

A deep learning-based Monte Carlo simulation scheme for stochastic differential equations driven by fractional Brownian motion

Fei Gao^a, Cornelis W. Oosterlee^b, Jianshe Zhang^{a,*}

^a School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an, Shanxi, China

^b Mathematical Institute, Utrecht University, Utrecht, Netherlands

ARTICLE INFO

Communicated by M. Bianchini

Keywords:

Deep learning
Stochastic differential equations
Fractional Brownian motion
Fractional Ornstein–Uhlenbeck process
Numerical simulation
Option pricing

ABSTRACT

Stochastic differential equations (SDEs) are widely used models to describe the evolution of stochastic processes. Among them, SDEs driven by fractional Brownian motion (fBm) have been shown to be capable of describing systems with temporal dependencies. In this paper, we develop a neural network based Monte Carlo methodology in which we can efficiently simulate SDEs that are governed by fBm. Particularly, we focus on large time step simulations. A property of fBm that complicates the development of such Monte Carlo schemes is the long-range temporal correlation. To this end, we build the network based on the encoder–decoder framework and employ the attention mechanism to learn the temporal relationships in the historical paths of such SDEs. In addition, a loss function based on the quantile loss is used, where the quantile levels to be predicted are determined by means of the stochastic collocation method. Experimental results show that this kind of loss function is superior to conventional loss functions in terms of solution accuracy, and the resulting scheme can learn and simulate SDEs driven by fBm accurately and highly efficiently.

1. Introduction

Stochastic differential equations (SDEs) are differential equations involving one or more terms that are stochastic processes. They are often employed to describe the behavior of systems that exhibit both regular and chaotic dynamics and are widely used in finance, physics, chemistry, and biology, for example, in [1–5].

Standard Brownian motion (sBm) with independent, normally distributed increments, is often the basic building block of SDEs giving rise to the well-known, memoryless Markov process properties. On the other hand, as a generalization of sBm, fractional Brownian motion (fBm) [6–8] has correlated increments and can describe systems with temporal dependencies. It is described by the Hurst index $H \in (0, 1)$ whereas sBm is governed by $H = 1/2$. fBm and SDEs driven by fBm play an increasingly important role in diverse application fields, such as weather forecast [9], life prediction [10], epidemic disease modeling [11], telecommunication traffic modeling [12,13], heterogeneous diffusion processes [14], etc. Especially, in finance, fBm has received a lot of attention in recent years. It may be not suitable to use fBm to describe stock dynamics directly, as fBm is not a semi-martingale and violates some basic properties of stock dynamics. However, we can use fractional processes to describe the dynamics of non-tradable assets and study the pricing problem of financial derivatives based on these non-tradable assets, for example, the weather [15], carbon

emission [16] and commodities [17]. Meanwhile, a large number of empirical findings show that the realized volatility is rough [18–21], which implies that the Hurst index H is less than $1/2$. Option pricing for rough volatility models is also studied in [22–24].

Numerical approximation plays a crucial role in solving SDEs, since analytical solutions are usually not available. Monte Carlo simulation, which requires a discrete approximation of the corresponding SDE and fine time grids to ensure convergence, is often the method of choice. Monte Carlo simulation can be slow to converge, meaning that many sample paths are required to achieve satisfactory accuracy.

Artificial neural networks have been used to solve differential equations in a data-driven way since the last century, for example, in [25]. And in recent years, with the rapid development of deep learning [26], many interesting developments have been presented in this field, such as the Physics-informed neural networks (PINN) [27] and the Deep Galerkin Method (DGM) [28]. For SDEs based on sBm, the authors of [29] propose a numerical SDE scheme based on the learning of the underlying conditional cumulative distribution functions (cCDFs). These cCDFs are learned through a data-driven approach that combines deep learning and the Stochastic Collocation Monte Carlo (SCMC) [30] method. Stochastic collocation points are deterministic optimal interpolation points, which can be learned by means of an

* Corresponding author.

E-mail address: jszhang@mail.xjtu.edu.cn (J. Zhang).

artificial neural network, to approximate the cCDFs with the help of interpolation techniques. With this paradigm, once sufficient training has taken place, accurate numerical simulation can be carried out with large time steps, provided the obtained cCDFs are accurately approximated. Since sBm has independent increments, the conditional distribution at the next time step relies only on the current value for the corresponding SDEs. The resulting scheme was named the Seven-League (7L) scheme, which is from the “seven-league boots fairytale” from European folklore.¹ Being able to simulate using large time steps will reduce the computational burden for processes that develop over corresponding large time scales.

Building upon this 7L scheme, here we develop a numerical scheme for SDEs that are driven by fBm and name the resulting scheme “7L-fBm”. For SDEs driven by fBm, previously processed values on the process path have an impact, because of the correlated increments of fBm. The fully-connected neural network used in the 7L scheme is no longer adequate in this case. To learn the temporal relationships in the historical paths of SDEs driven by fBm, we will use a neural network with an encoder-decoder structure [31,32], which is widely used in sequence modeling problems. In addition, attention mechanisms [33–35] are incorporated within the encoder and between the encoder and decoder. Noticing that the collocation points to be learned are actually quantiles of the corresponding distributions, the quantile loss will be used as the learning objective function, which is connected to quantile regression [36] and is widely used in probabilistic forecasting [37–39]. We will show by means of numerical experiments that the resulting scheme outperforms traditional numerical schemes, especially when the employed Monte Carlo time step gets large.

This paper is organized as follows. Section 2 briefly reviews fBm, SDEs driven by fBm, and the SMC method. Section 3 describes the 7L-fBm scheme, including the complete process, the loss function and the architecture of the neural network. In Section 4, we present and analyze the experimental results. Section 5 concludes this paper.

2. Preliminaries

To fix notation, we briefly introduce fractional Brownian motion, stochastic differential equations driven by fBm, and the stochastic collocation method.

2.1. Fractional Brownian motion

Fractional Brownian motion, $\mathbf{B}^H = \{B_t^H, t \geq 0\}$, with Hurst index $H \in (0, 1)$, is defined on an appropriate probability space $(\Omega, \Sigma, \mathbb{P})$, satisfying the following properties [40]:

- (1) \mathbf{B}^H has stationary increments, i.e., $B_t^H - B_s^H \sim B_{t-s}^H$, for $0 \leq s \leq t$;
- (2) $B_0^H = 0$ and $\mathbb{E}[B_t^H] = 0$, for $t \geq 0$;
- (3) $\mathbb{E}[B_t^H]^2 = t^{2H}$, for $t \geq 0$;
- (4) B_t^H has a Gaussian distribution, for $t > 0$.

Here, \mathbb{E} denotes the expectation with respect to the probability measure \mathbb{P} .

Denoting by $\mathbf{X}^{H,\Delta t} = \{X_t^{H,\Delta t}, t \geq 0\}$ the increment process of \mathbf{B}^H with time step Δt ,

$$X_t^{H,\Delta t} := B_{t+\Delta t}^H - B_t^H, \quad (2.1)$$

the covariance between two non-overlapping increments $X_t^{H,\Delta t}$ and $X_s^{H,\Delta t}$ is given by

$$\begin{aligned} \text{Cov}(X_t^{H,\Delta t}, X_s^{H,\Delta t}) &= \mathbb{E}[(B_{t+\Delta t}^H - B_t^H)(B_{s+\Delta t}^H - B_s^H)] \\ &= \frac{1}{2} (|t-s+\Delta t|^{2H} - 2|t-s|^{2H} + |t-s-\Delta t|^{2H}). \end{aligned} \quad (2.2)$$

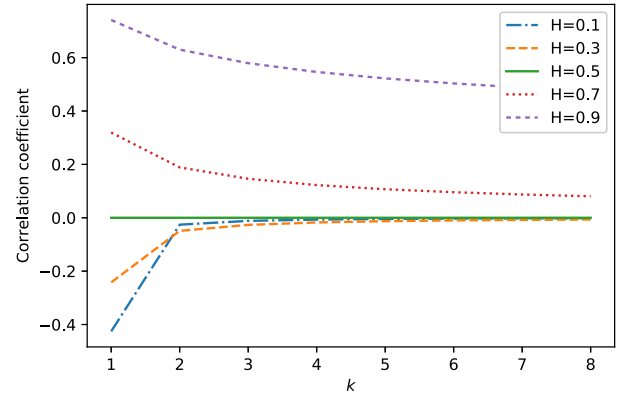


Fig. 1. Covariance coefficients of the increments of fBm $r(X_t^{H,\Delta t}, X_s^{H,\Delta t})$ with various Hurst index H and time distance k .

For sBm, a special case of fBm when $H = 1/2$, $\text{Cov}(B_t^{1/2}, B_s^{1/2}) = \min(t, s)$. Since

$$\text{Var}(X_t^{H,\Delta t}) = \text{Var}(X_s^{H,\Delta t}) = \text{Var}(B_{\Delta t}^H) = (\Delta t)^{2H}, \quad (2.3)$$

the correlation coefficient between $X_t^{H,\Delta t}$ and $X_s^{H,\Delta t}$ is found to be

$$r(X_t^{H,\Delta t}, X_s^{H,\Delta t}) = \frac{1}{2} (|k+1|^{2H} - 2|k|^{2H} + |k-1|^{2H}), \quad (2.4)$$

where $k := (t-s)/\Delta t$ indicates the number of time intervals between s and t .

The quantity $r(X_t^{H,\Delta t}, X_s^{H,\Delta t})$ is displayed for various H -values in Fig. 1. Clearly, for sBm, $r(X_t^{1/2,\Delta t}, X_s^{1/2,\Delta t}) = 0$ for any t and s . Increments are positively correlated when $H > 1/2$ and negatively correlated when $H < 1/2$. The absolute values of the covariance coefficients decrease with increasing k -values. From the perspective of the task of learning fBm, conditional on the realized history path $\{B_t^H\}_{t=0}^s$, $B_{s+\Delta t}^H$ depends only on B_s^H when $H = 1/2$, but also on previous values when $H \neq 1/2$. Especially, when $H > 1/2$, $r(X_t^{H,\Delta t}, X_s^{H,\Delta t})$ decays slowly as k increases, which is known as the long-range dependence property of fBm, and therefore historical paths will have a significant impact on $B_{s+\Delta t}^H$.

Properties of fBm have been presented in detail in [41] and [42]. For the Monte Carlo simulation of fBm, we refer to [43] and the Python package “fbm”.²

2.2. Stochastic differential equations driven by fractional Brownian motion

The general form of SDEs driven by fractional Brownian motion reads

$$dY_t = a(t, Y_t, \theta)dt + b(t, Y_t, \theta)dB_t^H, \quad t \in [0, T], \quad (2.5)$$

with model parameters θ , drift term $a(t, Y_t, \theta)$, diffusion term $b(t, Y_t, \theta)$, initial value Y_0 , and fBm B_t^H with Hurst index $H \in (0, 1)$. In [44], it was proved that fBm is not a semi-martingale for $H \neq 1/2$. As a direct consequence, we cannot use the Itô calculus developed for semi-martingales to define the stochastic integral with respect to fBm. To overcome this problem, different stochastic integrals for fBm have been proposed based on the Wick product. We refer to [42] for their definitions, relations, and the conditions to guarantee the existence and uniqueness of the solution of Eq. (2.5).

As a specific case of Eq. (2.5), we consider the following fractional Ornstein-Uhlenbeck (fOU) SDE in this paper,

$$dY_t = \lambda(\mu - Y_t)dt + \sigma dB_t^H, \quad t \in [0, T], \quad (2.6)$$

¹ https://en.wikipedia.org/wiki/Seven-league_boots

² <https://pypi.org/project/fbm/>

with model parameters $\theta = \{\mu, \lambda, \sigma\}$, with μ representing the long-term mean value of the process, λ the speed of mean reversion, and σ the volatility. The dynamics of temperature are for example modeled by an fOU process in [45], since empirical observations show long-range temporal temperature correlations. As the diffusion term, σ , is deterministic, the Wick product coincides with the original product here.

It was proved in [46] that under the filtration \mathcal{F}_s ($s \geq 0$), Y_t ($t \geq s$) is normally distributed, i.e.,

$$Y_t | \mathcal{F}_s = \mathbb{E}[Y_t | \mathcal{F}_s] + \sqrt{\text{Var}[Y_t | \mathcal{F}_s]} X, \quad (2.7)$$

where $X \sim \mathcal{N}(0, 1)$, and the conditional expectation and variance read

$$\mathbb{E}[Y_t | \mathcal{F}_s] = Y_s e^{-\lambda(t-s)} + \mu(1 - e^{-\lambda(t-s)}) + \int_0^s \Psi_c(s, t, v) dB_v^H, \quad (2.8)$$

and

$$\text{Var}[Y_t | \mathcal{F}_s] = \|c(r) \mathbf{1}_{[s,t]}(r)\|_{\kappa, T}^2 - \|\Psi_c(s, t, v) \mathbf{1}_{[0,s]}(v)\|_{\kappa, T}^2, \quad (2.9)$$

respectively, where $\kappa := H - 1/2$, $c(r) = \sigma e^{-\lambda(t-r)}$, and

$$\Psi_c(s, t, v) = \begin{cases} \frac{\sin(\pi\kappa)}{\pi} v^{-\kappa} (s-v)^{-\kappa} \int_s^t \frac{r^\kappa (r-s)^\kappa}{r-v} c(r) dr, & v \in (0, s), \\ 0, & \text{else.} \end{cases} \quad (2.10)$$

The norm of a function f is defined by the fractional Riemann–Liouville integral for $\kappa \neq 0$

$$\|f\|_{\kappa, T}^2 = \frac{\pi\kappa(2\kappa+1)}{\Gamma(1-2\kappa)\sin(\pi\kappa)\Gamma^2(\kappa)} \int_0^T z^{-2\kappa} \left(\int_z^T \frac{r^\kappa f(r)}{(r-z)^{1-\kappa}} dr \right)^2 dz, \quad (2.11)$$

where $\Gamma(\cdot)$ is the gamma function, and set as the L_2 norm for $\kappa = 0$

$$\|f\|_{0, T}^2 = \int_0^T f^2(r) dr. \quad (2.12)$$

Since the numerical simulation of conditional distributions is the main objective here, Eqs. (2.8) and (2.9) form the theoretical basis for the experiments in this paper. These integration formulas are nontrivial when considering all Hurst indices $H \in (0, 1)$, due to the singularities in the integrand of Eq. (2.11). We refer to [47] for a method to evaluate them accurately.

2.3. Stochastic collocation

The basis for the proposed numerical scheme in this paper is found in the stochastic collocation technique, which we will briefly explain here. For a variable of interest, Y , with cumulative distribution function (CDF) $F_Y(y)$, we aim to generate the samples efficiently.

Since $F_Y(y)$ is a monotonically increasing function in $[0, 1]$, we have

$$\mathbb{P}[F_Y(y) \leq u] = \mathbb{P}[Y \leq F_Y^{-1}(u)] = F_Y(F_Y^{-1}(u)) = u, \quad (2.13)$$

for any $u \in [0, 1]$, which implies that $F_Y(Y)$ is uniformly distributed in $[0, 1]$.

As a standard approach, samples of Y can therefore be obtained by sampling from a uniform distribution and inverting the CDF,

$$\hat{Y} = F_Y^{-1}(u). \quad (2.14)$$

However, the inversion $F_Y^{-1}(\cdot)$ is usually computationally expensive, especially when it is not known analytically. To reduce the number of expensive inversions, interpolation polynomials are adopted that are based on a “cheap to invert” variable X (which means samples of X are easy to obtain), for example, a normal distribution. As Eq. (2.13) is also valid for the CDF $F_X(X)$, the two variables can be connected via

$$F_Y(Y) \stackrel{d}{=} U \stackrel{d}{=} F_X(X), \quad (2.15)$$

where $U \sim \mathcal{U}(0, 1)$. For each sample \hat{Y} of Y , there is a corresponding \hat{X} of X satisfying

$$\hat{Y} = F_Y^{-1}(F_X(\hat{X})). \quad (2.16)$$

By means of a function $g(\cdot) = F_Y^{-1}(F_X(\cdot))$, which yields $\hat{Y} = g(\hat{X})$, samples of Y can be obtained by sampling from X and performing the transformation by function $g(\cdot)$. The function $g(\cdot)$ is then accurately approximated using suitable interpolation methods on specific points of X , “the stochastic collocation points”, and corresponding values of Y . The optimal collocation points can be determined based on the moments of X to get a stable and accurate interpolation.

Once the m collocation points $\{x_1, x_2, \dots, x_m\}$ and the corresponding values $\{y_1, y_2, \dots, y_m\}$ have been determined, we derive the interpolation function $g(\cdot)$ and can generate any number of samples of Y by sampling from X . The above methodology is named the Stochastic Collocation Monte Carlo (SCMC) method and was developed in [30]. It is also shown in [30] that highly accurate results can be obtained with only a small number of collocation points. In this work, we also base the methodology on X being a standard normal variable, with the corresponding optimal collocation points provided in Table A1 of [30].

3. Methodology

To build a numerical scheme to approximate the solution to the SDEs, we consider the conditional distributions of corresponding stochastic processes.

With an equally spaced partition of the time interval $[0, T]$, $t_i = i \cdot \Delta t$, $i = 0, 1, \dots, N$, where the time step $\Delta t = T/N$, the distribution of variable $Y_{t_{i+1}}$ under a realized historical path $\{Y_0, \hat{Y}_1, \dots, \hat{Y}_i\}$ is denoted by $p(Y_{t_{i+1}} | \hat{Y}_1, \hat{Y}_{t_{i-1}}, \dots, Y_0, \theta, H, \Delta t)$. When the conditional distributions are known for any $i \in \{0, 1, \dots, N-1\}$, the paths of the stochastic processes can be obtained by sequentially sampling from the corresponding conditional distributions, which gives us a numerical scheme for SDEs. Moreover, with sufficiently accurate conditional distributions, the scheme obtained is expected to perform well even when using large time steps, which is a significant advantage compared to traditional (Taylor-based) numerical SDE schemes.

The scheme includes an offline training stage and an online generation stage. In the offline stage, a model needs to be built and trained with a machine learning technique, which then outputs the conditional distributions with the input of the given historical paths. Then, in the generation stage, we will employ the trained model to generate Monte Carlo paths.

3.1. Offline and online stages

For most SDEs, a closed-form expression of the conditional distribution is not available. We therefore employ the SCMC method described in Section 2.3 to accurately approximate the conditional distributions, where we transform a conditional variable into a function of the given variable

$$Y_{t_{i+1}} | \hat{Y}_1, \hat{Y}_{t_{i-1}}, \dots, Y_0, \theta, H, \Delta t = g(X; \hat{Y}_1, \hat{Y}_{t_{i-1}}, \dots, Y_0, \theta, H, \Delta t), \quad (3.1)$$

with X chosen to be a standard normal variable. The function $g(\cdot)$ can be approximated in the form of an interpolation function based on several collocation points $\{x_1, x_2, \dots, x_m\}$ of X and corresponding values $\{y_{i+1,1}, y_{i+1,2}, \dots, y_{i+1,m}\}$ of $Y_{t_{i+1}}$. The objective is then to determine these values, which can be expressed as

$$y_{i+1,1}, y_{i+1,2}, \dots, y_{i+1,m} | \hat{Y}_1, \hat{Y}_{t_{i-1}}, \dots, Y_0, \theta, H, \Delta t = G(\hat{Y}_1, \hat{Y}_{t_{i-1}}, \dots, Y_0, \theta, H, \Delta t). \quad (3.2)$$

Neural networks are powerful function approximation mechanisms and can be employed to approximate the function $G(\cdot)$.

To train the network, a large number of paths need to be generated using classical numerical schemes, for example, the Euler–Maruyama

scheme. Algorithm 1 presents the process of data preparation for the training of the network. An important aspect here is that the time step used during the data generation, which is here denoted by $\Delta\tau$, should be set to a very small value to ensure the quality of training data.

Algorithm 1 7L-fBm scheme: Data preparation

Input: The value ranges of model parameters θ , Hurst index H , time step Δt , and initial value Y_0 of interest; terminal time T ; tiny time step for data generation $\Delta\tau$.

Output: Data set D

```

1: Initialize an empty data container  $D$ 
2: Determine all parameter combinations by sampling from their value ranges
3: for every parameter combination  $\{\theta, H, \Delta t, Y_0\}$  do
4:    $N = \lfloor T/\Delta t \rfloor$ 
5:   Simulate a path  $\{Y_0, \hat{Y}_{\Delta\tau}, \dots, \hat{Y}_T\}$  of the corresponding stochastic process using the classical numerical scheme with tiny time step  $\Delta\tau$ 
6:   for  $i = 0$  to  $N - 1$  do
7:     Store a sample  $\{\theta, H, \Delta t, Y_0, \hat{Y}_{\Delta\tau}, \dots, \hat{Y}_{i\Delta t}, \hat{Y}_{(i+1)\Delta t}\}$  in  $D$ 
8:   end for
9: end for
10: return  $D$ 

```

With a well-trained neural network, we can subsequently perform the numerical simulations following the steps presented in Algorithm 2.

Algorithm 2 7L-fBm scheme: Online generation stage

Input: Model parameters θ , Hurst index H , time step Δt , initial value Y_0 and terminal time T ; the well-trained neural network $G(\cdot)$; the number of collocation points m and collocation points $\{x_1, \dots, x_m\}$ of X .

Output: A path $\{Y_0, \hat{Y}_{\Delta t}, \dots, \hat{Y}_{N\Delta t}\}$

```

1:  $N = \lfloor T/\Delta t \rfloor$ 
2: for  $i = 0$  to  $N - 1$  do
3:    $y_{i+1,1}, y_{i+1,2}, \dots, y_{i+1,m} = G(\hat{Y}_{i\Delta t}, \hat{Y}_{(i-1)\Delta t}, \dots, Y_0, \theta, H, \Delta t)$ 
4:   Compute the interpolation function,  $y = g(x)$ , based on these  $m$  point pairs  $(x_j, y_{i+1,j})$ ,  $j = 1, \dots, m$ 
5:   Sample  $\hat{X}$  from  $X$ 
6:    $\hat{Y}_{(i+1)\Delta t} = g(\hat{X})$ 
7: end for
8: return  $\{Y_0, \hat{Y}_{\Delta t}, \dots, \hat{Y}_{N\Delta t}\}$ 

```

3.2. Quantile loss

As described in Eq. (3.2), by inputting model parameters θ of the SDE of interest, the Hurst index H , time step Δt and a realized historical path $\{Y_0, \hat{Y}_{t_1}, \dots, \hat{Y}_{t_i}\}$, the network should generate as output the corresponding collocation points $\{y_{i+1,1}, y_{i+1,2}, \dots, y_{i+1,m}\}$ of the variable at the next time point $Y_{t_{i+1}}$.

Eq. (2.16) can be rewritten as

$$F_Y(\hat{Y}) = q = F_X(\hat{X}), \quad (3.3)$$

which indicates that \hat{X} and \hat{Y} are in fact the q th quantiles of X and Y , respectively. From this point of view, the output of the network $\{y_{i+1,1}, y_{i+1,2}, \dots, y_{i+1,m}\}$ is given by the $\{q_1, q_2, \dots, q_m\}$ -th quantiles of $Y_{t_{i+1}}$, where $\{q_1, q_2, \dots, q_m\}$ are determined by $\{x_1, x_2, \dots, x_m\}$. In the case of X being the standard normal variable and $m = 3$, the collocation points $\{-1.732, 0, 1.732\}$ are the $\{0.042, 0.5, 0.958\}$ -th quantiles of X , and $\{y_{i+1,1}, y_{i+1,2}, y_{i+1,3}\}$ should then correspond to the $\{0.042, 0.5, 0.958\}$ -th quantiles of $Y_{t_{i+1}}$.

For this reason, the objective function for the training of the network will be the *quantile loss*, also known as pinball loss, which is

defined as

$$QL(y, y_{pred}, q) = q(y - y_{pred})^+ + (1 - q)(y_{pred} - y)^+, \quad (3.4)$$

where $(\cdot)^+ = \max(0, \cdot)$, y is the target value and y_{pred} is the output of the network. Intuitively, as q takes values between 0 and 1, the first term of Eq. (3.4) is positive and dominates when the network would under-predict (i.e. $y > y_{pred}$), whereas the second term dominates when the network over-predicts (i.e. $y < y_{pred}$). For $q = 0.5$, under-predictions and over-predictions are penalized by the same factor and the median is obtained. For $q > 0.5$, under-predictions are penalized by a larger factor than over-predictions. The network optimization will then strongly avoid under-predictions and a large quantile will be obtained. For $q < 0.5$, the opposite is expected to occur.

Besides being suitable for our problem, the quantile loss has the following advantages. First, using the quantile loss greatly improves the efficiency of data usage, as the $\hat{Y}_{(i+1)\Delta t}$ here is just a realization of variable $Y_{t_{i+1}}$ under the condition of $\{Y_0, \hat{Y}_{t_1}, \dots, \hat{Y}_{t_i}\}$. In contrast, error-based losses, such as the MSE loss used in [29], require knowledge of the true quantiles, which need to be estimated through multiple simulations. Second, it does not require any assumptions about the shape of the distribution, and can therefore be widely applied to various processes.

So, the network in our framework is trained by minimizing the quantile loss summed across all quantile output:

$$\mathcal{L} = \frac{1}{m} \sum_{j=1}^m QL(\hat{Y}_{(i+1)\Delta t}, y_{i+1,j}, q_j). \quad (3.5)$$

3.3. Neural network

Because sBm has independent increments, the distribution of $Y_{t_{i+1}}$ is fully determined by \hat{Y}_{t_i} . In that case, Eq. (3.2) degenerates into

$$y_{i+1,1}, y_{i+1,2}, \dots, y_{i+1,m} | \hat{Y}_{t_i}, \theta, \Delta t = G(\hat{Y}_{t_i}, \theta, \Delta t) \quad (3.6)$$

and a fully-connected neural network can be used to approximate the function $G(\cdot)$, as in [29].

Since fBm has, especially for $H > 1/2$, long-range dependence, the network should be able to handle time series data and a fully-connected neural network is no longer a suitable choice.

For this reason, we develop a more complex network in this paper. Generally speaking, this network has an encoder-decoder structure. The encoder takes a historical path $\{Y_0, \hat{Y}_{t_1}, \dots, \hat{Y}_{t_i}\}$, as well as the static features $\{\theta, H, \Delta t\}$, as input data and maps them onto hidden representations $\{Z_0, Z_{t_1}, \dots, Z_{t_i}\}$, that are passed to a decoder. Based on these hidden representations and the static features, the decoder then outputs the desired distribution of $Y_{t_{i+1}}$. Furthermore, it employs an attention mechanism to learn long- and short-term temporal relations in the input sequence.

Fig. 2 schematically presents the complete architecture of the network, with the key components described in detail in the subsequent subsections.

3.3.1. Attention mechanism

As shown in Fig. 3, there are three relevant concepts in the attention mechanism, namely the query, key, and value. Their meaning is similar to how they are used in recommendation systems [48]. For example, in a movie recommendation, the query is the user's preference information about the movie (such as age, gender, interest point, etc.), the key is the type of movie (comedy, science, romance, etc.), and the value is the movie to be recommended.

Given a query c , which is a vector of dimension d_c , and a set of key-value pairs (k_n, v_n) , $n = 1, 2, \dots, N$, where k_n and v_n are vectors of dimension d_k and d_v respectively, the general idea of the attention mechanism is to determine the attention weights of c for each key-value pair, denoted as w_1, w_2, \dots, w_N , and output a weighted sum of

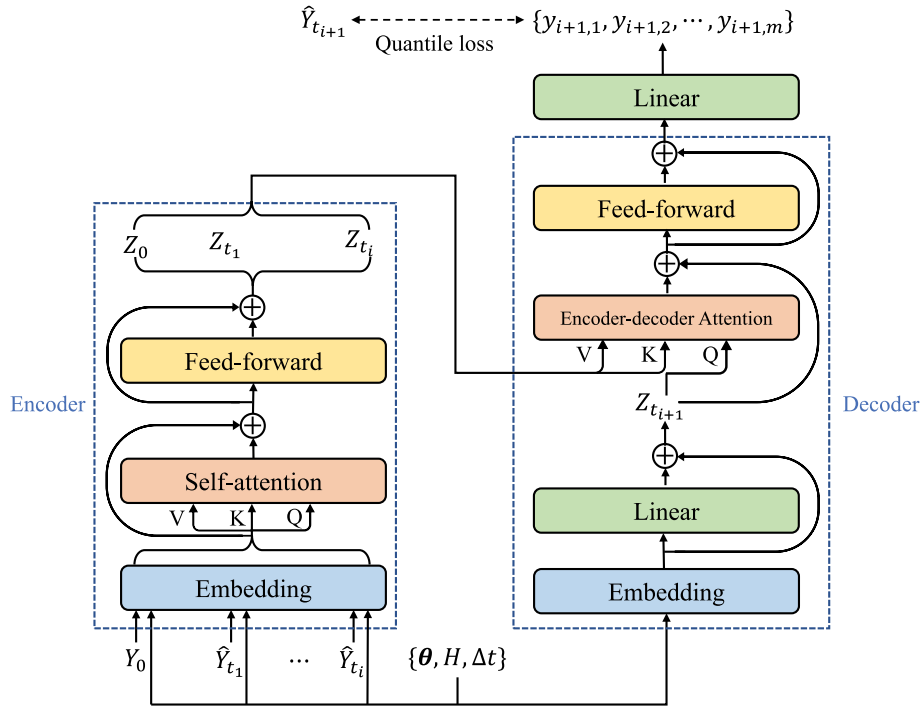


Fig. 2. Neural network architecture of 7L-fBm. The network has an encoder-decoder structure. The encoder takes the historical path $\{Y_0, \hat{Y}_1, \dots, \hat{Y}_i\}$ and static features $\{\theta, H, \Delta t\}$ as input, where the input at position j is $\{\hat{Y}_j, \theta, H, \Delta t\}$, $j = 0, 1, \dots, i$. Through the embedding layer, self-attention layer, and feed-forward layer, the encoder outputs hidden representations of the input sequence $\{Z_0, Z_{t_1}, \dots, Z_{t_i}\}$. The decoder first transforms static features to a vector $Z_{t_{i+1}}$, then performs the encoder-decoder attention together with the output of the encoder $\{Z_0, Z_{t_1}, \dots, Z_{t_i}\}$. Finally, the network outputs the predicted quantiles $\{y_{i+1,1}, y_{i+1,2}, \dots, y_{i+1,m}\}$ for the distribution of $Y_{t_{i+1}}$ and the quantile loss can be calculated by comparing with the realized path value $\hat{Y}_{t_{i+1}}$.

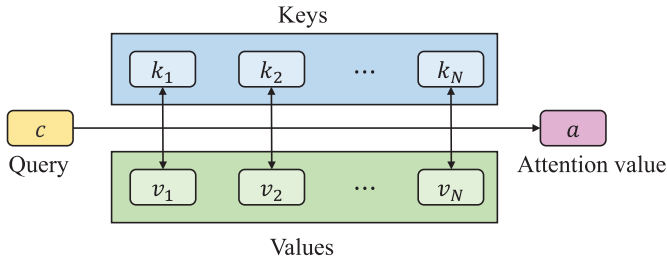


Fig. 3. A diagram of the attention mechanism.

the values:

$$a = \sum_{n=1}^N w_n v_n. \quad (3.7)$$

The attention weights can be determined by the similarity between the query and keys. That is, when the query and the key are “similar”, the weight is large. There are different ways to compute the similarity and the scaled dot-product similarity is employed here. The attention weights are then obtained by computing the dot products of the query with all keys, dividing them by the square root of the dimension of the query, and applying a softmax function:

$$\hat{w}_n = \frac{c \cdot k_n}{\sqrt{d_c}}, \quad n = 1, 2, \dots, N, \quad (3.8)$$

$$w_n = \frac{\exp(\hat{w}_n)}{\sum_{j=1}^N \exp(\hat{w}_j)}, \quad n = 1, 2, \dots, N. \quad (3.9)$$

In practice, we can compute several attention values simultaneously as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_c}}\right)V, \quad (3.10)$$

where $Q \in \mathbb{R}^{M \times d_c}$ is the matrix packed by M queries, $K \in \mathbb{R}^{N \times d_c}$ and $V \in \mathbb{R}^{N \times d_v}$ are the matrices packed by N keys and values, respectively.

In encoder-decoder architectures, the attention mechanism is used initially between the encoder and decoder in the form of “encoder-decoder attention” [33,34]. Specifically, the queries come from the decoder, while the keys and values come from the encoder. It then enables the decoder to utilize the most relevant parts of the input sequence dynamically. Later in the well-known Transformer model [35], the attention mechanism is used also in the encoder and decoder to handle sequence data in the form of “self-attention”, where the queries, keys, and values all come from the encoder or the decoder. Compared with the recurrent layers commonly used in earlier settings, models with self-attention enable fast parallel processing and require less time to train.

3.3.2. Layers of the network

As shown in Fig. 2, there are four kinds of layers in the network.

- **Embedding layer.** The embedding layer is a fully-connected layer and is used to convert the input tokens to vectors of dimension d_{model} at the beginning of the encoder and the decoder. Here, $d_{model} = d_c = d_v$.
- **Attention layer.** The attention layer is used in both the encoder and the decoder but in different ways. In the encoder, the attention layer is called “self-attention”, as queries, keys, and values all come from the embeddings of the encoder, i.e.,

$$Q = FC_Q^1(x_{in}), \quad K = FC_K^1(x_{in}), \quad \text{and} \quad V = FC_V^1(x_{in}), \quad (3.11)$$

where x_{in} is the output of the embedding layer of the encoder, FC_Q^1 , FC_K^1 , and FC_V^1 are three fully-connected layers. In the decoder, we use “encoder-decoder attention”, where the query comes from the previous layer of the decoder, while the keys and values come from the output of the encoder, i.e.,

$$Q = FC_Q^2(Z_{t_{i+1}}), \quad K = FC_K^2(Z), \quad \text{and} \quad V = FC_V^2(Z), \quad (3.12)$$

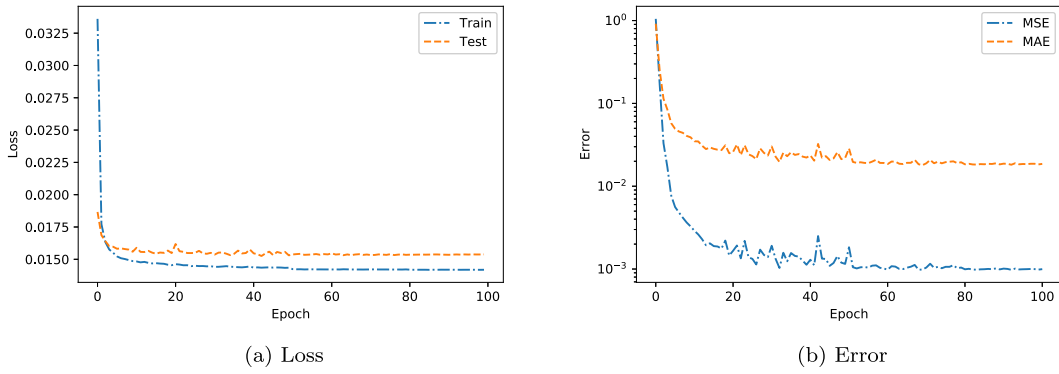


Fig. 4. The change curves of some metrics in the training process. (a) Losses in the training and test set after every epoch. (b) MSE and MAE in the test set after every epoch.

where Z is the matrix packed by $\{Z_0, Z_{t_1}, \dots, Z_{t_i}\}$. In this way, the decoder is able to attend to all positions in the encoder.

- **Feed-forward layer.** The feed-forward layers are placed after the attention layers of the encoder and the decoder to further process the information. It includes two linear transformations with a ReLU activation in between, i.e.,

$$x_{out} = \max(0, x_{in}W_1 + b_1)W_2 + b_2, \quad (3.13)$$

where $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$ and $W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$ are the weights, $b_1 \in \mathbb{R}^{d_{ff}}$ and $b_2 \in \mathbb{R}^{d_{model}}$ are the bias.

- **Linear layer.** There are two linear layers in the network. One is located behind the embedding layer in the decoder to obtain the hidden representation of position t_{i+1} , $Z_{t_{i+1}}$. The other acts as the output layer, converting the output of the decoder into a vector of dimension m .

Residual connections [49] are employed over all layers except the embedding and output layers, followed by layer normalization [50], i.e.,

$$x_{out} = \text{LayerNorm}(x_{in} + f(x_{in})), \quad (3.14)$$

where $f(\cdot)$ is a function of the corresponding layer.

4. Numerical experiments

In this section, we conduct a series of experiments to study the performance of the proposed scheme.

The experiments are implemented for the fOU process. Since the diffusion parameter of this process, σ , does not depend on Y_t , the traditional numerical schemes, including the Euler and Milstein schemes, are identical, and are given by,

$$\hat{Y}_{t_{i+1}} = \hat{Y}_{t_i} + \lambda(\mu - \hat{Y}_{t_i})\Delta t + \sigma(\hat{B}_{t_{i+1}} - \hat{B}_{t_i}). \quad (4.1)$$

4.1. Training details

As explained in Section 4, we first prepare the training data following Fig. 1. In the experiments, the terminal time T is set to 4, the tiny time step Δt is set to 0.01, and the initial value Y_0 is fixed at 1. Table 1 summarizes the other parameter value ranges and sampling methods. For the fOU process, the model parameters θ include the long-term mean value μ , the speed of mean reversion λ , and the volatility σ . We use the Latin Hyper-cube Sampling (LHS) method [51] to generate the random points in the domain of their value ranges. The value space of the Hurst index H is set to 9 evenly spaced samples in $[0.1, 0.9]$ here, i.e., $H \in \{0.1, 0.2, \dots, 0.9\}$, and $\Delta t \in \{0.1, 0.2, \dots, 1.0\}$. Noting that given a path, more data can be obtained with a smaller Δt , we compute more points of θ for larger Δt to balance the number of data with different Δt values in the data set. Specifically, the numbers of θ for Δt from 0.1 to 1.0 are 100, 200, 307, 400, 500, 666, 800, 800, 1000, 1000. With these

Table 1

Value ranges and sampling methods of parameters during data preparation.

Parameters		Value range	Method
Model parameters θ	Long-term mean μ	$[0.5, 1.5]$	LHS
	Speed of mean reversion λ	$(0, 0.3]$	LHS
	Volatility σ	$(0, 0.3]$	LHS
Time step Δt		$[0.1, 1.0]$	Equidistant
Hurst index H		$[0.1, 0.9]$	Equidistant

settings, we obtain approximately 360,000 data points and divide them into a training set (80%) and a test set (20%).

The network is implemented in PyTorch [52] and trained for 100 epochs with batch size 1024. We use the Adam optimizer [53]. The learning rate is initially set to 10^{-3} and reduced by a factor of 0.1 after 50 and 80 epochs. The dimension of the embedding tokens d_{model} , as well as the dimension of queries and values, is set to 48, and the dimension of the inner layer of the feed-forward layers d_{ff} is set to 64. The number of collocation points m is set to 5. The losses in the training and test sets during the training process are shown in Fig. 4(a).

We further investigate the performance of the model by comparison with theoretical results. The mean square error (MSE) and mean absolute error (MAE) in the test set after every epoch are shown in Fig. 4(b). They are computed as

$$\text{MSE} = \frac{1}{m} \sum_{j=1}^m (y_{i+1,j} - \bar{y}_{i+1,j})^2, \quad \text{and} \quad \text{MAE} = \frac{1}{m} \sum_{j=1}^m |y_{i+1,j} - \bar{y}_{i+1,j}| \quad (4.2)$$

respectively, where $\bar{y}_{i+1,j}$ is the theoretical q_j -th quantile of the distribution of $Y_{t_{i+1}}$ under corresponding historical path $\{Y_0, \hat{Y}_{t_1}, \dots, \hat{Y}_{t_i}\}$. It is worth noting that $\bar{y}_{i+1,j}$ is obtained from the analytical solution of the fOU process and has never been used in data generation and model training. Therefore, the comparison makes sense. As shown in Fig. 4(b), both MAE and MSE decrease and then stabilize to a small value as the training progresses, which indicates the effectiveness of the model.

4.2. Comparison with traditional scheme

In this part, we compare the performance of the 7L-fBm scheme with a traditional numerical scheme, especially the performance with different time steps. First, we generate many sample paths with Δt ranging from 0.1 to 1.0. Then, for each path, the distribution of the variable at the next time step can be estimated either by the network we have trained or by multiple sampling using a traditional numerical scheme. Their performance is compared by the MSE and MAE of their results and the theoretical results, which are presented in Tables 2 and 3, respectively. We can see that as Δt increases, the performance of the traditional numerical scheme deteriorates sharply, as expected, while the errors of the 7L-fBm scheme stay more-or-less constant. In general,

Table 2

MSE of $\{q_1, q_2, \dots, q_5\}$ -th quantiles with Δt from 0.1 to 1.0. The numbers in brackets are MSE (%) of the traditional numerical scheme and the 7L-fBm scheme, respectively.

Δt	q_1	q_2	q_3	q_4	q_5	Avg
0.1	(0.23, 0.08)	(0.21, 0.06)	(0.21, 0.06)	(0.21, 0.06)	(0.23, 0.07)	(0.22, 0.07)
0.3	(0.87, 0.12)	(0.76, 0.10)	(0.74, 0.09)	(0.76, 0.09)	(0.86, 0.11)	(0.80, 0.10)
0.5	(1.44, 0.12)	(1.20, 0.10)	(1.12, 0.09)	(1.18, 0.10)	(1.40, 0.12)	(1.27, 0.10)
0.7	(2.16, 0.12)	(1.64, 0.10)	(1.47, 0.10)	(1.62, 0.10)	(2.12, 0.12)	(1.80, 0.11)
0.9	(3.04, 0.14)	(2.16, 0.12)	(1.87, 0.12)	(2.14, 0.12)	(2.99, 0.14)	(2.44, 0.13)
1.0	(3.31, 0.15)	(2.33, 0.13)	(1.99, 0.12)	(2.30, 0.13)	(3.28, 0.16)	(2.64, 0.14)

Table 3

MAE of $\{q_1, q_2, \dots, q_5\}$ -th quantiles with Δt from 0.1 to 1.0. The numbers in brackets are MAE (%) of the traditional numerical scheme and the 7L-fBm scheme, respectively.

Δt	q_1	q_2	q_3	q_4	q_5	Avg
0.1	(3.21, 1.84)	(3.08, 1.57)	(3.03, 1.49)	(3.08, 1.54)	(3.22, 1.88)	(3.12, 1.67)
0.3	(6.60, 2.19)	(6.15, 1.92)	(5.90, 1.82)	(6.18, 1.84)	(6.62, 2.11)	(6.29, 1.98)
0.5	(8.59, 2.08)	(7.78, 1.89)	(7.21, 1.83)	(7.68, 1.89)	(8.47, 2.18)	(7.95, 1.97)
0.7	(10.36, 2.04)	(8.95, 1.83)	(8.01, 1.77)	(8.88, 1.85)	(10.26, 2.12)	(9.29, 1.92)
0.9	(12.16, 2.24)	(10.20, 1.96)	(8.90, 1.89)	(10.19, 1.99)	(12.13, 2.25)	(10.72, 2.07)
1.0	(12.68, 2.40)	(10.63, 2.06)	(9.23, 1.96)	(10.58, 2.07)	(12.68, 2.40)	(11.16, 2.18)

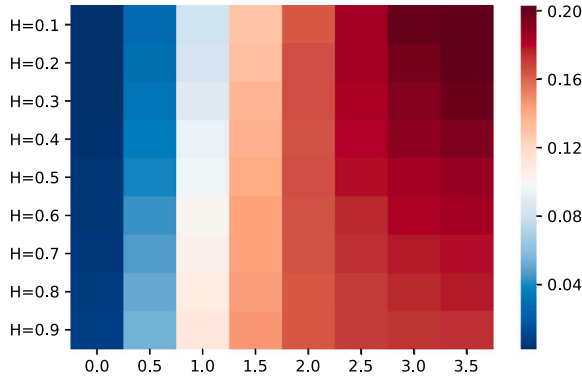


Fig. 5. Attention weights of Y_4 to $Y_0, \hat{Y}_{0.5}, \dots, \hat{Y}_{3.5}$ with Hurst index H from 0.1 to 0.9.

the 7L-fBm scheme outperforms the traditional scheme, even for small Δt .

4.3. Comparison with other networks

The network we propose, denoted here as ED-SA, has an encoder-decoder structure, where the encoder employs the attention mechanism to learn the temporal relationships in the input sequence. To show the effectiveness of the specific network, we compare its performance with other neural network set-ups in this subsection.

Firstly, the attention mechanism in the encoder is replaced by long short-term memory (LSTM) [54], a recurrent neural network that is well-known in sequence modeling, and the obtained network is denoted as ED-LSTM. Secondly, we test the encoder-decoder structure by removing the decoder and passing the hidden representation of the last time point, i.e., Z_{t_i} in Fig. 2, directly to the output layer. The obtained networks are denoted by SA and LSTM, where the encoders are based on self-attention and LSTM, respectively. The results are presented in Table 4, from which we can see that ED-SA outperforms the other network configurations.

Moreover, with the encoder-decoder structure, we can quantify the importance of each value in the historical path to the target variable. By fixing the time step Δt at 0.5 and the length of the historical path to 8 points, we collect the attention weights generated by the encoder-decoder attention of the decoder and average them over the samples based on their Hurst index H . The results are presented as a heatmap in Fig. 5. It clearly indicates that values closer to the target variable are

Table 4

Errors (%) in the test set of networks with different architectures. Due to limited space, we do not present errors of q_2 and q_4 , which show similar patterns with other quantiles.

Network	MSE				MAE			
	q_1	q_3	q_5	Avg	q_1	q_3	q_5	Avg
ED-SA	0.12	0.08	0.11	0.10	2.07	1.55	2.06	1.80
ED-LSTM	0.16	0.10	0.14	0.12	2.57	1.77	2.12	2.05
SA	0.14	0.10	0.15	0.12	2.19	1.62	2.44	1.94
LSTM	0.19	0.09	0.23	0.15	2.80	1.70	3.18	2.32

given more attention, for all values of H . Interestingly, by comparing the distribution of the attention weights for different H values, we observe that the attention paid to more distant values increases as H becomes larger. These findings align with the characteristics of the fBm process.

4.4. Loss function analysis

As described in Section 3.2, the network is trained by minimizing the quantile loss, where the quantile levels are set to the corresponding values of the optimal collocation points. In this part, we analyze the advantage of the loss function. First, we compare the quantile loss with the commonly used MSE loss and show that the efficiency of data usage is greatly improved by using the quantile loss. Second, the approach of determining the quantile levels is compared with the commonly used method. Experimental results show that the new methodology provides a better estimation of the overall distribution.

4.4.1. Comparison with MSE loss

As defined in Eq. (4.2), the computation of the MSE loss requires the theoretical quantile values, which need to be estimated through multiple simulations. Assuming quantile values on the condition of each historical path are estimated through N_p simulations, the amount of training data when using the quantile loss will be N_p times that of the MSE loss.

For fractional processes, the above data generation procedure is intractable, as there are too many possible historical paths. But for processes based on sBm, as in Eq. (3.6), only the last values in historical paths affect the prediction and therefore the procedure can be simplified by just varying the last value, instead of the whole path. Therefore, we compare the performance of quantile loss with MSE loss in the case of the usual OU process (i.e., Eq. (2.6) with $H = 1/2$). The network here is a fully-connected network with four hidden layers, each of which consists of 50 neurons and is followed by the ReLU activation function.

Table 5

MAE of $\{q_1, q_2, \dots, q_5\}$ -th quantiles with different loss functions. The numbers in brackets are MAE (%) when using MSE loss and quantile loss, respectively.

N_p	q_1	q_2	q_3	q_4	q_5	Avg
1000	(2.28, 1.38)	(1.64, 0.87)	(1.60, 0.80)	(1.84, 0.98)	(2.74, 1.39)	(2.02, 1.08)
100	(5.15, 2.12)	(1.89, 1.21)	(1.62, 1.01)	(2.15, 1.36)	(5.34, 2.47)	(3.23, 1.63)
10	(12.18, 3.59)	(2.44, 2.37)	(1.67, 2.01)	(3.39, 2.64)	(12.86, 4.81)	(6.49, 3.08)

Table 5 presents the MAE for different values of N_p . It is obvious that with the same number of simulated paths, the quantile loss achieves smaller errors than the MSE loss. Moreover, as N_p decreases, the targets for the MSE loss, especially the quantiles at the more extreme levels like q_1 and q_5 , cannot be accurately estimated by only a few paths, which leads to the poor performance of the MSE loss.

4.4.2. Comparison with multi-quantile forecasting

In most probabilistic forecasting works that utilize the quantile loss [37–39], the quantile levels to be predicted are predefined as some common values, with additional quantiles, as well as the overall distribution, derived by interpolation on these quantiles. This process is referred to as the MQ method later on. Our approach differs from these works in two ways. First, we base ourselves on the connection between the SCMC method and quantile forecasting and set the quantile levels to the corresponding values of the optimal collocation points. These collocation points are obtained based on the moments of X and lead to a stable interpolation. Second, we estimate the overall distribution through sufficient samples of it, which are obtained by sampling from X and performing the interpolation function.

In this subsection, we compare these two methods to see which one can estimate the overall distribution best. For the SCMC methodology, the quantile levels are given by $\{0.042, 0.5, 0.958\}$ when $m = 3$, and by $\{0.002, 0.088, 0.5, 0.912, 0.998\}$ when $m = 5$. For MQ, we find $\{0.1, 0.5, 0.9\}$ and $\{0.01, 0.25, 0.5, 0.75, 0.99\}$, respectively. We use two interpolation techniques: linear interpolation and Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) [55].

To evaluate the quality of the obtained distributions, we calculate the distances between the obtained and theoretical distributions by the following metrics.

- **Jensen–Shannon divergence.** The Jensen–Shannon divergence is a symmetric variant of the Kullback–Leibler divergence and it measures the difference between two probability distributions. It is defined as follows:

$$\text{JSD}(P \parallel Q) = \frac{1}{2} \text{KL}(P \parallel M) + \frac{1}{2} \text{KL}(Q \parallel M), \quad (4.3)$$

where P and Q are probability distributions, $M = (P + Q)/2$ is the midpoint distribution, and KL denotes the Kullback–Leibler divergence, i.e., the relative entropy of the first distribution with respect to the second distribution.

- **Wasserstein distance.** The Wasserstein distance, which is also known as the earth mover’s distance (EMD), is another important metric to measure the distance between two probability distributions. It is defined as the minimum cost of transforming one distribution into another distribution. For two one-dimensional distributions P and Q , the Wasserstein distance is computed as:

$$W(P, Q) = \int_0^1 |F_P^{-1}(s) - F_Q^{-1}(s)| ds, \quad (4.4)$$

where $F_P^{-1}(\cdot)$ and $F_Q^{-1}(\cdot)$ are the inverse cumulative distribution functions of P and Q , respectively.

The average results on the test set are presented in Table 6, where both the Jensen–Shannon divergence and Wasserstein distance of the proposed 7L-fBm scheme are smaller than for MQ. This indicates that the distributions provided by the new method are closer to the theoretical distributions according to the two metrics. Additionally, the

Table 6

Comparison of SCMC and MQ.

		PCHIP		Linear	
		Wasserstein	JS divergence	Wasserstein	JS divergence
$m = 3$	SCMC	0.0166	0.0552	0.0167	0.0554
	MQ	0.0227	0.0792	0.0227	0.0791
$m = 5$	SCMC	0.0163	0.0546	0.0162	0.0549
	MQ	0.0219	0.0638	0.0257	0.0777

MAE curves of the estimated quantiles at quantile level from 0 to 1 are plotted in Fig. 6, using the PCHIP interpolation. As is shown, for almost all quantile levels, especially for those close to 0 and 1, the 7L-fBm scheme estimates the quantiles more accurately. These results demonstrate that the new methodology provides a precise estimation of the overall distribution in this case.

4.5. Application in finance

Processes based on fBm are not semi-martingales, so the risk-neutral pricing formula traders usually use cannot be simply applied to financial derivatives based on tradable assets in this case. Fortunately, it is shown that with the Wick-based stochastic integration with respect to fBm and some restrictions on the corresponding market, (strong) arbitrage can be excluded [42]. On the other hand, the pricing of derivatives based on non-tradable assets, the weather for example, can still be done, where the real-world probability measure is often employed.

Here, we assume the dynamics of the asset are modeled as an fOU process and consider an Asian-style derivative based on it. The payoff of the specific Asian option is determined by the average value of the underlying asset over a certain period of time. For example, the payoff of a European-style Asian option with a fixed strike is given by

$$V(T) = \max[\eta(A(T) - K), 0] \quad \text{with} \quad \eta = \begin{cases} 1 & \text{for a call,} \\ -1 & \text{for a put,} \end{cases} \quad (4.5)$$

where T is the expiry time of the option contract, K is the strike, and $A(T)$ is the average value of the underlying asset over N_b observations,

$$A(T) = \frac{1}{N_b} \sum_{i=1}^{N_b} \hat{Y}_{t_i \Delta t}, \quad (4.6)$$

where $\Delta t = T/N_b$ is the time step.

It is obvious that $A(T)$ is path-dependent, and the same is true for the payoff $V(T)$. The accuracy of Asian option pricing then directly relies on the accuracy of the path simulation. We present the average errors of $A(T)$ over 10,000 paths in Table 7 and several of the generated paths in Fig. 7. For each path, the random samples of X are the same for the theoretical result (Eq. (2.7)), 7L-fBm scheme (2), and traditional numerical scheme (Eq. (4.1), where sampling from X happens in the process of the fBm generation). These results suggest that in most cases, the 7L-fBm scheme generates paths that are closer to the theoretical results compared to the traditional numerical schemes, particularly when the time step is large.

The ability to simulate with large time steps is certainly useful in computational finance. For example, in many financial contracts, the

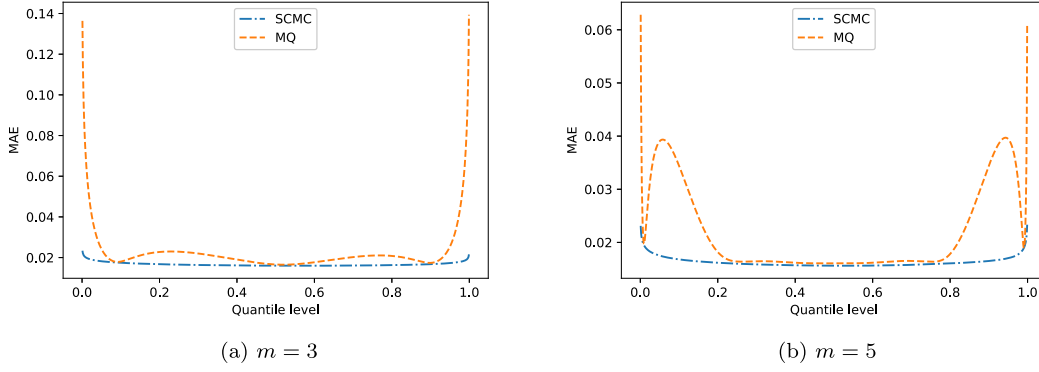


Fig. 6. MAE of estimated quantiles at quantile level from 0 to 1. (a) $m = 3$ (b) $m = 5$.

Table 7

MAE(%) of $A(T)$. The numbers in brackets are the results of the traditional numerical scheme and 7L-fBm scheme, respectively. Errors reported are the averages of 10,000 paths.

		$\Delta t = 0.4$	$\Delta t = 0.5$	$\Delta t = 0.8$	$\Delta t = 1.0$
$\sigma = 0.2, \lambda = 0.2$	$H = 0.5$	(0.44, 0.51)	(0.55, 0.42)	(0.87, 0.22)	(1.07, 0.21)
	$H = 0.7$	(1.00, 0.90)	(1.04, 0.88)	(1.28, 0.60)	(1.55, 0.59)
	$H = 0.9$	(4.23, 1.04)	(2.95, 0.86)	(2.97, 0.95)	(3.11, 0.96)
$\sigma = 0.1, \lambda = 0.1$	$H = 0.5$	(0.14, 0.35)	(0.21, 0.23)	(0.27, 0.15)	(0.34, 0.15)
	$H = 0.7$	(0.40, 0.42)	(0.42, 0.39)	(0.45, 0.37)	(0.49, 0.30)
	$H = 0.9$	(2.04, 0.53)	(1.70, 0.54)	(1.21, 0.56)	(1.11, 0.53)

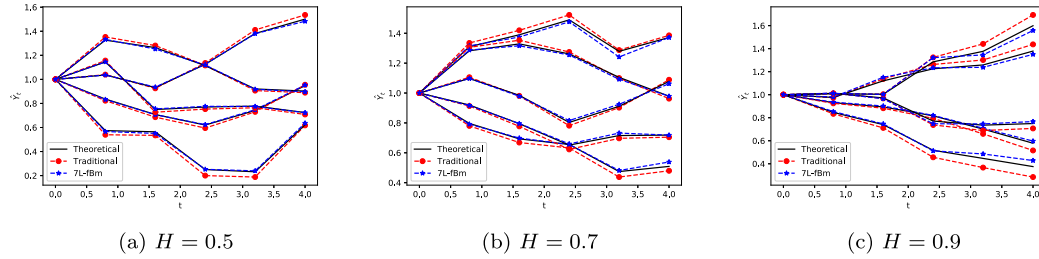


Fig. 7. Paths generated by different methods, with $\sigma = 0.2$, $\lambda = 0.2$, $\Delta t = 0.8$, and $Y_0 = 1$.

underlying assets are updated on a daily basis, whereas monitoring of the contracts and risk management is typically only done on a weekly, monthly, or even yearly basis.

5. Conclusion

In this paper, we develop the 7L-fBm scheme, a Monte Carlo simulation scheme for SDEs that are driven by fBm. It is generalized from the 7L scheme from [29] which deals with SDEs driven by sBm. Like the 7L scheme, the new scheme is also based on the learning of the underlying conditional distributions in a data-driven way. In contrast to sBm based SDEs, the path followed by the stochastic process is important for future time steps in the case of fBm. To deal with this phenomenon, the new scheme has the following two main changes. First of all, we build the neural network with the encoder-decoder structure and attention mechanism so that the path values of multiple previous time steps can be taken into account. Moreover, the huge number of possible historical paths makes it intractable to get the distribution under each path by multiple simulations. We replace the mean squared error loss with the quantile loss, which can be calculated on path values without knowing the distribution.

As evidenced by multiple numerical experiments, the new 7L-fBm scheme performs very well for the fOU process, including the case

of long range dependence ($H > 1/2$). Especially when the time step gets large, the 7L-fBm scheme shows a significant advantage over traditional schemes. Being able to simulate using large time steps will be highly beneficial for the pricing of path-dependent options and risk management in finance.

As a general methodology, it will be important to apply the 7L-fBm scheme to more processes. To do this, numerical schemes that are easy to implement are necessary to generate enough training data of high quality. This might be nontrivial when the diffusion term of the SDE depends on the state of the process, i.e., for the multiplicative case, because of the existence of Wick product, which cannot be calculated pathwise, but depends on all possible path [56]. The geometric fractional Brownian motion (GfBm) is an important process that belongs to this case. It replaces the geometric Brownian motion in the well-known Black-Scholes model and leads to the fractional Black-Scholes model. Finding suitable numerical schemes and applying the 7L-fBm scheme to such processes will be a meaningful research direction in the future.

CRedit authorship contribution statement

Fei Gao: Data curation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing. **Cornelis W. Oosterlee:** Writing – original draft, Writing – review & editing. **Jiangshe**

Zhang: Supervision, Resources, Funding acquisition, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 12371512).

References

- [1] Q. Liu, A.O. Khadidos, P. Wan, Discretization processing of financial risk management using stochastic differential equation simulation method, *Fractals* 30 (02) (2022) 2240069.
- [2] A.G. Cherstvy, D. Vinod, E. Aghion, I.M. Sokolov, R. Metzler, Scaled geometric Brownian motion features sub-or superexponential ensemble-averaged, but linear time-averaged mean-squared displacements, *Phys. Rev. E* 103 (6) (2021) 062127.
- [3] B. Kafash, R. Lalehzari, A. Delavarkhalafi, E. Mahmoudi, Application of stochastic differential system in chemical reactions via simulation, *MATCH Commun. Math. Comput. Chem.* 71 (2014) 265–277.
- [4] M. Baccouch, H. Temimi, M. Ben-Romdhane, A discontinuous Galerkin method for systems of stochastic differential equations with applications to population biology, finance, and physics, *J. Comput. Appl. Math.* 388 (2021) 113297.
- [5] A.P. Browning, D.J. Warne, K. Burrage, R.E. Baker, M.J. Simpson, Identifiability analysis for stochastic differential equation models in systems biology, *J. R. Soc. Interface* 17 (173) (2020) 20200652.
- [6] A.N. Kolmogorov, Wiener'sche Spiralen und einige andere interessante Kurven in Hilbertschem Raum, C. R. (Doklady), *Acad. Sci. URSS (NS)* 26 (1940) 115–118.
- [7] B.B. Mandelbrot, J.W. Van Ness, Fractional Brownian motions, fractional noises and applications, *SIAM Rev.* 10 (4) (1968) 422–437.
- [8] W. Deng, E. Barkai, Ergodic properties of fractional Brownian-Langevin motion, *Phys. Rev. E* 79 (1) (2009) 011112.
- [9] C.R. Rivero, D. Patiño, J. Pucheta, V. Sauchelli, A new approach for time series forecasting: Bayesian enhanced by fractional Brownian motion with application to rainfall series, *Int. J. Adv. Comput. Sci. Appl.* 7 (3) (2016).
- [10] H. Zhang, M. Chen, J. Shang, C. Yang, Y. Sun, Stochastic process-based degradation modeling and RUL prediction: from Brownian motion to fractional Brownian motion, *Sci. China Inf. Sci.* 64 (7) (2021) 171201.
- [11] M. Akinlar, M. Inc, J. Gómez-Aguilar, B. Boutarfa, Solutions of a disease model with fractional white noise, *Chaos Solitons Fractals* 137 (2020) 109840.
- [12] A. Pashko, Simulation of telecommunication traffic using statistical models of fractional Brownian motion, in: 2017 4th International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), IEEE, 2017, pp. 414–418.
- [13] M. Li, Generalized fractional Gaussian noise and its application to traffic modeling, *Phys. A: Stat. Mech. Appl.* 579 (2021) 126138.
- [14] W. Wang, A.G. Cherstvy, X. Liu, R. Metzler, Anomalous diffusion and nonergodicity for heterogeneous diffusion processes with fractional Gaussian noise, *Phys. Rev. E* 102 (1) (2020) 012146.
- [15] F.E. Benth, J. Saltyte-Benth, Modeling and Pricing in Financial Markets for Weather Derivatives, Vol. 17, World Scientific, 2012.
- [16] Z. Liu, S. Huang, Carbon option price forecasting based on modified fractional Brownian motion optimized by GARCH model in carbon emission trading, *North Am. J. Econ. Finance* 55 (2021) 101307.
- [17] S.N.I. Ibrahim, M. Misiran, M.F. Laham, Geometric fractional Brownian motion model for commodity market simulation, *Alex. Eng. J.* 60 (1) (2021) 955–962.
- [18] J. Gatheral, T. Jaisson, M. Rosenbaum, Volatility is rough, *Quant. Finance* 18 (6) (2018) 933–949.
- [19] X. Wang, W. Xiao, J. Yu, Modeling and forecasting realized volatility with the fractional Ornstein-Uhlenbeck process, *J. Econometrics* (2021).
- [20] M. Bennedsen, A. Lunde, M.S. Pakkanen, Decoupling the short-and long-term behavior of stochastic volatility, *J. Financ. Econom.* 20 (5) (2022) 961–1006.
- [21] M. Fukasawa, T. Takabatake, R. Westphal, Consistent estimation for fractional stochastic volatility model under high-frequency asymptotics, *Math. Finance* 32 (4) (2022) 1086–1132.
- [22] C. Bayer, P. Friz, J. Gatheral, Pricing under rough volatility, *Quant. Finance* 16 (6) (2016) 887–904.
- [23] C. Bayer, J. Qiu, Y. Yao, Pricing options under rough volatility with backward SPDEs, *SIAM J. Financial Math.* 13 (1) (2022) 179–212.
- [24] Z. Shi, P. Lyu, J. Ma, High-order methods for the option pricing under multivariate rough volatility models, *Comput. Math. Appl.* 139 (2023) 173–183.
- [25] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, *IEEE Trans. Neural Netw.* 9 (5) (1998) 987–1000.
- [26] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, *Nature* 521 (7553) (2015) 436–444.
- [27] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *J. Comput. Phys.* 378 (2019) 686–707.
- [28] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, *J. Comput. Phys.* 375 (2018) 1339–1364.
- [29] S. Liu, L.A. Grzelak, C.W. Oosterlee, The Seven-League Scheme: Deep learning for large time step Monte Carlo simulations of stochastic differential equations, *Risks* 10 (3) (2022) 47.
- [30] L.A. Grzelak, J.A. Witteveen, M. Suarez-Taboada, C.W. Oosterlee, The stochastic collocation Monte Carlo sampler: highly efficient sampling from 'expensive' distributions, *Quant. Finance* 19 (2) (2019) 339–356.
- [31] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 2, 2014, pp. 3104–3112.
- [32] K. Cho, B. Merriënboer, C. Gulcehre, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, in: EMNLP, 2014.
- [33] D. Bahdanau, K.H. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, in: 3rd International Conference on Learning Representations, 2015.
- [34] M.-T. Luong, H. Pham, C.D. Manning, Effective approaches to attention-based neural machine translation, in: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2015, pp. 1412–1421.
- [35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 6000–6010.
- [36] R. Koehler, K.F. Hallock, Quantile regression, *J. Econ. Perspect.* 15 (4) (2001) 143–156.
- [37] R. Wen, K. Torkkola, B. Narayanaswamy, D. Madeka, A multi-horizon quantile recurrent forecaster, in: 31st Conference on Neural Information Processing Systems (NIPS 2017), Time Series Workshop, 2017.
- [38] B. Lim, S.-Ö. Arık, N. Loeff, T. Pfister, Temporal fusion transformers for interpretable multi-horizon time series forecasting, *Int. J. Forecast.* 37 (4) (2021) 1748–1764.
- [39] Y. Chen, Y. Kang, Y. Chen, Z. Wang, Probabilistic forecasting with temporal convolutional neural network, *Neurocomputing* 399 (2020) 491–501.
- [40] I. Norros, A storage model with self-similar input, *Queueing Syst.* 16 (3) (1994) 387–396.
- [41] D. Nualart, Fractional Brownian motion: stochastic calculus and applications, in: International Congress of Mathematicians, Vol. 3, European Mathematical Society, 2006, pp. 1541–1562.
- [42] F. Biagini, Y. Hu, B. Øksendal, T. Zhang, Stochastic Calculus for Fractional Brownian Motion and Applications, Springer Science & Business Media, 2008.
- [43] T. Dieker, Simulation of fractional Brownian motion (Ph.D. thesis), Masters Thesis, Department of Mathematical Sciences, University of Twente, 2004.
- [44] L.C.G. Rogers, Arbitrage with fractional Brownian motion, *Math. Finance* 7 (1) (1997) 95–105.
- [45] D.C. Brody, J. Siroka, M. Zervos, Dynamical pricing of weather derivatives, *Quant. Finance* 2 (3) (2002) 189.
- [46] H. Fink, C. Klüppelberg, M. Zähle, Conditional distributions of processes related to fractional Brownian motion, *J. Appl. Probab.* 50 (1) (2013) 166–183.
- [47] F. Gao, S. Liu, C.W. Oosterlee, N.M. Temme, Evaluation of integrals with fractional Brownian motion for different Hurst indices, *Int. J. Comput. Math.* 100 (4) (2023) 847–866.
- [48] G. Cai, X. Li, Q. Dai, G. Wang, Z. Dong, C. Zhang, X. He, L. Shang, Dual sequence transformer for query-based interactive recommendation, in: 2021 22nd IEEE International Conference on Mobile Data Management, MDM, IEEE, 2021, pp. 139–144.
- [49] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
- [50] J.L. Ba, J.R. Kiros, G.E. Hinton, Layer normalization, in: Advances in Neural Information Processing Systems 2016 Deep Learning Symposium, 2016.
- [51] M.D. McKay, R.J. Beckman, W.J. Conover, A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, *Technometrics* 42 (1) (2000) 55–61.

- [52] 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, in: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 8026–8037.
- [53] D. Kingma, J. Ba, Adam: A method for stochastic optimization, in: *International Conference on Learning Representations, ICLR*, San Diego, CA, USA, 2015.
- [54] S. Hochreiter, J. Schmidhuber, Long short-term memory, *Neural Comput.* 9 (8) (1997) 1735–1780.
- [55] F.N. Fritsch, R.E. Carlson, Monotone piecewise cubic interpolation, *SIAM J. Numer. Anal.* 17 (2) (1980) 238–246.
- [56] C. Bender, P. Parczewski, On the connection between discrete and continuous wick calculus with an application to the fractional black-scholes model, in: *Stochastic Processes, Finance and Control: A Festschrift in Honor of Robert J Elliott*, World Scientific, 2012, pp. 3–40.



Fei Gao is currently pursuing the Ph.D. degree in the School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an, China. And she was an international scholar in the Mathematical Institute of Utrecht University, Utrecht, the Netherlands, from 2021 to 2022. Her research interests include deep learning and reinforcement learning.



Cornelis W. Oosterlee received the M.S. and Ph.D. degrees in Applied Mathematics from Delft University of Technology, the Netherlands, in 1989 and 1993, respectively. He has been a professor in the Mathematical Institute of Utrecht University since 2021, where he is also the head of the department now. Before that, he was a part-time professor in Delft University of Technology and also a scientist at the Centrum Wiskunde & Informatica (CWI) in Amsterdam. He has authored and coauthored two textbooks and approximately 150 research articles. His research focus is on computational problems in financial mathematics.



Jiangshe Zhang received the M.S. and Ph.D. degrees in Applied Mathematics from Xi'an Jiaotong University, Xi'an, China, in 1987 and 1993, respectively. He is currently a professor with the School of Mathematics and Statistics, Xi'an Jiaotong University. He has authored and coauthored one monograph and over 100 conference and journal publications. His research interests include Bayesian statistics, cognitive representation, statistical decision making and deep learning.