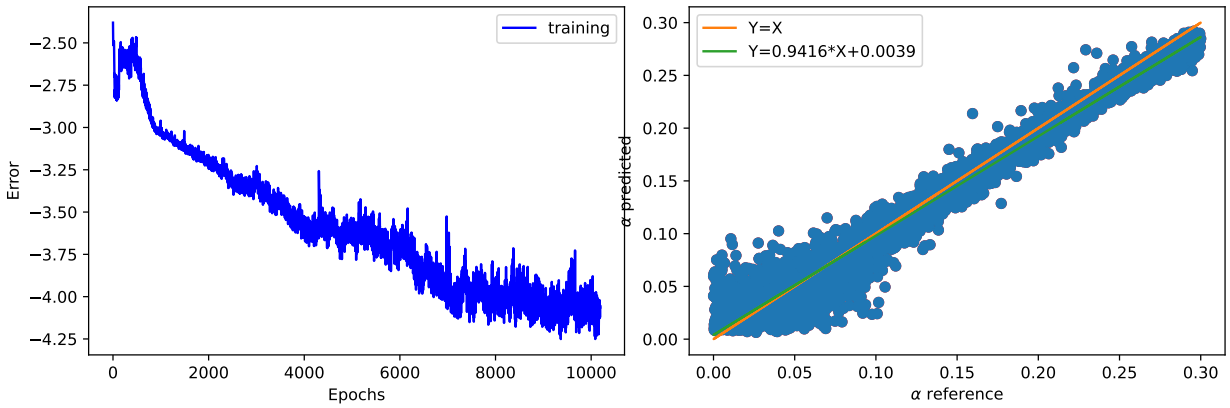

Figure 47: Left:Training error for the training set $(\mathbb{E}[x^2] + \epsilon_1,\ \Gamma_4 + \epsilon_2,\ \Gamma_6 + \epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2] + \epsilon_1, \Gamma_4 + \epsilon_2, \Gamma_6 + \epsilon_3)$ using $std(\epsilon_1) \sim 5.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 5.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 5.10^{-2} std(\Gamma_6)$.

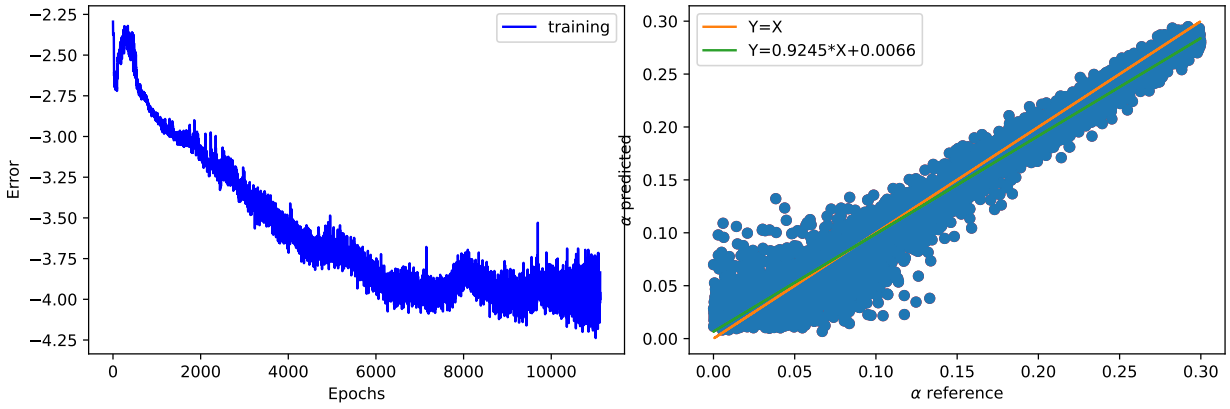

Figure 48: Left:Training error for the training set $(\mathbb{E}[x^2] + \epsilon_1,\ \Gamma_4 + \epsilon_2,\ \Gamma_6 + \epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2] + \epsilon_1, \Gamma_4 + \epsilon_2, \Gamma_6 + \epsilon_3)$ using $std(\epsilon_1) \sim 8.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 8.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 8.10^{-2} std(\Gamma_6)$.

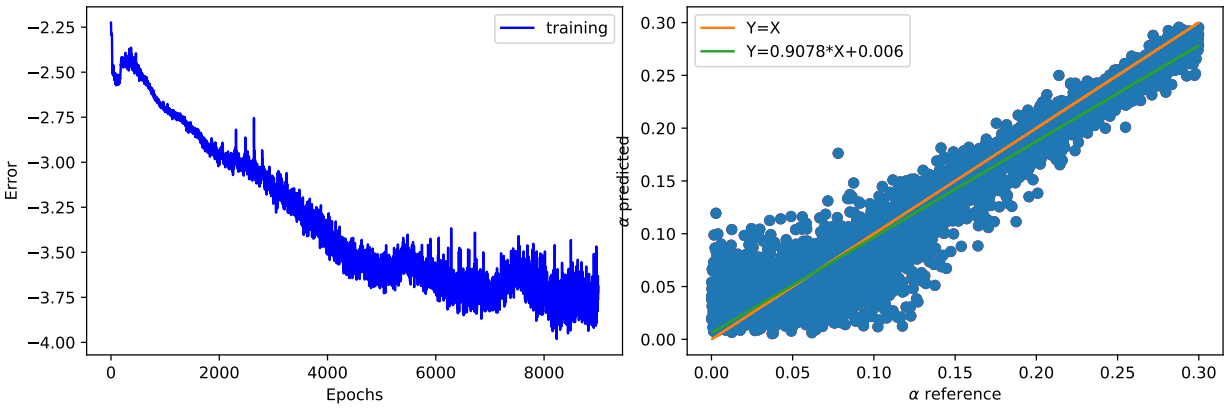

Figure 49: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 10^{-1} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 10^{-1} std(\Gamma_4)$ and $std(\epsilon_3) \sim 10^{-1} std(\Gamma_6)$.

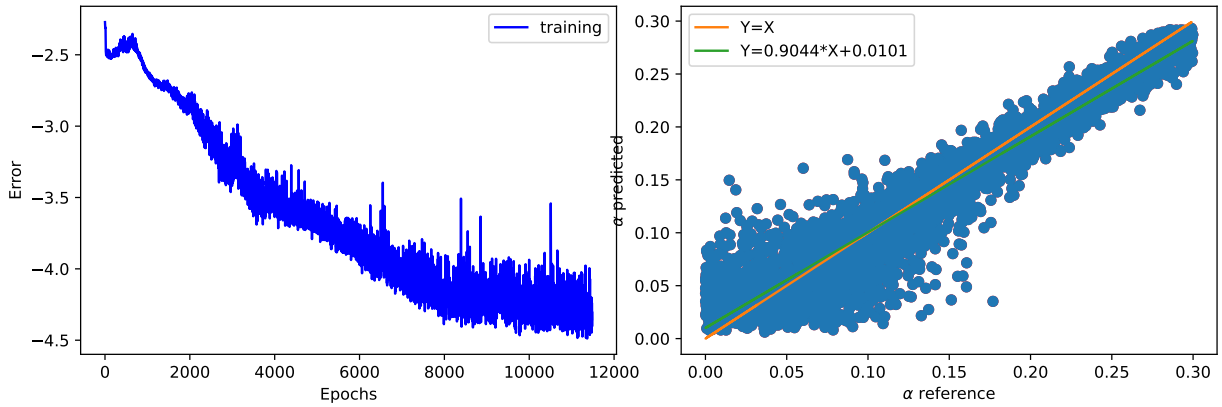

Figure 50: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 12.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 12.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 12.10^{-2} std(\Gamma_6)$.

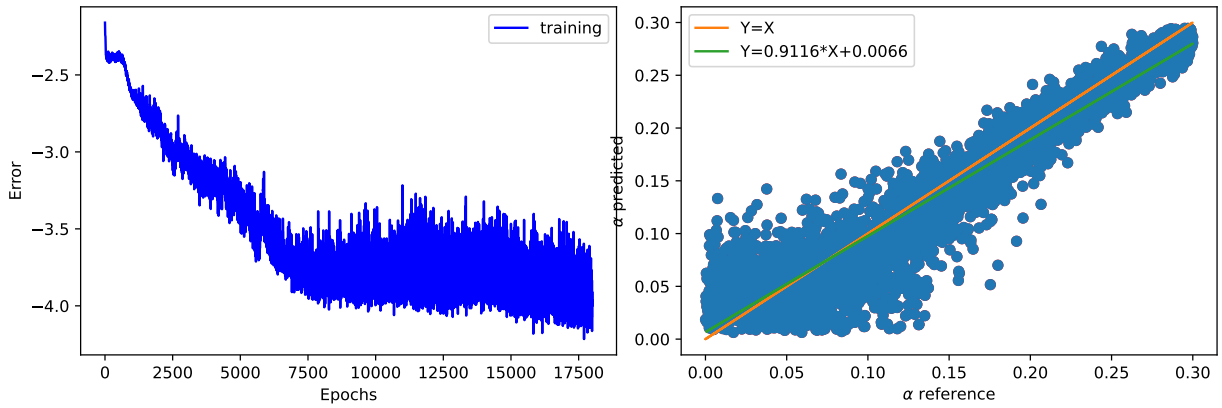

Figure 51: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 14.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 14.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 14.10^{-2} std(\Gamma_6)$.

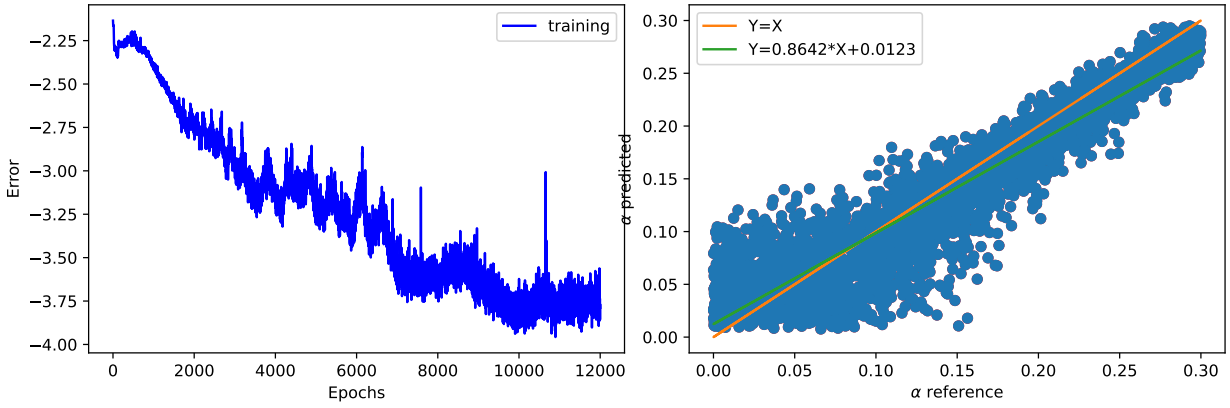

Figure 52: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 16.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 16.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 16.10^{-2} std(\Gamma_6)$.

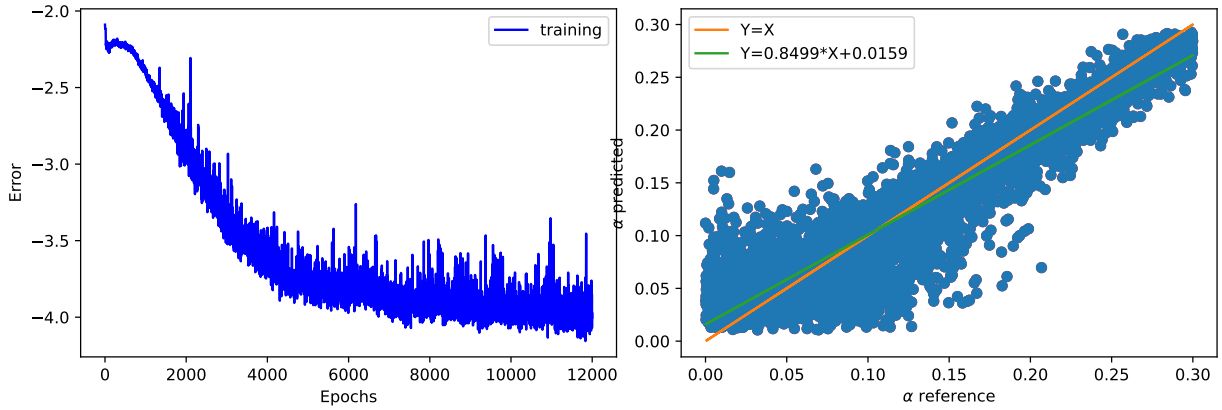

Figure 53: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 18.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 18.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 18.10^{-2} std(\Gamma_6)$.

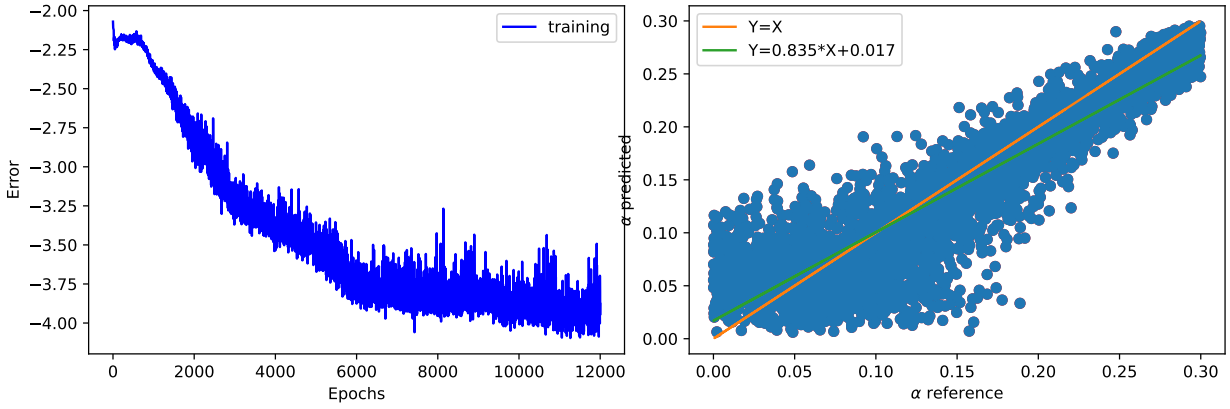

Figure 54: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 20.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 20.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 20.10^{-2} std(\Gamma_6)$.

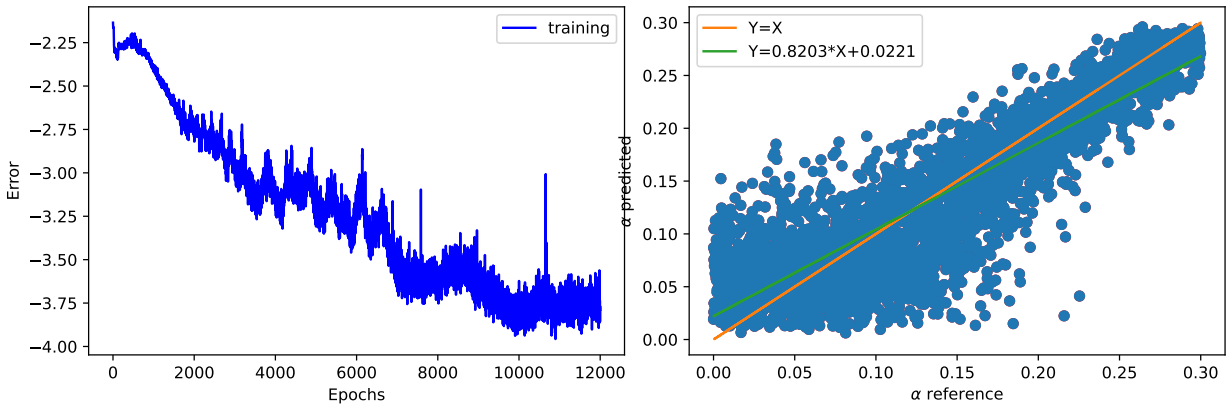

Figure 55: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ using $std(\epsilon_1) \sim 24.10^{-2} std(\mathbb{E}[x^2])$, $std(\epsilon_2) \sim 24.10^{-2} std(\Gamma_4)$ and $std(\epsilon_3) \sim 24.10^{-2} std(\Gamma_6)$.

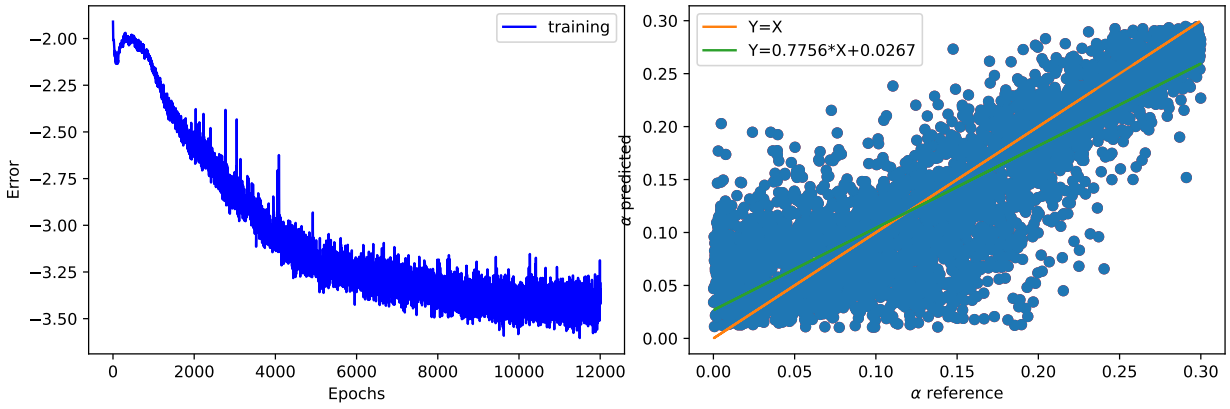

Figure 56: Left:Training error for the training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 for the perturbated training set $(\mathbb{E}[x^2]+\epsilon_1,\ \Gamma_4+\epsilon_2,\ \Gamma_6+\epsilon_3)$ 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:

$$\sigma_t^2 = \alpha_0 + \alpha_1 x_{t-1}^2 + \beta_1 \sigma_{t-1}^2 \tag{38}$$

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.

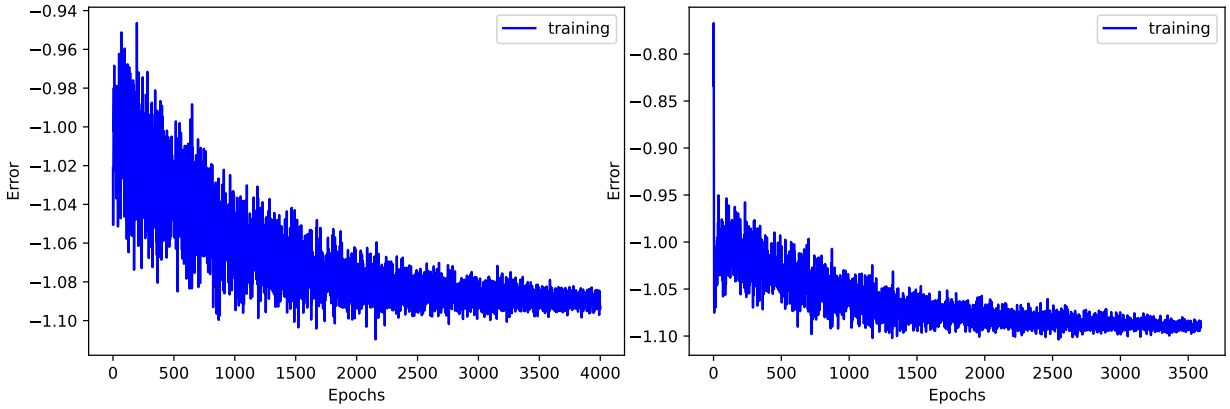

Figure 57: Left: Training error for the training set $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \hat{\gamma}_6^N)$ in logarithmic scale using dynamic learning rate and using 128 neurons in the first layer; Right: Training error for the same set using 1280 neurons in the first layer. Dynamic learning rate.

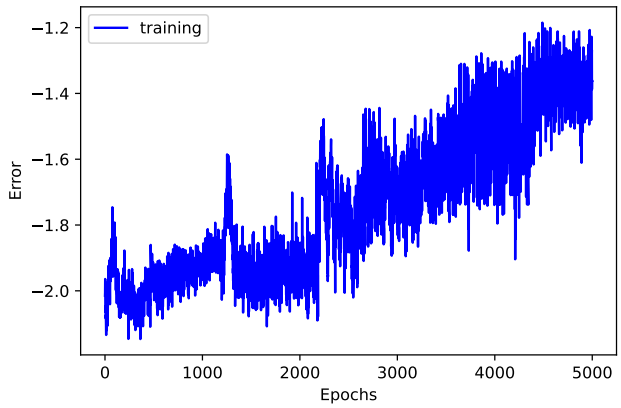

Figure 58: Training error for the training set $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \hat{\gamma}_6^N)$ in logarithmic scale using dynamic learning rate and using 128 neurons in the first layer. Fixed learning rate $lr = 10^{-4}$.

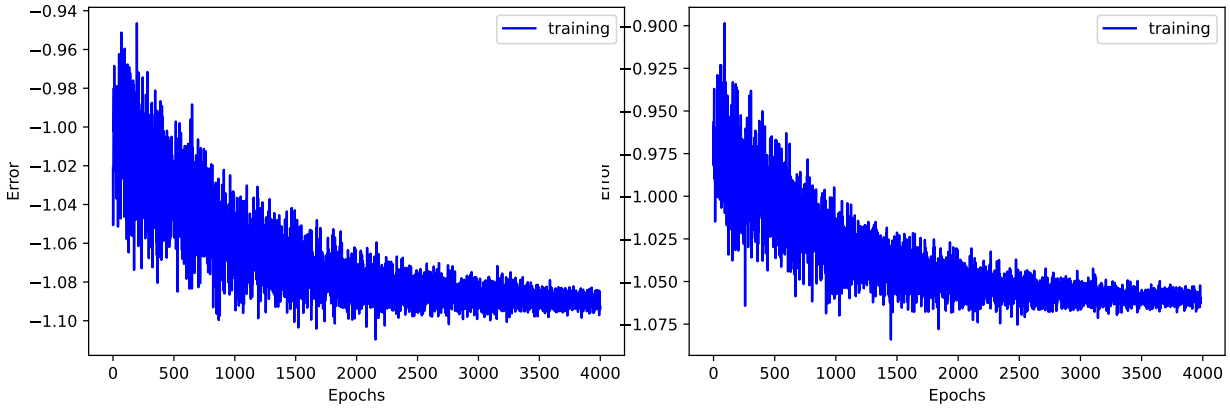

Figure 59: Left: Training error for the training set $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \hat{\gamma}_6^N)$ in logarithmic scale using dynamic learning rate and using 1280 neurons in the first layer; Right: Training error for the same neural network using the training set "Exact" $(\hat{\gamma}_2, \hat{\gamma}_4, \hat{\gamma}_6)$. Dynamic learning rate.

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

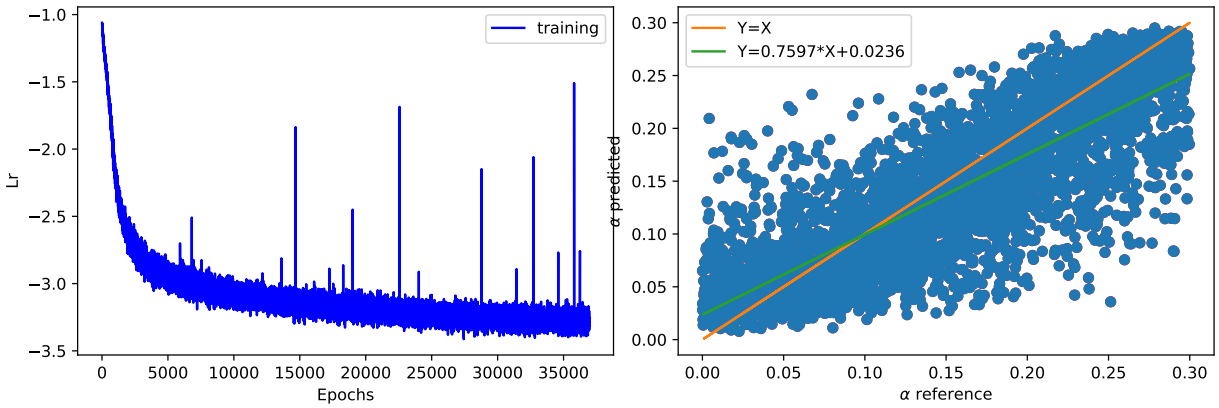

Figure 60: Left: Training error for the training set $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \hat{\gamma}_6^N, \hat{\gamma}_8^N, \lambda_1, \lambda_2, \tau_3, \tau_4, \tau_5)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 compared to α_1 of the reference. We use here 5.10^4 data.

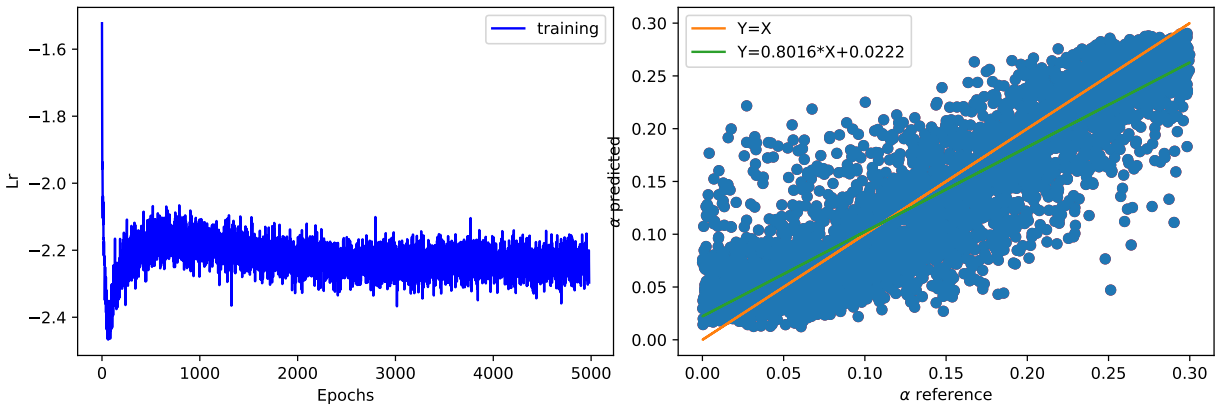

Figure 61: Left: Training error for the training set $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \tau_3, \tau_4, \tau_5)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 compared to α_1 of the reference. We use here 5.10^4 data.

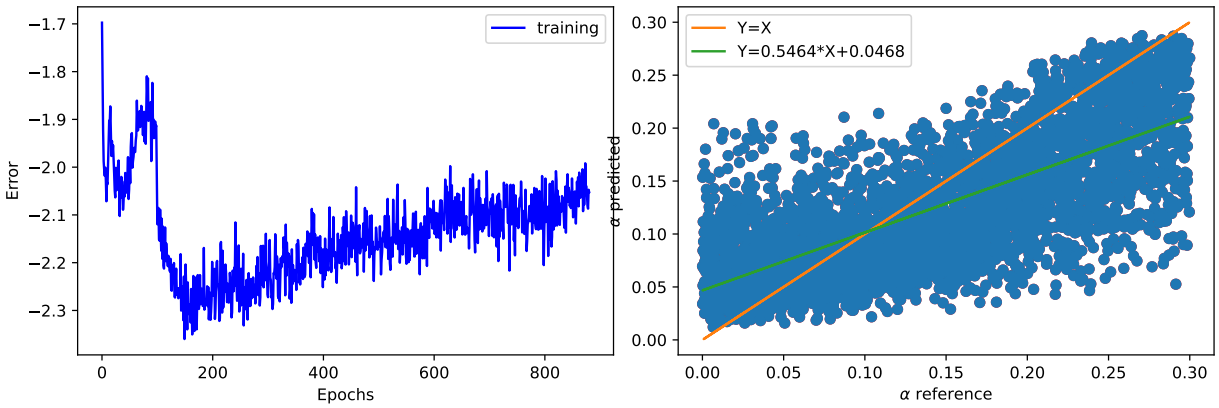

Figure 62: Left: Training error for the training set $(\hat{\gamma}_6^N, \hat{\gamma}_8^N, \tau_3, \tau_4, \tau_5)$ in logarithmic scale using fixed learning rate and using 1280 neurons in the first layer; Right: Prediction of α_1 compared to α_1 of the reference. We use here 5.10^4 data.

4.2 A Convolutional network using dynamic learning rate

As in section 3.2.3, We propose here to use a dynamic learning rate for the generator, by using two neural networks, one encapsulated in the second, where the first is the generator that calibrates α_1 and the second is the "decision maker" ,a neural network that learns to choose the best learning rate value.

For the decision maker, the neural network is unsupervised, composed by a linear layer with 4 neurons and a softmax activation function in order to consider the output as a set of probabilities on the indexed neuron. Each index will be linked to one among 4 learning rates. We aim to learn this neural network to choose the learning rate that has the better probability to get a better performance. To do so, we use the reward notion, defined in this case as equal to $reward = -generator_{loss}$ and thus the loss of decision maker is taken as

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

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

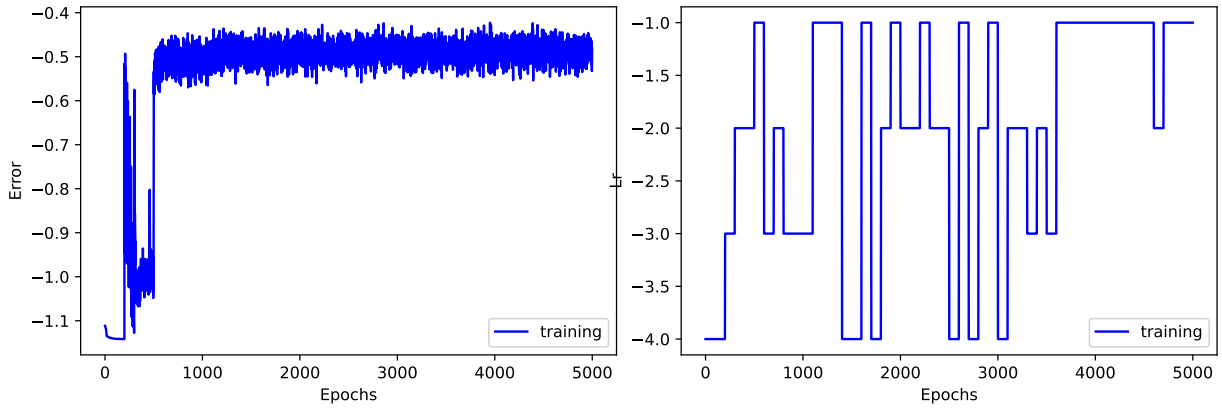

Figure 63: Left: Training error for the training set $(\hat{\gamma}_2^N, \hat{\gamma}_4^N, \hat{\gamma}_6^N)$ in logarithmic scale using dynamic learning rate and using 1280 neurons in the first layer; Right: Evolution of the learning rate during the learning phase.

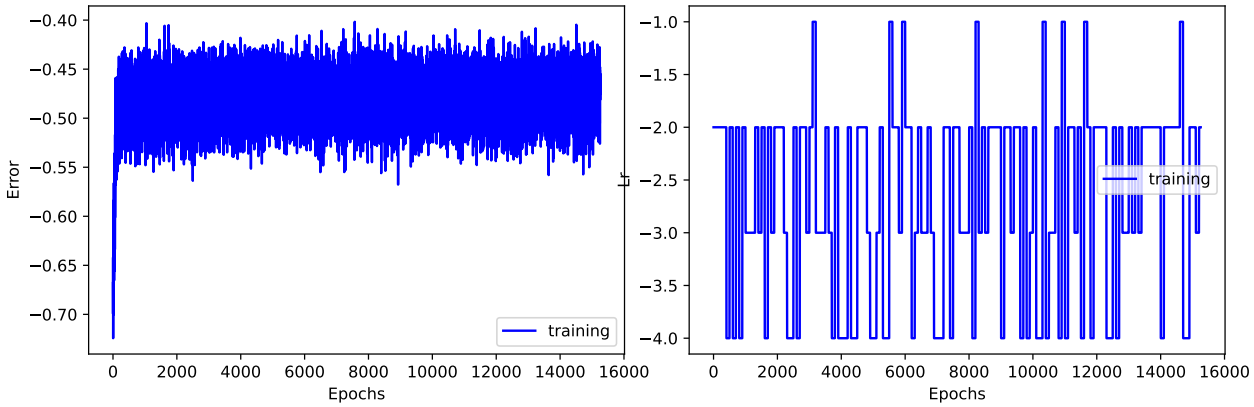

Figure 64: Left: Training error for the training set $(\hat{\gamma}_6^N, \hat{\gamma}_8^N, \tau_3, \tau_4, \tau_5)$ in logarithmic scale using adaptive learning rate and using 1280 neurons in the first layer; Right: Evolution of the learning rate during the learning phase.

5 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] Rajendra Bhatia, Tanvi Jain, and Yongdo Lim. On the bures-wasserstein distance between positive definite matrices. *arXiv*, 2017.

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

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

[4] Luigi Malagò, Luigi Montrucchio, and Giovanni Pistone. Wasserstein riemannian geometry of positive definite matrices. *arXiv*, 2018.

[5] Carl Remlinger, Joseph Mikael, and Romuald Elie. Conditional loss and deep euler scheme for time series generation. *arXiv*, 2021.