# Project work 3

Parameter estimation of stochastic processes with neural networks

Dávid Kovács

Supervisor: András Lukács

## 1 Introduction

There are several stochastic processes that are important in theoretical finance, but barely usable in practice, due to the lack of methods to estimate their parameters. However, often it is possible to generate (discretized) trajectories from the processes in question. If we can generate trajectories of a stochastic process, then we can also train neural networks on the generated trajectories for the parameter estimation of the process. This chain of thought also outlines the goal of this work. We want to construct estimators via deep learning, for processes that are relatively easy to generate but hard to estimate by methods of classical statistics. The deep learning modules referenced throughout this work are described in [4].

## 2 Bayesian estimation with deep learning

The introductory statement regarding the relation between generation and parameter estimation remains true in a much more general setting. This can be formalized as follows, pointing out that estimators constructed this way are Bayesian in nature. Let $\vartheta$ be the set of the possible parameters and let $P$ be the prior distribution on $\vartheta$. Then $\forall \vartheta \in \Theta$ let $Q_\vartheta : \mathcal{X} \to [0,1]$ be the distribution indexed by $\vartheta$ . Moreover let $L : \vartheta^2 \to \mathbb{R}_+$ be a loss function. $\forall \vartheta \in \vartheta$ let $G_\vartheta \sim Q_\vartheta$ be a generator. We can then sample $\vartheta_1, \vartheta_2, \ldots, \vartheta_n$ from distribution $P$. Then we can create the learning set $(G_{\vartheta_1}, \vartheta_1), (G_{\vartheta_2}, \vartheta_2), \ldots, (G_{\vartheta_n}, \vartheta_n)$ where $n$ can be arbitrarily large. The goal of training a neural network on this set with loss function $L$ is trying to find the estimator $S$ that minimizes the Bayesian risk

$$R_P(S) = \int_\vartheta \int_\mathcal{X} L(t, S(x)) Q_t(dx)\, P(dt).$$

In many ways datasets generated this way are ideal for training neural networks. First of all, the data is labelled perfectly, assuming that the generators are functioning correctly. Second of all, we have practically infinite unique training data. This is due to the fact, that if the generation is fast enough then we can generate enough data to eliminate the need for reusing any $(G_\vartheta, \vartheta)$ pairs. In practice this means that during the training process every batch is generated on the fly and one epoch simply means a certain number of training pairs. Using unique data means that overfitting is not possible, hence training losses can be treated as validation losses.

## 3 Estimation of shape, location and scale with deep learning

Let $\Theta$ be the set of shape parameters, now let $\Theta^* = \Theta \times \mathbb{R} \times \mathbb{R}_+$ be the location and scale extended parameter set. For any $(\theta, \nu, \lambda) = \vartheta^* \in \Theta^*$, $Q_{\vartheta^*}$ is defined as the distribution of $\lambda X + \nu \cdot v_0$, where $X \sim Q_\vartheta$ and $v_0 \in \mathcal{X}$ is the predefined location basis. We want to estimate $\vartheta$, $\nu$ and $\lambda$. Put into the context of Section 2, we may think that having these two extra parameters means having $\Theta^*$ as the Bayesian parameter set. However, in the following we propose a way to solve the problem defined by $\Theta^*$, while actually working on just $\Theta$. We achieve this by defining neural modules, that combined appropriately result in neural networks, that can extrapolate to $\Theta^*$ after being taught on $\Theta$.

Before we begin, some explanation is in order about the notation. Modules will have capital letters as names, suggestive of their role, eg., $E$ stands for embedding, $A$ stands for average and $P$ stands for projection. They will have $\nu$, $\lambda$ as lower or upper indicies. Lower indices will suggest some kind of invariance and upper indices will suggest some kinds of additivity (or homogeneity) to the respective parameters. Moreover, the abbreviation seq2vec_seq will be used to mean modules embedding sequences to sequences of vectors, and vec2scal will mean modules projecting vectors into scalars. Furthermore, vec_seq2vec will mean modules reducing sequences of vectors into vectors.

### 3.1 Homogeneous modules

Here we propose a homogeneous seq2vec_seq neural module $E^\lambda$, ie., a module having $E^\lambda(\lambda x) = \lambda E^\lambda(x)$. A straightforward way to implement such a module is having a multilayer 1D convolution module with no bias and PReLU activations between the layers. We also consider a homogeneous vec2scal module $P^\lambda$. A possible easy way to implement $P^\lambda$ is having any vec2scal MLP with no bias and PReLU activation between the layers. And finally, let $A^\lambda$ be an adaptive average pooling layer, which is a homogeneous vec_seq2vec module.

The homogeneity of $A^\lambda$ is trivial. To check the homogeneity of $E^\lambda$ and $P^\lambda$, note that when there is no bias convolutional and fully connected layers are homogeneous, and the PReLU activation is homogeneous as well. So $E^\lambda$ and $P^\lambda$ are compositions of homogeneous functions, so they are homogeneous.

## 3.2 Scale-invariant modules

In this subsection we propose a scale-invariant vec2vec module $M_\lambda$. For clarity, scale invariance means $M_\lambda(\lambda x) = M_\lambda(x), \forall\lambda \in \mathbb{R}, x \in \mathcal{X}$. Such a module can be implemented as $M_\lambda(x) = \dfrac{x}{P^\lambda(x)}$. To see the scale-invariance of $M_\lambda$, we can write

$$M_\lambda(\lambda x) = \frac{\lambda x}{P^\lambda(\lambda x)} = \frac{\lambda x}{\lambda P^\lambda(x)} = \frac{x}{P^\lambda(x)} = M_\lambda(x).$$

## 3.3 Location-additive modules

Here we propose a location-additive and homogeneous seq2scal module $M^{\nu,\lambda}$. By location additivity we mean $M^{\nu,\lambda}(x+\nu v_0) = M^{\nu,\lambda}(x)+\nu$. Let $B$ be a sec2scal linear module, then let $M^{\nu,\lambda}(x) = \frac{B(x)}{B(v_0)}$. To check the location additivity property, we can write

$$M^{\nu,\lambda}(x+\nu v_0) = \frac{B(x+\nu v_0)}{B(v_0)} = \frac{B(x)}{B(v_0)} + \nu\frac{B(v_0)}{B(v_0)} = M^{\nu,\lambda}(x) + \nu.$$

And the homogeneity is trivial as $B$ is a linear module. We implemented $B$ as the composition of a single 1D convolution layer, an adaptive average layer and a single fully connected linear layer. And by linearity, we mean having no bias and no activation function.

## 3.4 Location-invariant modules

Here we propose a location-invariant, homogeneous sec2sec module $M_\nu^\lambda$. By location invariance we mean $M_\nu^\lambda(x+\nu v_0) = M_\nu^\lambda(x), \forall\nu \in \mathbb{R}, x \in \mathcal{X}$. We can implement this module with $M_\nu^\lambda(x) = x - M^{\nu,\sigma}(x)\, v_0$. To see the location-invariance of $M_\nu$, we can write

$$\begin{aligned} M_\nu^\lambda(x+\nu v) &= (x+\nu v_0) - M^{\nu,\lambda}(x+\nu v_0)\, v_0 = \\ &= x + \nu v_0 - (M^{\nu,\lambda}(x)+\nu)\, v_0 = x - M^{\nu,\lambda}(x)\, v_0 = M_\nu^\lambda(x). \end{aligned}$$

And the homogeneity is trivial.

## 3.5 Parameter estimation

For the estimation of $\vartheta$, let $\mathcal{M}^\vartheta = P \circ M_\lambda \circ A^\lambda \circ E^\lambda \circ M_\nu^\lambda$, where $P$ is an arbitrary vec2sec MLP. This way, $\mathcal{M}^\vartheta$ is a scale-invariant and location-invariant neural network. Therefore, it suffices to train and validate $\mathcal{M}^\vartheta$ on $\Theta$. And of course any loss will be the same on $\Theta^*$ as on $\Theta$.

For the estimation of $\lambda$, let $\mathcal{M}^\lambda = P^\lambda \circ A^\lambda \circ E^\lambda \circ M_\nu^\lambda$, which is a homogeneous and location-invariant neural network. Interestingly, it suffices to train $\mathcal{M}^\lambda$ on $\Theta$ as well, with constant $\lambda = 1$ labels. After having been trained on $\Theta$, $\mathcal{M}^\lambda$ works on $\Theta^*$ as well, because it is invariant to $\nu$ and can extrapolate to $\lambda \neq 1$. To see this let $\vartheta^* \in \Theta^*$, and let $X \sim Q_{\vartheta^*}$. Then $X = \lambda X_0 + \nu v_0$, where

$X_0 \sim Q_\vartheta$. We can use the scale-invariance and homogeneity of $\mathcal{M}^\lambda$ to write $\mathcal{M}^\lambda(\lambda X_0 + \nu v_0) = \lambda \mathcal{M}^\lambda(X_0)$. Then $(\mathcal{M}^\lambda(X) - \lambda)^2 = \lambda^2(\mathcal{M}^\lambda(X_0) - 1)^2$. So having $\lambda \neq 1$ means a $\lambda^2$ multiplier to the squared error. Which is the best one can hope for, as $I(\vartheta, \nu, \lambda) = I(\vartheta, \nu, 1)/\lambda^2$, where $I$ is the Fisher information. This conclusion may come as a surprise, because this means that the model can learn to estimate a parameter, that may not even be implemented.

For the estimation of $\nu$, let $\mathcal{M}^\nu = P^\lambda \circ A^\lambda \circ E^\lambda \circ M_\nu^\lambda + M^{\nu,\lambda}$, which is a homogeneous and scale-additive neural network. Similarly, it can be trained on $\Theta$ with constant $\nu = 0$ labels. To see this we can write $\mathcal{M}^\nu(\lambda X_0 + \nu v_0) = \lambda \mathcal{M}^\nu(X_0) + \nu$. Then $(\mathcal{M}^\nu(X) - \nu)^2 = \lambda^2(\mathcal{M}^\nu(X_0) - 0)^2$. So having $\nu \neq 0$ makes no difference in the squared error. However $\lambda \neq 1$ constitutes a $\lambda^2$ multiplier again, which is an unavoidable fact because of the Fisher information.

## 3.6 Consistency

In this subsection we examine the consistency of $\mathcal{M}^\vartheta$, assuming sequential input. If we train $\mathcal{M}^\vartheta$ on some fixed series length $N$, there is no guarantee that the Bayesian risk will descrease when we validate on some $N' > N$ series length. And there is even less guarantee that the model will be consistent. However if the input sequence is stationary and the location base $v_0$ is constant, then $\mathcal{M}^\vartheta$ has some desirable properties.

To recap, $\mathcal{M}^\vartheta = P \circ M_\lambda \circ A^\lambda \circ E^\lambda \circ M_\nu^\lambda$ with an arbitrary vec2scal MLP $P$. The output of $M_\nu^\lambda$ is stationary and just as importanly, it is the same stationary sequence for every input length. The former comes from the fact, that for stationary input, the output of a 1D convolution module is stationary as well. And the latter comes from $v_0$ being constant, so in $M_\nu(x) = x - \frac{B(x)}{B(v_0)} v_0$, $B(v_0)$ does not depend on the length of $v_0$. Then obviosuly the output of $E^\lambda$ is stationary as well. Then $A^\lambda$ has a stationary vector sequence as input, so the output converges to the expected value $m_\vartheta$ of the stationary vector distribution. Then $\mathcal{M}^\vartheta$ is consistent iff $P(M_\lambda(m_\vartheta)) = \vartheta$. We have no guarantee that this holds, but $m_\vartheta$ is not only the expected value of the stationary distribution, but that of $A^\lambda$'s output as well. So everything depends on $A^\lambda$ "working well on average", which is usually true to some extent. This last step was supported by our empirical findings.

It is also worth mentioning, that in the above chain of thought we assumed that the new series length $N' > N$ is achieved by observing new values $X_{(N+1)/N \cdot T}, X_{(N+2)/N \cdot T}, \ldots, X_{N'/N \cdot T}$. This way the new values are the continuation of the same stationary series. Having $X_0, X_{T/N'}, X_{2T/N'}, \ldots, X_T$ as the new sample would usually forfeit the consistency. A notable exception is if $X$ is self-similar, because that way the new stationary distribution is the same as the old one, only it is scaled differently, but $M$ is scale-invariant so that does not count.

## 4 Fractional Brownian motion

The aforementioned lack of estimators is somewhat less apparent in the case of the fractional Brownian motion, as the Higuchi method [1] is a fairly efficient estimator for the Hurst exponent. For this reason, trying to estimate the Hurst exponent of the fractional Brownian motion is an adequate first step to gauge the viability of deep learning based parameter estimation. Therefore our first goal is to construct an estimator for the Hurst exponent of the fractional Brownian motion, that outperforms the Higuchi method.

### 4.1 Parametrizing the fractional Brownian motion

We want to estimate the parameters of the process $\left(\sigma B_t^H + \mu t\right)_{t\in[0,T]}$, where $H \in (0,1)$, $\mu \in \mathbb{R}$, $\sigma \in \mathbb{R}_+$ and $B^H$ is a fractional Brownian motion with Hurst exponent $H$. At first glance, there are 4 parameters: $H$, $\mu$, $\sigma$ and practically even $T$ is a parameter, as we can only observe discretized trajectories of this process, thus we have no knowledge of the underlying time interval. However, using better parametrization, it is revealed that we actually only have shape parameter $\vartheta = H$ parametrizing $(B_t^H)_{t\in[0,1]}$. And adding scale parameter $\lambda = \sigma T^H$, and location parameter $\nu = \mu T$ with basis $v_0 = (t)_{t\in[0,1]}$ yields every processes parametrized above. To see this we will use the self-similarity of the fractional Brownian motion to write

$$\lambda \cdot \left(B_t^H\right)_{t\in[0,1]} + \nu \cdot (t)_{t\in[0,1]} = \left(\sigma T^H B_t^H + \mu T t\right)_{t\in[0,1]} \overset{d}{=} \left(\sigma B_t^H + \mu t\right)_{t\in[0,T]}.$$

So we have a problem defined by the set of shape parameters $\Theta = (0,1)$ and as described in Section 3, we extend $\Theta$ into $\Theta^*$. As shown in Section 3, we only need a prior distribution on $\Theta$, for which a reasonable choice is $U(0,1)$. And obviously we cannot work with $\left(\sigma B_t^H\right)_{t\in[0,1]}$ and $(t)_{t\in[0,1]}$, we need to discretize them in $\{0, 1/N, 2/N, \ldots, 1-1/N, 1\}$, assuming we have $N$ equidistant observations.

With these and the results of Section 3 in mind, we have neural networks $\mathcal{M}^\vartheta$, $\mathcal{M}^\lambda$ and $\mathcal{M}^\nu$ readily available for the estimation of $\vartheta = H$, $\lambda = \sigma T^H$ and $\nu = \mu T$.

### 4.2 Data generation and training

To be able to train $\mathcal{M}^\vartheta$, we need to generate observations from $Q_\vartheta$ for arbitrary $\vartheta \in \Theta$. In other words, $\forall H \in (0,1)$ we need to generate unscaled and undrifted fBm sequences on $[0,1]$ with Hurst exponent $H$. For this purpose, we used a Python implementation of the method described in [2]. The Python version was implemented by I. Ivkovic and D. J. Boros. Having a fast generator is highly beneficical, because to achieve the results documented in the next subsection, we needed to train the model for 300 epochs. Here one epoch means 100000

| Squared Bayesian risk of $\hat{H}$ | Higuchi | $\mathcal{M}^{\vartheta}_{200}$ |
|---|---|---|
| $N = 200$ | 0.00416 | 0.00200 |
| $N = 400$ | 0.00197 | 0.000985 |
| $N = 800$ | 0.00105 | 0.000522 |
| $N = 1600$ | 0.00058 | 0.000304 |
| $N = 3200$ | 0.000353 | 0.000212 |
| $N = 6400$ | 0.000231 | 0.000177 |
| $N = 12800$ | 0.000151 | 0.000163 |

Table 1: Squared Bayesian risk of the Higuchi estimator and $\mathcal{M}^{\vartheta}$ for different input length and fixed $N = 200$ training sequence length

unique input sequences (and of course every epoch is unique, there is no reusage of data).

We want $\mathcal{M}^{\vartheta}$ to be consistent to some extent, so we need stationary input. The fractional Brownian motion is not stationary, however, its increment are. So we train $\mathcal{M}^{\vartheta}$ on fBm increments. This means that we need to use the increments of $v_0$ as well, which would be the vector $(1/N, \ldots, 1/N)$. However as mentioned in Subsection 3.6, having a $v_0$ independent of $N$ is vital to the consistency. So we simply use $v_0 = (1, \ldots, 1)$. In practice this makes no difference to the validity of $\mathcal{M}^{\vartheta}$.

## 4.3 Results

At this work, we only document the results of estimating $\theta = H$. We pit $\mathcal{M}^{\vartheta}$ against the Higuchi method, which is a statistical estimator for $H$, that works on fractional Brownian motions with scaling but no drift. We test the two estimators on undrifted, unscaled fBm sequences. Both estimators would yield the same result on scaled input. However, while $\mathcal{M}^{\vartheta}$ would give the same results for drifted input, the Higuchi estimator would be completely off.

We will denote a version of $\mathcal{M}^{\vartheta}$ trained on length $N$ by $\mathcal{M}^{\vartheta}_N$. After training $\mathcal{M}^{\vartheta}$ on fixed $N = 200$ length, we have the results of Table 1 validating on different $N$ values compared to the Higuchi method. $\mathcal{M}^{\vartheta}$ indeed has some constistency, but the rate of the consistency vanishes as $N$ increases. In constrast, the Higuchi method has better consistency and it becomes the better option for $N = 12800$. But keep in mind that $\mathcal{M}^{\vartheta}$ solves a harder, more general problem as it can deal with drifted fBm series. Also, forcing $\mathcal{M}^{\vartheta}$ to be consistent is not necessary right now, as we can train seperate models for different ranges of $N$. However, in a use case where we do not have infinite training data, having a consistent model could be beneficial.

Observe what happens when we fine-tune seperate models for the $N$ values of Table 1. In practice, we train a model for $N = 200$, then fine-tune that model for $N = 400$, then fine-tune the new one for $N = 800$ and so on. The fine-tuning required is usually only 1 or 2 epochs. We also observe the performance of the

| Squared Bayesian risk of $\hat{H}$ | Higuchi | $\mathcal{M}_N^{\vartheta}$ | $\mathcal{M}_{12800}^{\vartheta}$ |
|---|---|---|---|
| $N = 200$ | 0.00416 | 0.00200 | 0.00214 |
| $N = 400$ | 0.00197 | 0.00097 | 0.00106 |
| $N = 800$ | 0.00105 | 0.000475 | 0.000526 |
| $N = 1600$ | 0.00058 | 0.000237 | 0.000249 |
| $N = 3200$ | 0.000353 | 0.000125 | 0.000125 |
| $N = 6400$ | 0.000231 | 0.000065 | 0.000064 |
| $N = 12800$ | 0.000151 | 0.0000326 | 0.0000326 |

Table 2: Squared Bayesian risk of the Higuchi estimator and $\mathcal{M}^{\vartheta}$ for different input length and different training (fine-tuning) sequence length

last model, $\mathcal{M}_{12800}^{\vartheta}$ for shorter sequences. As we can see from Table 2, by fine-tuning for every $N$ value, we have perfect $1/N$ rate consistency. So if one has enough resources, the fine-tuning method is the way to go. In this case this would mean having 8 different versions of the same model, and using each one when appropriate. Finally, we should note that these 8 seperate models are capable of parameter estimation on input lengths close to the length they were trained on, eg., a model fine-tuned for $N = 1600$ can yield basically the same accuracy for $N = 3200$ as the one fine-tuned on $N = 3200$. Which means, that we have every possible input length covered with a better rate of consistency than the Higuchi estimator. Also, it is wort noting that $\mathcal{M}_{12800}^{\vartheta}$ is very close to the multiple-headed version in performance, so it is also a viable option.

We conclude this section with Figure 1, which plots the mean squared error of $\mathcal{M}_{200}^{\vartheta}$ and the Higuchi method as a function of $H$. It shows that even on sequences of length 200, $\mathcal{M}^{\vartheta}$ works better for every $H$. As $N$ increases this superiority of $\mathcal{M}^{\vartheta}$ only increases. We could also reduce the loss of $\mathcal{M}^{\vartheta}$ by about 10% if we turned off the initial $M_{\mu}^{\lambda}$ module (so it would not work for drifted fBm).

## 5 Fractional Ornstein-Uhlenbeck process

The Fractional Ornstein-Uhlenbeck (FOU) process fits more the description in the Introduction. It is the fractional noise driven version of the well-known and widely used Ornstein-Uhlenbeck (OU) process. However, its parameter estimation is far more complicated than that of the OU process. As we will see, the few existing estimators are fairly easily surpassed by our DL-based approach.

### 5.1 Parametrizing the fractional Ornstein-Uhlenbeck process

We want to estimate the parameters of the process $(X_t)_{t\in[0,T]}$, where

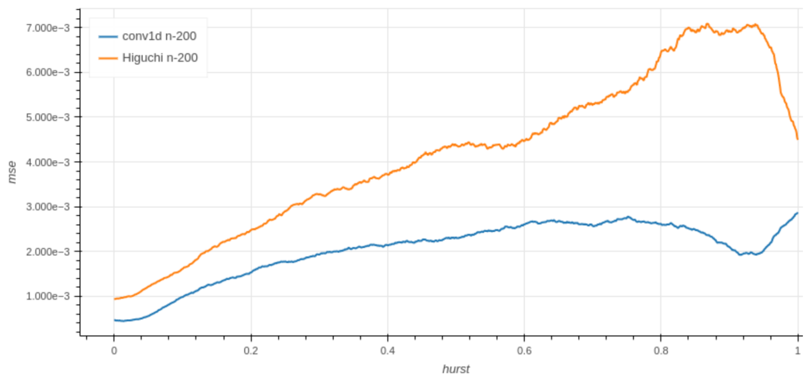

Figure 1: The mean squared error of the Higuchi method and $\mathcal{M}^{\vartheta}$, plotted as a function of $H$.

$$X_t = X_0 - \alpha \int_0^t (X_s - \mu)\, ds + \sigma B_t^H \quad (\forall t \in [0,T]), \tag{1}$$

with $T, \alpha, \sigma \in \mathbb{R}_+$, $H \in (0,1)$ and $\mu \in \mathbb{R}$. At first glance, there are 5 parameters: $\alpha$, $H$, $\mu$, $\sigma$ and even $T$ seems to be an unknown parameter. However, we will show that there are actually only two shape parameters: $\vartheta_1 = \alpha T$ and $\vartheta_2 = H$ parametrizing processes $(X_t)_{t\in[0,1]}$, where

$$X_t = X_0 - \alpha T \int_0^t X_s\, ds + B_t^H \quad (\forall t \in [0,1]). \tag{2}$$

And with the additional location parameter $\nu = \mu$ (with basis $v_0 \equiv 1$) and scale parameter $\lambda = T^H\sigma$ we have every process parameterized in (1). To see this, multiply each side of (2) by $\lambda$ and add $\mu$ to both sides,

$$\lambda X_t + \mu = \lambda X_0 + \mu - \alpha T \int_0^t (\lambda X_s + \mu) - \mu\, ds + \lambda B_t^H \quad (\forall t \in [0,1]).$$

Using that $\lambda B_t^H \overset{d}{=} \sigma B_{tT}^H$ we have

$$Y_t = Y_0 - \alpha T \int_0^t Y_s - \mu\, ds + \sigma\lambda B_{tT}^H \quad (\forall t \in [0,1]),$$

with $Y = \lambda X + \nu$. As a final step, substitute $u = tT$ into the equation and $r = sT$ into the integral

$$Y_{u/T} = Y_{0/T} - \alpha \int_0^u (Y_{r/T} - \mu)\, dr + \sigma B_u^H \quad (\forall u \in [0,T]).$$

So we started with $(X_t)_{t\in[0,1]}$ parametrized by (2), scaled it with $\lambda$ and shifted it with $\nu$, and we arrived at $(Y_{u/T})_{t\in[0,T]}$, which is a process parametrized by (1). The only thing left to consider is the role of $X_0$ in (2). We need a distribution to yield the initial $X_0$ value. Let $[-\delta(\alpha, H), \delta(\alpha, H)]$ be the support of this distribution, where $\delta : \mathbb{R}_+ \times (0, 1) \to \mathbb{R}_+$. Then we have $[\mu - \lambda\delta(\alpha), \mu + \lambda\delta(\alpha)]$ as the support of the distribution of $Y_0$. With an appropriate choice of $\delta$, this interval will contain the stationary FOU distribution with high probability. Choosing $\delta$ requires further investigation, for now let $\delta \equiv 10$.

In conclusion, the FOU process is indeed parametrized by $\Theta = R_+ \times (0, 1)$, and the rest of the parameters are the result of $\Theta$ being extended into $\Theta^*$. So for estimating $\vartheta_1 = \alpha T$ and $\vartheta_2 = H$ we have $\mathcal{M}^\vartheta$, for $\nu = \mu$ we have $\mathcal{M}^\nu$ and for $\lambda = T^H\sigma$ we have $\mathcal{M}^\lambda$.

## 5.2 Data generation and training

As before, we need to generate instances from any distribution in $\{Q_\vartheta : \vartheta \in \Theta\}$. In this case, this means generating $(X_t)_{t\in[0,1]}$ processes satisfying

$$X_t = -\alpha T \int_0^t X_s \, ds + B_t^H \quad (\forall t \in [0, 1]).$$

A method developed and implemented by I. Ivkovic was used to generate the processes in question.

To train our neural networks, we need a distribution to generate $H$ and $\alpha$. The former is straightforward, we can have $H \sim U(0, 1)$. The distribution for $\alpha$ is more problematic. First of all, for large $\alpha$ values the simulation becomes unstable. Second of all, The FOU process converges to the constant $X \equiv \mu$ process if $\alpha \to \infty$. So after a certain $\alpha$ treshold there is little difference between the corresponding FOU distributions. Which means we should put less weight on large $\alpha$ values, as getting those right is less important than getting small $\alpha$ values right. This convergence of course only causes a problem for $\mathcal{M}^\vartheta$. For the other two esimators it might even help. For now, we use $\alpha \sim U(0, 10)$, but this will probably change in future works.

## 5.3 Partial results

As a baseline, we will use the Nualart estimator, described in [3]. This estimator works on FOU processes with $\mu = 0$ and gives an estimate for $H$, $\sigma$ and $\alpha$. We will fix $T = 1$, so $\theta_1 = \alpha$, $\lambda = \sigma$.

First, compare $M^\vartheta$ to the Nualart estimator for the estimation of $H$. The squared Bayesian risks can be seen in Table 3. Clearly, $M^\vartheta$ has a clear edge over the Nualart method. Moreover, as opposed to the Nualart estimator, $M^\vartheta$ works even if $\mu \neq 0$. One thing that is different from the fBm case, is that now $\mathcal{M}^\vartheta_{12800}$ does not work as well for shorter sequences. This is probably due to the FOU not being stationary just ergodic.

Now compare the Nualart estimator for $\sigma$ to $\mathcal{M}^\lambda$. The results of this comparison can be seen in Table 4. Here the Bayesian paradigm is of course still

| Squared Bayesian risk of $\hat{H}$ | Nualart | $\mathcal{M}_N^\vartheta$ | $\mathcal{M}_{12800}^\vartheta$ |
|---|---|---|---|
| $N = 800$ | 0.00152 | 0.000487 | 0.00586 |
| $N = 1600$ | 0.000736 | 0.000250 | 0.00200 |
| $N = 3200$ | 0.000367 | 0.000136 | 0.000227 |
| $N = 6400$ | 0.000185 | 0.000071 | 0.0000790 |
| $N = 12800$ | 0.00009 | 0.000038 | 0.000038 |

Table 3: Squared Bayesian risk of the Nualart estimator and $\mathcal{M}^\vartheta$ for different input length and different training (fine-tuning) sequence length

| Squared Bayesian risk of $\hat{\sigma}$ | Nualart | $\mathcal{M}_N^\lambda$ |
|---|---|---|
| $N = 800$ | $0.0780 \cdot \sigma^2$ | $0.0326 \cdot \sigma^2$ |
| $N = 1600$ | $0.0446 \cdot \sigma^2$ | $0.0225 \cdot \sigma^2$ |
| $N = 3200$ | $0.0263 \cdot \sigma^2$ | $0.0162 \cdot \sigma^2$ |
| $N = 6400$ | $0.016 \cdot \sigma^2$ | $0.011 \cdot \sigma^2$ |
| $N = 12800$ | $0.009 \cdot \sigma^2$ | $0.0127 \cdot \sigma^2$ |

Table 4: Squared Bayesian risk of the Nualart estimator and $\mathcal{M}^\lambda$ for different input length and different training (fine-tuning) sequence length

understood w.r.t. $H$ and $\alpha$, and there is a $\sigma^2$ multiplier for the risk of $\hat{\sigma}$ (see Subsection 3.5). As we can see, $\mathcal{M}_N^\lambda$ performs better than the Nualart method. However, the advantage of $\mathcal{M}_N^\lambda$ decreases as $N$ increases. And for $N = 12800$, $\mathcal{M}^\lambda$ could not yet be satisfactorily trained.

Comparing the estimates for $\alpha$ and finding an estimator to which we can compare $\mathcal{M}^\nu$ are left to future works.

## 6 Conclusion

We proposed three neural network prototypes that can be used in a wide variety of sequential modeling problems. Altough they have been only tested out in a handful of scenarios, we believe they can yield better results than the existing statistical estimators in the majority of situations. Also, once these models get more solidified, training them for a new problem will be significantly easier then searching for existing statistical methods.

## References

[1] Higuchi, T. (1990). Relationship between the fractal dimension and the power law index for a time series: A numerical investigation. Physica D: Nonlinear Phenomena.

[2] Kroese, D. P. & Botev, Z. I. (2014). Spatial Process Simulation. Stochastic Geometry, Spatial Statistics and Random Fields(pp. 369-404). Springer International Publishing.

[3] Yaozhong Hu, David Nualart & Hongjuan Zhou (2017). Parameter estimation for fractional Ornstein–Uhlenbeck processes of general Hurst parameter. https://arxiv.org/abs/1703.09372.

[4] Goodfellow, I., Bengio, Y. & Courville, A. (2016). Deep learning. MIT Press.