ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240784

Parameter Estimation of Long Memory Stochastic Processes with Deep Neural Networks

Bálint Csanády[a,*], Lóránt Nagy[a,b], Dániel Boros[a], Iván Ivkovic[a,b], Dávid Kovács[a], Dalma Tóth-Lakits[a], László Márkus[a,c] and András Lukács[a,*]

[a]Institute of Mathematics, ELTE Eötvös Loránd University, Budapest, Hungary
[b]Alfréd Rényi Institute of Mathematics, Budapest, Hungary
[c]Department of Statistics, University of Connecticut, Connecticut, USA

Abstract. We present a purely deep neural network-based approach for estimating long memory parameters of time series models that incorporate the phenomenon of long-range dependence. Parameters, such as the Hurst exponent, are critical in characterizing the long-range dependence, roughness, and self-similarity of stochastic processes. The accurate and fast estimation of these parameters holds significant importance across various scientific disciplines, including finance, physics, and engineering. We harnessed efficient process generators to provide high-quality synthetic training data, enabling the training of scale-invariant 1D Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) models. Our neural models outperform conventional statistical methods, even those augmented with neural network extensions. The precision, speed, consistency, and robustness of our estimators are demonstrated through experiments involving fractional Brownian motion (fBm), the Autoregressive Fractionally Integrated Moving Average (ARFIMA) process, and the fractional Ornstein-Uhlenbeck process (fOU). We believe that our work will inspire further research in the field of stochastic process modeling and parameter estimation using deep learning techniques.

1 Introduction

Long-range dependence, or long memory, plays a critical role in the scientific modeling of natural and industrial phenomena. On the one hand, in the field of natural sciences, long memory finds application in diverse areas such as climate change [44, 9], hydrology [20], detection of epilepsy [1], DNA sequencing [36], data networks [43], and in cybersecurity through anomaly detection [27]. On the other hand, research on long memory implies achievements in financial mathematics, see for example [35, 7] and [2] for the application of long memory in volatility modeling. The ubiquity of long memory across time series data has drawn considerable attention to models capable of capturing this phenomenon. In most stochastic models, the impact of past events on future events decays rapidly, making the effect of observations from the distant past negligible in terms of forecasting ability. If long-range dependence is present in a system, predictions concerning the future require information from the complete history of the process, in contrast to Markovian environments, where the most recent events already contain all the necessary information for an optimal forecast. Modern literature associates long-range dependence with a slow hyperbolic decay, namely when autocorrelations ρ_t vanish with a rate that is comparable to $t^{-\alpha}$ for some $\alpha \in (0,1)$ [12]. The range $\alpha \in (1,2)$ is often referred to as anti-persistence. In broader terms, "long memory" can be used to mean that autocorrelations have a longer decay than exponential.

When one models data with long memory, it is a crucial task to estimate model parameters, and classical inference methods are often not applicable in the case of long memory processes. We focus our attention on three stochastic processes that are frequently utilized in modern applied mathematics: the fractional Brownian motion (fBm), the Autoregressive Fractionally Integrated Moving Average (ARFIMA), and the fractional Ornstein-Uhlenbeck (fOU) process. In the case of fBm and fOU, we concentrate on estimating the Hurst parameter. The Hurst exponent controls the roughness, self-similarity, and long-range dependence of fractional Brownian motion paths, and this way also influences the characteristics of derivative processes such as the fractional Ornstein-Uhlenbeck process. Regarding ARFIMA models, our target is the differencing parameter d, governing the decay of autocovariances, and thus, the decay of memory in the system.

1.1 Main results

We propose the adoption of well-known neural network architectures as versatile tools for estimating the parameters that characterize long memory. A key aspect of our approach is that efficient process generators provide the opportunity to train the models on large amounts of data. This facilitates better performance compared to traditional statistical estimation methods, even those supplemented with neural networks. Inference is efficient in terms of speed, even for samples of long sequences, making our approach valuable for practical applications. We can also guarantee various invariances such as shift, drift, and scale-invariance. A further advantage of our method is that it performs consistently across the entire $[0,1]$ range of the Hurst parameter. We found that the proposed neural network estimator improves in accuracy when trained on longer sequences, a consistency that persists even during inference on sequences of lengths different from those used in training. The above virtues are supported by the measurements presented in the paper. Our code is publicly available on GitHub.[1]

* Corresponding authors: csbalint@protonmail.ch, andras.lukacs@ttk.elte.hu

[1] https://github.com/aielte-research/LMSParEst

1.2 Related work

In recent years, a number of studies have emerged that utilize neural networks to estimate the Hurst parameter of fBM. Some of these works apply MLPs which, due to their fixed-size input requirement, present one of the following alternatives: inference can either be performed only on a fixed-length series [26, 15], or on a set of process specific statistical measures enabling the fixed-size input to the neural networks [23, 30]. A more general approach is the signature-based method described in [21], which can also be used to estimate fBM Hurst, where the extracted statistical descriptors are processed by an LSTM. In the case of these methods, the hybrid application of statistical descriptors and neural networks brings less improvement compared to our purely neural network solutions. This is reflected in the comparison to several classical estimation methods. Another shortcoming in recently published methods is that they do not address the possible limitations caused by scaled inputs. Regarding the ARFIMA and fOU processes, so far, we could not find neural network based parameter estimators in the literature. We note that our estimator's architecture is close to the work by Wang et al. [41] which was used for the non-fractional Ornstein-Uhlenbeck process. A more complex architecture in subsequent work by Feng et al. [8] estimates the Hurst parameter in fBm. In the case of the fOU process, we compared the neural network-based Hurst estimates with a quadratic variation estimator [4], and our method presented much higher accuracy. For the ARFIMA process Whitte's method [42] yielded comparable performance to our approach.

The success of the utilized networks (Section 4.3) mostly stems from a large volume of high-quality training data, manifested with the software that was built around the framework of the so-called isonormal processes (see [33] for the mathematical background, and Section 4.2 on the implementation). The underlying path-generating methodology includes the circulant embedding of covariance matrices and the utilization of fast Fourier transform.

2 Processes

In this section the most important properties of the stochastic processes involved in this paper are summarized.

2.1 Fractional Brownian motion

The fractional Brownian motion $\mathrm{fBm}(H) := \left(B_t^H\right)_{t\geq 0}$, $H \in (0,1)$ is a continuous centered Gaussian process with covariance function

$$\mathrm{Cov}\left(B_t^H, B_s^H\right) = \frac{1}{2}\left(|t|^{2H} + |s|^{2H} - |t-s|^{2H}\right).$$

Here, H is called the Hurst exponent of the process. Let $\mathrm{FBM}(H,n,S,T) : S < T$ denote the distribution of the $(T-S)/n$-equidistant realizations of $\mathrm{fBm}(H)$ on the time interval $[S,T]$. If $\Delta\,\mathrm{FBM}(H,n,S,T)$ denotes the sequence of increments, it can be shown that $\Delta\,\mathrm{FBM}(H,n,S,T) \sim \lambda(H,T-S)\Delta\,\mathrm{FBM}(H,n,0,1)$ where $\lambda(H,T-S)$ is a scalar. If we want to estimate H from $\mathrm{FBM}(H,n,S,T)$ we might want to consider a shift invariant neural network on the increments, since then it will be sufficient to train it only on $\mathrm{FBM}(H,n,0,1)$. We might also consider the scaled and drifted fBm process $\mathrm{fBm}(H,\sigma,\mu) := \left(\sigma B_t^H + \mu t\right)_{t\geq 0}$, with $H \in (0,1), \sigma > 0, \mu \in \mathbb{R}$, which is the fractional counterpart of the so called Bachelier model [31]. When the network is also drift invariant, it is still sufficient to train the network on realizations of $\mathrm{FBM}(H,n,0,1)$ to yield an estimator for the parameter H of $\mathrm{fBm}(H,\sigma,\mu)$.

2.2 Autoregressive Fractionally Integrated Moving Average

A covariance stationary sequence of random variables $\zeta_j,\ j \in \mathbb{Z}$ is said to form white-noise if $E\zeta_0 = 0$, $\gamma(0) = E\zeta_0^2 < \infty$, and $\gamma(k) = 0$ for all $k \in \mathbb{Z},\ k \neq 0$. For $d > -1$ the fractional difference operator is defined as

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

where B is the backward shift operator, that is $BX_j = X_{j-1}$.We define the $\mathrm{ARFIMA}(0,d,0)$ process for $d \in (-1/2, 1/2)$ as the solution of the difference equation

$$\nabla^d X_j = \zeta_j, \tag{1}$$

where $\zeta_j,\ j \in \mathbb{Z}$ is a white-noise sequence. It is known that when $\zeta_j,\ j \in \mathbb{Z}$ is ergodic (e.g. in the sense of Definition 2.8.4 in [11]), and $d \neq 0$, there exists a unique stationary solution to Equation (1) – see Theorem 7.2.2 in [12].

2.3 The fractional Ornstein-Uhlenbeck process

Let $H \in (0,1), \alpha, \sigma > 0, \eta, \mu \in \mathbb{R}$. The fractional Ornstein-Uhlenbeck process $(Y_t)_{t\geq 0}$ is the solution of the following stochastic differential equation:

$$\begin{aligned} dY_t &= -\alpha(Y_t - \mu)\,dt + \sigma\,dB_t^H \\ Y_0 &= \eta. \end{aligned}$$

Let $\mathrm{fOU}(\eta, H, \alpha, \mu, \sigma)$ denote the distribution of this process on the Borel σ-algebra of continuous functions. Note that μ and σ are scaling and shifting parameters. If $Y \sim \mathrm{fOU}\left((\eta-\mu)/\sigma, H, \alpha, 0, 1\right)$, then $\sigma Y + \mu \sim \mathrm{fOU}(\eta, H, \alpha, \mu, \sigma)$. This means that if we can guarantee the scale and shift invariance of the network, it will be sufficient to train a H-estimator on realizations from $\mathrm{fOU}\left(\eta, H, \alpha, 0, 1\right)$ to cover the distribution on $\mathrm{fOU}(\eta, H, \alpha, \mu, \sigma)$.

3 Baseline estimators

To provide baseline comparisons to our neural network based results we considered the following estimators.

3.1 Rescaled range analysis

The term and concept of rescaled range analysis stems from multiple works of Harold E. Hurst - see e.g. [20], a study in hydrology, and for a historical account on the methodology see [14]. The statistics R/S, defined below, is the rescaled and mean adjusted range of the progressive sum of a sequence of random variables, more precisely, given $Z_1, Z_2, ...$, for a positive integer n consider the statistics

$$R/S(n) = \frac{\max\limits_{1\leq k\leq n}\left\{X_k - \frac{k}{n}X_n\right\} - \min\limits_{1\leq k\leq n}\left\{X_k - \frac{k}{n}X_n\right\}}{\sqrt{\frac{1}{n}\sum\limits_{k=1}^{n}\left(Z_k - \frac{1}{n}X_n\right)^2}}, \tag{2}$$

where $X_k = \sum_{i=1}^{k} Z_i$. The analysis is done via assuming an asymptotics for the statistics in Equation (2), namely we postulate that on the long run, it holds that $R/S(n) \approx cn^h$, where c is an unknown constant and h is the parameter we are looking for. Utilizing a one

parameter log-log regression on the above formula, that is, using the relation $\log(R/S(n)) = \log(c) + h\log(n)$, one can estimate h.

Turning to fractional Brownian motion, it is shown in [37], that its increment process, fractional Gaussian noise has the property $R/S(n) \approx c_0 n^H$, where H is the Hurst parameter of the underlying fractional process, and c_0 is some positive constant: yielding a numerical method for the estimation of H.

A known limitation of this methodology, when the underlying process is a fractional Brownian motion, is that using the statistics in (2) produces inferred values that are lower when the true value of H is in the long memory range, and substantially higher values when the time series shows heavy anti-persistence. A possible mitigation of this is to introduce a correction that calibrates the method to fit fractional Brownian motion data and use the corrected estimator as a baseline.

3.2 Variation Estimators

For a stationary increment Gaussian process, consider the lag t second order statistics defined by

$$\gamma_p(t) = \frac{1}{2}\mathbf{E}|X_t - X_0|^p. \tag{3}$$

This, when setting $p = 2$, is also called the variogram of order 2. Assume that the behavior of $\gamma_2(t)$ at the coordinate origin (that is the asymptotics when $t \to 0$) is like that of $|t|^\alpha$, for some fractal index $\alpha \in (0, 2]$. Then, the relationship between the Hausdorff dimension and the fractal index,

$$D = 2 - \frac{\alpha}{2}$$

holds – assuming a topological dimension of 1 – and performing a log-log regression yields an estimate for D.

However, the relationship between the Hausdorff dimension and the fractal index appears to be more robust and inclusive when we choose $p = 1$. In this case, when similar asymptotics holds for $\gamma_1(t)$, estimators presented e.g. in [13] give statistically efficient procedures to determine the Hausdorff dimension of Gaussian processes. In our work, the Hurst parameter of fractional processes is then estimated using the relationship $D = 2 - H$. For detailed notes on how γ is approximated and the optimality of the choice $p = 1$ we refer to [13].

3.3 Higuchi's method

Higuchi's method [18] relies on the computation of the fractal dimension by a one dimensional box counting. For a sliding box size $b \in \mathbb{N}$ and a starting point $i \in \mathbb{N}, i \leq b$, consider

$$L_b(i) = \frac{1}{\left[\frac{n-i}{b}\right]} \sum_{k=1}^{\left[\frac{n-i}{b}\right]} \left|X_{i+kb} - X_{i+(k-1)b}\right| .$$

Then let $L_b := \frac{1}{b}\sum_{i=1}^{b} L_b(i)$. If $X \sim \mathrm{fBm}(H, 0, \sigma)$ then $E(L_b) = cb^H$ holds. Thus, the slope coefficient of the linear regression $\log(L_b) \sim \log(b)$ yields an estimate for H.

3.4 Whittle's method

The likelihood procedure dubbed as Whittle's method, see e.g. [42], is based on approximating the Gaussian log-likelihood of a sample of random variables $X = (X_1, ..., X_n)$, where the underlying process is stationary and Gaussian. We give the details in the case when we wish to estimate the Hurst parameter of fractional Brownian motion with Hurst parameter $H \in (0, 1)$, and we apply Whittle's method on its increments. Denoting with Γ_H the covariance matrix corresponding to the vector X, the likelihood of the sample with respect to H can be written as $L(X) = (2\pi)^{-n/2} |\Gamma_H|^{-1/2} e^{-\frac{1}{2}X^T\Gamma_H^{-1}X}$, where $|\Gamma_H|$ and Γ_H^{-1} denotes the determinant and the inverse of the matrix Γ_H respectively, and X^T denotes the transpose of the vector X. To speed up the procedure, instead of numerical computations, an approximation can be introduced, see e.g. [3], and the Hurst parameter H can be approximated by minimizing the quantity

$$Q(H) = \int_{-\pi}^{\pi} \frac{I(\lambda)}{f_H(\lambda)} d\lambda, \tag{4}$$

where $I(\lambda)$ is the periodogram, an unbiased estimator of the spectral density f_H, defined as $I(\lambda) = \sum_{j=-(n-1)}^{n-1} \hat{\gamma}(j)e^{ij\lambda}$, with the complex imaginary unit i, and where the sample autocovariance $\hat{\gamma}(j)$, using the sample average $\bar{X} = \frac{1}{n}\sum_{k=1}^{n} X_k$, is $\hat{\gamma}(j) = \sum_{k=0}^{n-|j|-1} \left(X_k - \bar{X}\right)\left(X_{k+|j|} - \bar{X}\right)$. The quantity in Equation (4) is usually approximated with the sum $\tilde{Q}(H) = \sum_{k=1}^{\lfloor n/2 \rfloor} \frac{I(\lambda_k)}{f_H(\lambda_k)}$, with $\lambda_k = \frac{2\pi k}{n}$, to obtain an asymptotically correct estimate $\hat{H}$ of the Hurst parameter H.

3.5 The case of ARFIMA and fOU

According to Theorem 7.2.1 in [12], for the autocovariance of an ARFIMA process, we have $\gamma(k) \approx c_d k^{2d-1}$. Thus, in terms of the decay of autocovariance and memory properties (see Definition 3.1.2 in [12]), the ARFIMA$(0, d, 0)$ process corresponds to a fractional noise with Hurst parameter $H = d + 1/2$. Also, an ARFIMA process, in an asymptotic sense, has similar spectral properties to that of fractional Brownian motion incremets. On one hand this means, that an ARFIMA process offers a potential way to test estimators calibrated to fractional Brownian motion. On the other hand, it is reasonable to apply the above baseline Hurst parameter estimators for estimating the parameter d of ARFIMA$(0, d, 0)$.

To estimate the Hurst parameter of a fractional Ornstein-Uhlebeck process, Brouste and Iacus [4] provide a statistical method (QGV) that is based on building an estimator that compares generalized quadratic variations corresponding to certain filtered versions of the input samples. The method presents consistent and asymptotically Gaussian estimators, and can be considered a state of the art analytical tool regarding Hurst parameter inference on fOU processes.

4 Methods

4.1 Training paradigm

In contrast to a situation characterized by a limited amount of data, we can leverage synthetic data generators to train our neural network models on a virtually infinite dataset. The loss computed on the most recent training batches simultaneously serves as a validation loss, as each batch comprises entirely new synthetic data. This setup prevents overfitting in the traditional sense. The only potential issue associated with this training paradigm is the quality of the process generator itself. If the generator fails to approximate the target distribution effectively, there is a potential risk of "overfitting" on generated distribution and not generalizing. Thus, high-quality process generators are essential.

Our setup to obtain parameter estimators by utilizing generators for given families of stochastic processes is as follows: Let Θ be the

set of the possible parameters and P be the prior distribution on Θ. For a fixed $a \in \Theta$, the generator G^a denotes an algorithm that generates sample paths of a stochastic process, where the sample paths are distributed according to the process distribution Q_a. This is deterministic in the sense that every iteration, returns a sample path. The generated sample path however can be treated as a random object, by introducing randomness into the parameter a by setting $a = \vartheta$, where ϑ is a random variable distributed according to some law P. Denote the compound generator by $G^{(\vartheta)}$. Now, suppose we have $G^{(\vartheta)}$ as input and we would like to estimate ϑ. Formally, an optimal M estimator would minimize the MSE $E[M(G^{(\vartheta)}) - \vartheta]^2$. By having independent realizations of series from $G^{(\vartheta)}$, we can consider the training set T. Training a proper neural network $\mathcal{M}$ on T with the squared loss function would be a heuristic attempt to obtain the above M estimator. We may assume that Q is only parameterized by the target a without loss of generality. If Q is parameterized by other parameters besides a, then these can be randomized obtaining a redefined Q_a mixed distribution.

4.2 Generating fractional processes

To generate the fractional processes fBM and fOU, we employed the circular matrix embedding method belonging to the Davies-Harte procedure family [6]. In the available Python packages [5], the original Davies-Harte method for generation is accessible. However, the generation procedure we use is based on Kroese's method [25], which we have re-implemented specifically for generating sequences using the most efficient tools within the Python framework. In our implementation, for multiple sequences, we store the covariance structure so that it does not need to be recomputed each time it is needed. Additionally, the modified version of the traditional Cholesky method is available in the implemented package, which yields a similar level of acceleration for generating large quantities of data, comparable to the currently available solutions.

4.3 Neural network architecture

The neural network architecture $\mathcal{M}$ is designed for regression on a sequential input, and consists of three main components: an optional standardizing layer to ensure invariances, a sequence transformation, and a regression head. There are three kinds of invariances that we might require from $\mathcal{M}$: shift, scale, and drift invariance. In order to make an fBm Hurst-estimator which works well in practice, we want to rely on all three of the above invariances. Shift invariance can be obtained by transforming the input sequence to the sequence of its increments. Differentiating the input this way also turns drift invariance to shift invariance. By performing a standardization on the sequence of increments we can ensure all three invariances. Such a standardizing step can also be considered as a preprocessing layer to the network, applying the transformation $x \mapsto (x - \overline{x})/\hat{\sigma}(x)$ to each sequence x in the input batch separately, where $\hat{\sigma}(x)$ is the empirical standard deviation over the sequence x, and $\overline{x}$ is the arithmetic mean over x. Given the ergodicity properties of the underlying processes, the term "standardizing" remains adequate even with co-dependent data points as inputs. We considered two alternatives for sequence transformation. In $\mathcal{M}_{\text{conv}}$ the transformation is realized by a 1D CNN [22], and in $\mathcal{M}_{\text{LSTM}}$, it is achieved by an LSTM [19]. The regression head first takes the transformed sequence and applies global average pooling resulting in an average feature vector which has a fixed dimension (the same as the output feature size in the sequence transformation). The head then obtains the final scalar output from the average feature vector using an MLP [17].

We found that unless it is severely underparameterized, the specific hyperparameter configuration does not have a significant effect the performance of $\mathcal{M}$. Generally $\mathcal{M}_{\text{LSTM}}$ achieves a somewhat smaller loss than $\mathcal{M}_{\text{conv}}$, while $\mathcal{M}_{\text{conv}}$ is computationally faster than $\mathcal{M}_{\text{LSTM}}$. Slight differences can arise in the speed of convergence, but these are not relevant due to the unlimited data and fast generators. The following are the hyperparameters that we used in the experiments below.

We implement $\mathcal{M}_{\text{conv}}$ using a 1D convolutional network with 6 layers. The input has 1 channel, and the convolution layers have output channels sizes of 64, 64, 128, 128, 128, and 128. Every layer has stride 1, kernel size 4 and no padding. The activation function is PReLU after each layer. Our $\mathcal{M}_{\text{LSTM}}$ consists of an unidirectional LSTM with 2 layers, its input dimension is 1 and the dimension of its inner representation is 128. In both models, the regression head uses an MLP of 3 layers (output dimensions of 128, 64 and 1), with PReLU activation function between the first two layers. Unless stated otherwise the models also include a standardizing layer to ensure shift and scale invariance, and when drift invariance necessary, the model also contains a finite differentiation step. AdamW optimization on the MSE loss function was used for training the models [28]. The learning rate was set to 10^{-4} and the train batch size to 32.

4.4 Technical details

The process generators were implemented in Python [39], using NumPy [16] and SciPy [40]. We imported Higuchi's method from the package AntroPy [38]. The R/S method was imported from the package hurst [29]. We generated the ARFIMA trajectories using the arfima package [24]. The framework responsible for the training process was implemented in PyTorch [34]. Every neural module we used was readily available in Pytorch. We managed our experiments using the experiment tracker [32]. The neural models were trained on Nvidia RTX 2080 Ti graphics cards. Training took approximately one GPU hour to one GPU day per model, depending on the type of process, the applied architecture, and on the length of sequences used for training. A shorter training time can mostly be expected from the acceleration (parallelization) of the sequence generation.

5 Experiments

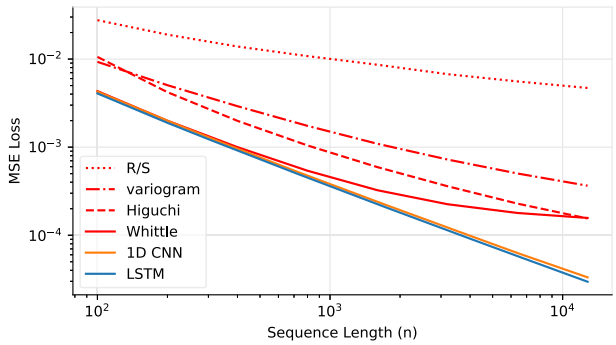
5.1 Metrics

In addition to the standard MSE loss we use two metrics. For a H-estimator M, let $b_\varepsilon(x) = m_\varepsilon(x) - x$ be the empirical bias function of radius ε, where $m_\varepsilon(x)$ is the average of estimations the estimator M produces for the input sequences with $H \in [x - \varepsilon, x + \varepsilon]$. Similarly, let the function $\sigma_\varepsilon(x)$ be defined as the empirical standard deviation of the estimations M produces for inputs inside the sliding window of radius ε.

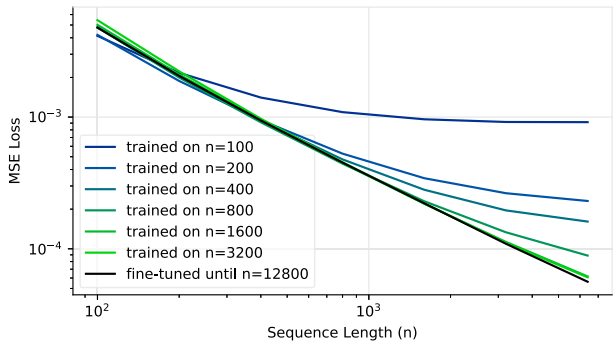
Let us denote the approximate absolute area under the Hurst–bias curve by $\hat{b}_\varepsilon := \varepsilon \sum_{j=0}^{[1/\varepsilon]} |b_\varepsilon(\varepsilon j)|$, and the approximate area under the Hurst–σ curve by $\hat{\sigma}_\varepsilon := \varepsilon \sum_{j=0}^{[1/\varepsilon]} \sigma_\varepsilon(\varepsilon j)$. We use these two metrics in addition to MSE, as they highlight different aspects of the estimators performance.

5.2 Evaluating neural fBm Hurst estimators

Due to self-similarity properties of fractional Brownian motion, and the stationarity of increments, a standardizing layer in the

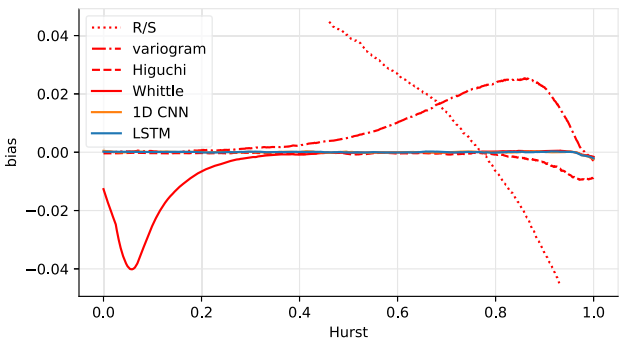


(a) Comparison of baseline estimators and neural models.

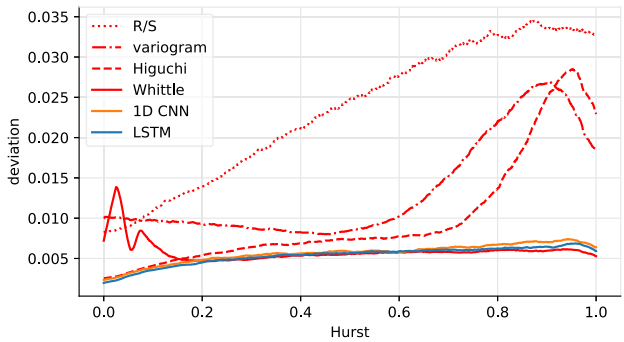


(b) Empirical consistency of different LSTM models.

Figure 1: MSE losses of different fBm Hurst-estimators by sequence length on a log-log scale.

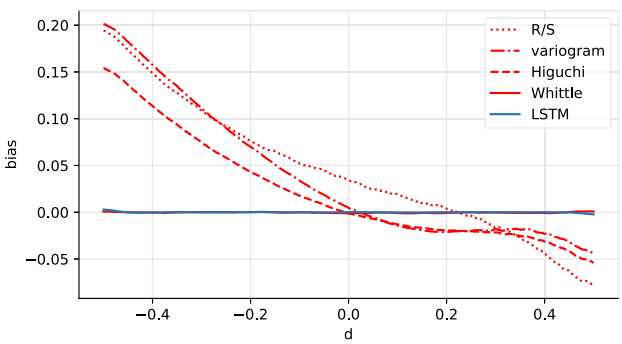


(a) Empirical bias $b_{0.025}$.

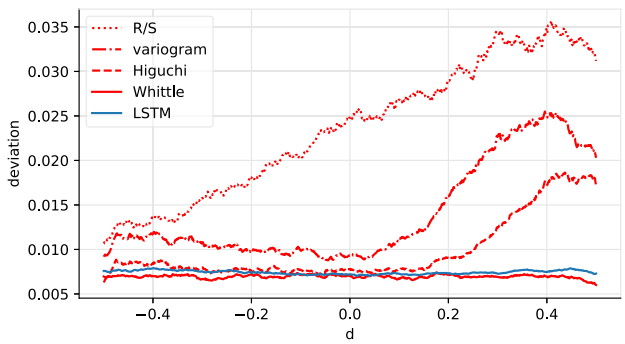


(b) Standard deviation $\sigma_{0.025}$.

Figure 2: Empirical bias and deviation of the different fBm estimators by Hurst value, measured on sequences of length 12800. The neural models were fine tuned until $n = 12800$. The bias of the R/S estimator ranges from 0.125 to -0.075, and was truncated on the plot.

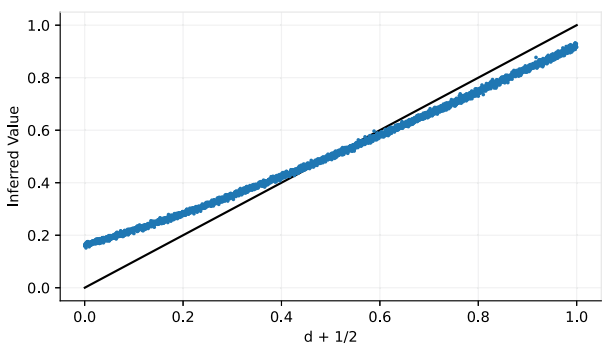


(a) Empirical bias $b_{0.025}$.

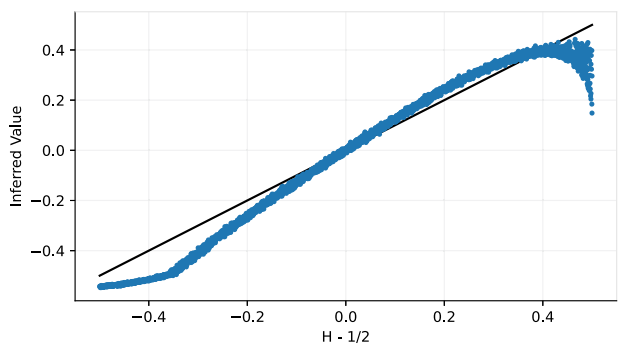


(b) Standard deviation $\sigma_{0.025}$.

Figure 3: Empirical bias and deviation of the ARFIMA$(0, d, 0)$ estimators by d. Measured on sequences of length 12800.



(a) Trained on fBm, inferring on ARFIMA sequences.



(b) Trained on ARFIMA, inferring on fBm sequences.

Figure 4: Scatterplots of M_{LSTM} model inferences on cross-processes of length 12800. The fBm model was fine-tuned on sequences up to length 12800, and the ARFIMA$(0, d, 0)$ model was trained on sequences of length 12800.

network architecture enables us to train only on realizations $\Delta FBM(H, n, 0, n)$. This simplified procedure yields an estimator that, in case of inference, is universally efficient regardless of the equidistant time-scale we choose, and regardless of the terminal time of the targeted process. Consequently, we generated sequences for training and evaluation from $\Delta\,\mathrm{FBM}(H, n, 0, n)$, where $H \sim U(0, 1)$ is random, and included the standardization layer in our models.

In the first experiment we fine-tuned the neural models M_{conv} and M_{LSTM} up to $n = 12800$. An initial training phase on trajectories of length $n = 100$ was performed on 200 and 100 virtual epochs each containing 100000 sequences for M_{conv} and M_{LSTM} respectively. The models were fine tuned on $n = 200, 400, 800, 1600, 3200, 6400$ and 12800 for an additional 20 and 10 virtual epochs for M_{conv} and M_{LSTM} respectively. At the end we got a model which was trained on all of the sequence lengths in the list, albeit not all at the same time. The results can be found in Table 1 and Figure 1a, where the loss for sequence length of n was measured after fine tuning on trajectories of length n. The neural models outperform all baseline estimators especially as the sequence length increases. It is of note that while M_{LSTM} achieved slightly smaller loss than M_{conv}, M_{LSTM} requires a bit more computational resources. Figure 2 compares the empirical bias and standard deviation of the final fine-tuned neural models and the baseline estimators as a function on the Hurst parameter.

Table 1: MSE losses of different fBm Hurst-estimators by sequence length. To enable direct comparisons with other solutions in the literature, we also included the performance of M_{LSTM} where only shift invariance is ensured by turning off the standardizing layer (M^*_{LSTM}), here the training and evaluation was performed on $\Delta\,\mathrm{FBM}(H, n, 0, 1)$.

	MSE loss ($\times 10^{-3}$)						
seq. len.	R/S	variogram	Higuchi	Whittle	M_{conv}	M_{LSTM}	M^*_{LSTM}
100	27.6	9.30	10.6	4.33	4.27	**4.07**	*0.214*
200	18.9	5.05	4.21	2.00	1.99	**1.91**	*0.0826*
400	13.9	2.92	1.99	1.00	0.959	**0.917**	*0.0366*
800	10.8	1.75	1.05	0.540	0.476	**0.453**	*0.0141*
1600	8.62	1.09	0.593	0.324	0.240	**0.224**	*0.0072*
3200	6.74	0.724	0.360	0.225	0.122	**0.114**	*0.0037*
6400	5.57	0.502	0.229	0.179	0.063	**0.058**	*0.0029*
12800	4.70	0.365	0.155	0.157	0.033	**0.030**	*0.0032*

We evaluated the empirical consistency of M_{LSTM} trained exclusively on certain length sequences. We can see the results in Table 2, and Figure 1b. Trained on shorter sequences the performance of M_{LSTM} improved when tested on longer sequences, but not as fast as M_{LSTM} variants trained on longer sequences. M_{LSTM} variants trained on longer sequences still performed well on shorter sequences, but not as well as dedicated variants.

Table 2: MSE losses of LSTM-based fBm Hurst-estimators trained on different sequence lengths.

train seq. len.	MSE loss by validation seq. len. ($\times 10^{-3}$)						
	100	200	400	800	1600	3200	6400
100	4.14	2.17	1.41	1.09	0.962	0.918	0.915
200	4.21	1.88	0.947	0.528	0.344	0.264	0.231
400	4.78	2.02	0.940	0.477	0.281	0.196	0.161
800	4.80	2.00	0.913	0.443	0.230	0.134	0.0888
1600	5.01	2.11	0.952	0.447	0.220	0.113	0.0617
3200	5.44	2.23	0.972	0.454	0.221	0.111	0.0608
6400	5.59	2.30	1.01	0.471	0.229	0.121	0.0692

5.2.1 *Testing on real-world data*

To evaluate the real-world performance of our neural fBm Hurst-estimators, we analyzed historical data from the S&P 500 stock market index. Volatility in financial markets can be modeled using fBm with Hurst exponent $H \approx 0.1$ [10]. Volatility is often calculated with overlapping sliding windows, but this increases the correlation between the neighboring values and skews the Hurst-estimates to be around 0.5. To avoid this, we calculated the daily volatility from 15 minute log-returns, which enabled us to estimate the Hurst parameter in yearly windows, without overlaps in the volatility calculation. Figure 5 shows the results, indicating that M_{LSTM} and M_{conv} effectively capture the temporal dynamics of market volatility, correlating well with traditional methods, most closely with Whittle's method, which was the best-performing classical method in the synthetic measurements. The neural models yield Hurst-estimations in the $(0,\ 0.3)$ range, in the anti-persistent Hurst regime.

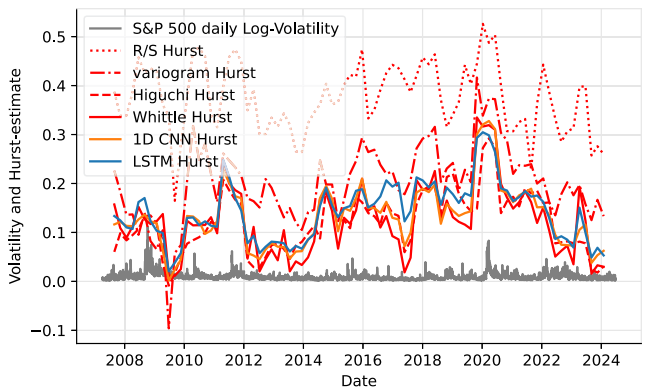


Figure 5: The figure shows Hurst-estimates for the daily S&P 500 log-volatility, calculated from 15-minute log-returns. Estimates use 252-day (one year) sliding windows with 189-day overlaps.

5.2.2 *Inference times*

In our experimental setup, the neural network estimators demonstrated a favorable balance between accuracy and speed at inference. The advantage was only partly due to GPU usage, as suggested by the overall inference times shown in Table 3. The indicated inference times depend on various factors, including the implementation and available hardware resources; they reflect the specific setup used at the time of writing.

Table 3: Overall running times for fBm Hurst-inference measured on 10000 sequences of length 3200.

CPU						GPU	
R/S	variogram	Higuchi	Whittle	M_{conv}	M_{LSTM}	M_{conv}	M_{LSTM}
7s	38m 42s	8s	2h 54m	1m 32s	5m 43s	10s	9s

5.3 *Evaluating neural ARFIMA parameter estimators*

We trained M_{LSTM} models for estimating the parameter d of the ARFIMA$(0, d, 0)$ process. These models contain no standardization, and work with the input sequence, not the increments. The models were trained on sequences of length 200, 400, 800, 1600, 3200, 6400 and 12800. Classical Hurst estimation techniques were evaluated for the inference of d as described in Section 3.5. Whittle's method was calibrated specifically for the ARFIMA d-estimation. The results are presented in Table 4 and Figure 3.

Table 4: MSE losses of different ARFIMA$(0, d, 0)$ d-estimators by sequence length.

	MSE loss ($\times 10^{-3}$)				
seq. len.	R/S	variogram	Higuchi	Whittle	M_{LSTM}
100	33.1	17.3	14.8	9.51	**5.96**
200	24.0	12.6	8.33	4.00	**3.03**
400	18.6	9.83	5.67	1.82	**1.54**
800	14.9	8.11	4.67	0.846	**0.787**
1600	12.6	7.70	4.24	0.401	**0.390**
3200	9.90	7.00	3.80	0.200	**0.199**
6400	8.64	6.95	3.80	**0.0960**	0.104
12800	7.51	6.94	3.75	**0.0487**	0.0552

5.3.1 *Stress-testing with ARFIMA*

We conducted cross-tests on M_{LSTM} models, by training them on ARFIMA and fBm trajectories, and then evaluating them against the alternate process. The ARFIMA d-estimator was trained on $n = 12800$, and the fBm H-estimator was finetuned up to $n = 12800$. In both cases the models were tested on sequences of length 12800. As Figure 4 shows, the models perform remarkably, with minor asymmetric bias with respect to the parameter range. This suggests that the model either captures the decay rate of autocovariance of fractional noise or some fractal property of sample paths.

5.4 *Evaluating neural fOU parameter estimators*

Estimating the parameters of the fractional Ornstein-Uhlenbeck process is significantly more difficult. We evaluated $\mathcal{M}_{\text{LSTM}}$ on the estimation of the Hurst parameter of fOU. Here M_{LSTM} does not work with the increments, but includes standardization to ensure scale and shift invariance. As we stated in Section 2.3 these invariances enable training on FOU$(\eta, H, \alpha, 0, 1)$ without the loss of generality. Additionally we evaluated the quadratic generalized variation (QGV) estimator for the Hurst parameter, as described in [4]. We trained M_{LSTM} on sequences of length 200, 400, 800, 1600, 3200 and 6400 with the fine-tuning technique similar to the one in Section 5.2. We generated the sequences for training and evaluation of the Hurst estimators from FOU$(\eta, H, \alpha, 0, 1)$, where $H \sim U(0, 1), \alpha \sim \text{Exp}(100)$ and $\eta \sim N(0, 1)$ are random.

Table 5: Performance metrics of different fOU Hurst-estimators by sequence length.

	MSE loss ($\times 10^{-3}$)		$\hat{b}_{0.025}$ ($\times 10^{-3}$)		$\hat{\sigma}_{0.025}$ ($\times 10^{-2}$)	
seq. len.	QGV	M_{LSTM}	QGV	M_{LSTM}	QGV	M_{LSTM}
100	41.0	**3.38**	106	**9.26**	9.86	**5.53**
200	34.2	**1.74**	97.1	**4.84**	8.07	**4.00**
400	29.4	**0.919**	86.4	**2.59**	6.92	**2.92**
800	25.0	**0.494**	76.2	**1.52**	5.88	**2.15**
1600	20.6	**0.269**	65.1	**0.827**	5.09	**1.59**
3200	16.3	**0.149**	53.8	**0.575**	4.37	**1.18**
6400	12.6	**0.081**	43.7	**1.95**	3.68	**0.842**

5.4.1 *Stress-testing with OU*

We stress-tested the M_{LSTM} Hurst-estimators trained on fBm, fOU and ARFIMA processes (in the case of ARFIMA, originally trained for estimating d) on the standard Ornstein-Uhlenbeck process FOU$(0, 0.5, \alpha, 0, 1)$; results are shown in Figure 6. The inferred value at $\alpha = 0$ is 0.5 as expected since the model receives an input that it already encountered in the learning phase. As the parameter α increases, all models return values tending towards zero. This could suggest that larger α values have a similar effect to smaller Hurst values, and sequences generated with a larger speed of mean reversion resemble trajectories that have greater anti-persistence. Moreover when $\alpha > 0$, the Ornstein-Uhlenbeck autocovariance shows exponential decay, contrary to the power decay associated with the data that the models were calibrated on. However, the fOU Hurst-estimator does stay around 0.5 longer, corresponding to the α range it encountered during training.

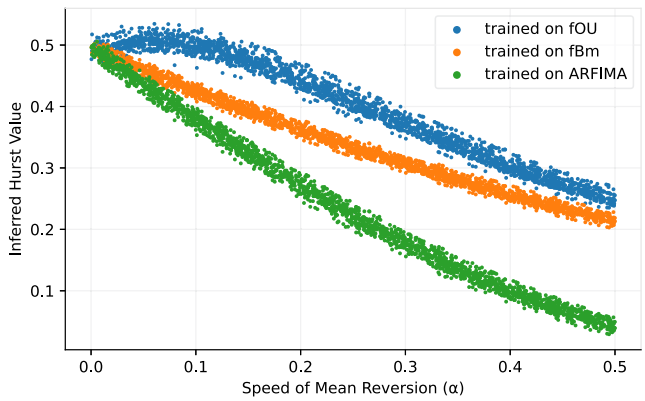


Figure 6: Scatterplot of Hurst estimations by M_{LSTM} models finetuned up to 12800 length sequences, inferring on Ornstein–Uhlenbeck processes of length 12800.

6 Conclusion

In this work, we have demonstrated the effectiveness of sequence-processing neural networks as powerful tools for estimating statistical parameters associated with long memory, such as the Hurst exponent. We have demonstrated the superior performance and consistency of purely neural network-based models over several commonly used statistical techniques in the context of fBm, fOU, and ARFIMA processes. This suggests that complex statistical descriptors can be effectively represented by relatively simple neural networks, a phenomenon that deserves further investigation.

The compact size of the neural networks used in our study enables fast inference, and the quality of the estimates they provide is on par with, or superior to, the most precise yet resource-intensive existing methods. A limitation of our arguments is the length of the sequences used in the measurements, nevertheless, in practice, the sequence lengths dealt with are sufficient in the vast majority of cases.

We believe that the proposed parameter estimators of stochastic processes, based on recurrent neural networks, with their usability and flexibility, form a good basis for the estimation methods that can be used for more intricate processes.

Acknowledgements

The research was supported by the Hungarian National Research, Development and Innovation Office within the framework of the Thematic Excellence Program 2021 – National Research Sub programme: “Artificial intelligence, large networks, data security: mathematical foundation and applications” and the Artificial Intelligence National Laboratory Program (MILAB), and the Hungarian National Excellence Grant 2018-1.2.1-NKP-00008. We would also like to thank GitHub and neptune.ai for providing us academic access.

References

[1] U. R. Acharya, S. V. Sree, P. C. A. Ang, R. Yanti, and J. S. Suri. Application of non-linear and wavelet based features for the automated identification of epileptic EEG signals. *International journal of neural systems*, 22(02):1250002, 2012.

[2] R. T. Baillie. Long memory processes and fractional integration in econometrics. *Journal of Econometrics*, 73(1):5–59, 1994.

[3] J. Beran. *Statistics for Long-Memory Processes*. Routledge, 11 2017. ISBN 9780203738481. doi: 10.1201/9780203738481.

[4] A. Brouste and S. Iacus. Parameter estimation for the discretely observed fractional Ornstein–Uhlenbeck process and the Yuima R package. *Computational Statistics*, 28(4):1529–1547, 2013.

[5] Christopher Flynn. fbm: Exact methods for simulating fractional brownian motion and fractional gaussian noise in python. https://github.com/crflynn/fbm, 2020.

[6] R. B. Davies and D. S. Harte. Tests for hurst effect. *Biometrika*, 74(1): 95–101, 1987.

[7] Z. Eisler and J. Kertesz. Size matters: some stylized facts of the stock market revisited. *The European Physical Journal B-Condensed Matter and Complex Systems*, 51:145–154, 2006.

[8] J. Feng, X. Wang, Q. Liu, Y. Li, and Y. Xu. Deep learning-based parameter estimation of stochastic differential equations driven by fractional brownian motions with measurement noise. *Communications in Nonlinear Science and Numerical Simulation*, 127:107589, 2023.

[9] C. L. Franzke, S. M. Osprey, P. Davini, and N. W. Watkins. A dynamical systems explanation of the hurst effect and atmospheric low-frequency variability. *Scientific reports*, 5(1):1–6, 2015.

[10] J. Gatheral, T. Jaisson, and M. Rosenbaum. Volatility is rough. In *Commodities*, pages 659–690. Chapman and Hall/CRC, 2022.

[11] I. I. Gikhman and A. V. Skorokhod. *The theory of stochastic processes I*. Springer Science & Business Media, 2004.

[12] L. Giraitis, H. Koul, and D. Surgailis. *Large Sample Inference For Long Memory Processes*, pages 33–54. World Scientific, 04 2012. ISBN 978-1-84816-278-5. doi: 10.1142/p591.

[13] T. Gneiting, H. Ševčíková, and D. B. Percival. Estimators of fractal dimension: Assessing the roughness of time series and spatial data. *Statistical Science*, pages 247–277, 2012.

[14] T. Graves, R. Gramacy, N. Watkins, and C. Franzke. A brief history of long memory: Hurst, mandelbrot and the road to arfima, 1951–1980. *Entropy*, 19(9):437, 2017.

[15] D. Han, N. Korabel, R. Chen, M. Johnston, A. Gavrilova, V. J. Allan, S. Fedotov, and T. A. Waigh. Deciphering anomalous heterogeneous intracellular transport with neural networks. *Elife*, 9:e52224, 2020.

[16] C. R. Harris, K. J. Millman, S. J. van der Walt, et al. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. doi: 10.1038/s41586-020-2649-2.

[17] S. Haykin. *Neural networks: A comprehensive foundation*. Prentice Hall PTR, 1994.

[18] T. Higuchi. Approach to an irregular time series on the basis of the fractal theory. *Physica D: Nonlinear Phenomena*, 31(2):277–283, 1988.

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] H. E. Hurst. The problem of long-term storage in reservoirs. *Hydrological Sciences Journal*, 1(3):13–27, 1956.

[21] P. Kidger, P. Bonnier, I. Perez Arribas, C. Salvi, and T. Lyons. Deep signature transforms. *Advances in Neural Information Processing Systems*, 32, 2019.

[22] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman. 1D convolutional neural networks and applications: A survey. *Mechanical systems and signal processing*, 151:107398, 2021.

[23] L. Kirichenko, K. Pavlenko, and D. Khatsko. Wavelet-based estimation of hurst exponent using neural network. In *2022 IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT)*, pages 40–43. IEEE, 2022.

[24] A. Kononovicius. arfima: Python implementation of arfima process with an aim to simulate series. https://github.com/akononovicius/arfima, 2021.

[25] D. P. Kroese and Z. I. Botev. Spatial process simulation. In *Stochastic geometry, spatial statistics and random fields: Models and algorithms*, pages 369–404. Springer, 2014.

[26] S. Ledesma-Orozco, J. Ruiz-Pinales, G. García-Hernández, G. Cerda-Villafaña, and D. Hernández-Fusilier. Hurst parameter estimation using artificial neural networks. *Journal of applied research and technology*, 9(2):227–241, 2011.

[27] M. Li. Change trend of averaged hurst parameter of traffic under DDOS flood attacks. *Computers & security*, 25(3):213–220, 2006.

[28] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[29] D. Mottl. hurst: Hurst exponent evaluation and R/S-analysis in Python. https://github.com/Mottl/hurst, 2019.

[30] S. Mukherjee, B. Sadhukhan, A. K. Das, and A. Chaudhuri. Hurst exponent estimation using neural network. *International Journal of Computational Science and Engineering*, 26(2):157–170, 2023.

[31] M. Musiela and M. Rutkowski. *Martingale methods in financial modelling*, volume 36. Springer Science & Business Media, 2006.

[32] neptune.ai. neptune.ai: experiment tracking and model registry, 2022. URL https://neptune.ai.

[33] D. Nualart and E. Nualart. *Introduction to Malliavin Calculus*. Cambridge University Press, 1th edition, 2018. doi: 10.1017/9781139856485.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[35] B. Qian and K. Rasheed. Hurst exponent and financial market predictability. In *IASTED conference on Financial Engineering and Applications*, pages 203–209. Proceedings of the IASTED International Conference Cambridge, MA, 2004.

[36] M. N. R.C. Lopes. Long memory analysis in dna sequences. *Physica A*, 361(2):569–588, 2006.

[37] M. S. Taqqu, V. Teverovsky, and W. Willinger. Estimators for long-range dependence: an empirical study. *Fractals*, 3(04):785–798, 1995.

[38] R. Vallat. Antropy: entropy and complexity of (eeg) time-series in python. https://github.com/raphaelvallat/antropy, 2022.

[39] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

[40] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al. SciPy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

[41] X. Wang, J. Feng, Q. Liu, Y. Li, and Y. Xu. Neural network-based parameter estimation of stochastic differential equations driven by lévy noise. *Physica A: Statistical Mechanics and its Applications*, 606: 128146, 2022.

[42] P. Whittle. *Hypothesis Testing in Time Series Analysis*. PhD thesis, Uppsala, 1951.

[43] W. Willinger, V. Paxson, R. Riedi, and M. S. Taqqu. Long-range dependence and data network traffic. *Theory and applications of long-range dependence*, pages 373–407, 2003.

[44] N. Yuan, C. L. Franzke, F. Xiong, Z. Fu, and W. Dong. The impact of long-term memory on the climate response to greenhouse gas emissions. *npj Climate and Atmospheric Science*, 5(1):70, 2022.