risks

Article

The Seven-League Scheme: Deep Learning for Large Time Step Monte Carlo Simulations of Stochastic Differential Equations †

Shuaiqiang Liu [1], Lech A. Grzelak [1,2] and Cornelis W. Oosterlee [1,3,*,‡]

1 Applied Mathematics (DIAM), Delft University of Technology, 2628 CD Delft, The Netherlands; shuaiqiang.liu@cwi.nl (S.L.); lech.grzelak@rabobank.com (L.A.G.)
2 Rabobank, 3500 HG Utrecht, The Netherlands
3 Centrum Wiskunde & Informatica (CWI), 1098 XG Amsterdam, The Netherlands
* Correspondence: c.w.oosterlee@uu.nl
† The views expressed in this paper are the personal views of the authors and do not necessarily reflect the views or policies of their current or past employers.
‡ Current address: Mathematical Institute, Utrecht University, 3584 CD Utrecht, The Netherlands.

Abstract: We propose an accurate data-driven numerical scheme to solve stochastic differential equations (SDEs), by taking large time steps. The SDE discretization is built up by means of the polynomial chaos expansion method, on the basis of accurately determined stochastic collocation (SC) points. By employing an artificial neural network to learn these SC points, we can perform Monte Carlo simulations with large time steps. Basic error analysis indicates that this data-driven scheme results in accurate SDE solutions in the sense of strong convergence, provided the learning methodology is robust and accurate. With a method variant called the compression–decompression collocation and interpolation technique, we can drastically reduce the number of neural network functions that have to be learned, so that computational speed is enhanced. As a proof of concept, 1D numerical experiments confirm a high-quality strong convergence error when using large time steps, and the novel scheme outperforms some classical numerical SDE discretizations. Some applications, here in financial option valuation, are also presented.

Keywords: artificial neural network; stochastic differential equations; large time step simulation; stochastic collocation Monte Carlo sampler; numerical scheme; path-dependent options

check for updates

Citation: Liu, Shuaiqiang, Lech A. Grzelak, and Cornelis W. Oosterlee. 2022. The Seven-League Scheme: Deep Learning for Large Time Step Monte Carlo Simulations of Stochastic Differential Equations. *Risks* 10: 47. https://doi.org/10.3390/risks10030047

Academic Editor: Mogens Steffensen

Received: 19 January 2022
Accepted: 16 February 2022
Published: 23 February 2022

1. Introduction

The highly successful deep learning paradigm (LeCun et al. 2015) receives a lot of attention in science and engineering. Within numerical mathematics, the machine learning methodology has, for example, successfully entered the field of numerically solving partial differential equations (PDEs) (Bar-Sinai et al. 2019; Beck et al. 2018; Han et al. 2018; Maziar et al. 2019; Sirignano and Spiliopoulos 2018; Xie et al. 2018). The aim with machine learning is then to either speed up the solution process or to solve high-dimensional problems that are not easily handled by the traditional numerical methods.

In this paper, we develop a highly accurate numerical discretization scheme for scalar *stochastic differential equations* (SDEs), which is based on taking possibly large discrete time steps. We "learn" to take large time steps, with the help of the stochastic collocation Monte Carlo sampler (SCMC) proposed by Grzelak et al. (2019), and by using an artificial neural network (ANN), within the classical supervised learning context.

SDEs are widely used to describe uncertain phenomena, in physics, finance, epidemics, amongst others, as a means to model and quantify uncertainty. The corresponding solutions are stochastic processes. Numerical approximation of the solution to an SDE is standard practice, as an analytic solution is typically not available. The most commonly known technique to solve SDEs is based on the Monte Carlo (MC) simulation, for which the SDE first needs to be discretized. There are quite a few applications that could benefit from an accurate and efficient numerical method on the basis of a large time step discretization (Li et al. 2021), for example, in finance, the valuation of path-dependent financial derivatives or financial risk management where counterparty credit risk plays a role.

Basically, there are two ways to measure the convergence rate of discrete solutions to SDEs, by means of the approximation to the sample path or by approximation to the corresponding distribution. This way, strong and weak convergence of a numerical SDE solutions have respectively been defined (see Platen 1999). Weak convergence, the convergence in distributional sense, is often addressed in the literature. Moment-matching, for example, is a basic technique used to improve weak convergence. Strong pathwise convergence is particularly challenging, and requires accurate *conditional distributions*. There are natural approaches to improve strong convergence properties, i.e., by adding higher order terms or by using finer time grids. However, these are nontrivial and costly, especially when considering multi-dimensional SDEs.

We aimed to develop highly accurate numerical schemes by means of deep learning, for which the strong error of the discretization does not depend on the size of the simulation time step. For this, we employ the SCMC method as an efficient approach for approximating (conditional) distribution functions. The distribution function of interest is expanded as a polynomial in terms of a random variable, which is *cheap to sample from* at given collocation points, and interpolation takes place between these points. The resulting big time step discretization, in which the SCMC methodology is combined with deep learning, is called *the Seven-League scheme*[1] here, and we abbreviate it by *the 7L scheme*.

There are different reasons to learn stochastic collocation points instead of the sample paths directly. Stochastic collocation points have a specific physical meaning, which makes the data-driven scheme explainable. Monte Carlo sample paths are random, while collocation points are path independent and deterministic (i.e., representing key features of a probability distribution), which simplifies the learning process when using neural networks. Unlike the SCMC method, which provides accurate Monte Carlo samples given a constant time step and for one specific instance of the SDE parameters, the 7L methodology enables us to generate samples for a wide range of time steps and for many different instances of the model parameters (i.e., for a family of SDEs), by means of a neural network to learn the evolution of the collocation points over time for many different model parameters.

The 7L scheme is composed of two separate phases under the framework of supervised learning, i.e., an offline (training) phase and an online (prediction) phase. The training phase, which usually requires heavy computation and many datasets, is done only once and offline. The prediction phase, which is a computationally cheap and highly efficient process, can be performed in an online fashion.

This paper serves to show that the proposed methodology performs very well for scalar SDEs. This work can thus be seen as a proof of concept. Higher-dimensional extensions of the 7L scheme will be presented in forth-coming work. Because this methodology is however "grid-based" we will not reach extremely high dimensions without additional enhancement. This is left for future work.

The remainder of this paper is organized as follows. In Section 2, SDEs, the discretization, stochastic collocation, and the connection between SDE discretizations and the SCMC method are introduced. In Section 3, the data-driven methodology is explained to address large time step simulation, i.e., the *7L scheme*, for SDEs. ANNs will be used as function approximators to learn the stochastic (conditional) collocation points. A brief description of their details is placed in Section 3.3. In Section 4, we introduce a decompression-compression technique to accelerate the computation. This latter efficient variant is named the *7L-CDC scheme* (i.e., seven-league compression–decompression scheme). Section 5 presents numerical experiments to show the performance of the proposed approach. Furthermore, the corresponding error is analyzed. Section 6 concludes.

2. Stochastic Differential Equations and Stochastic Collocation

We first describe the basic, well-known SDE setting, and explain our notation.

2.1. SDE Basics

We work with a real-valued random variable $Y(t)$, defined on the probability space $(\Omega, \Sigma, \mathbb{P})$ with filtration $\mathcal{F}_{t\in[0,T]}$, sample space Ω, σ-algebra Σ and probability measure $\mathbb{P}$. For the time evolution of $Y(t)$, consider the generic scalar Itô SDE,

$$\mathrm{d}Y(t) = a(t, Y(t), \boldsymbol{\theta})\mathrm{d}t + b(t, Y(t), \boldsymbol{\theta})\mathrm{d}W(t), \qquad 0 \le t \le T, \tag{1}$$

with the drift term $a(t, Y(t), \boldsymbol{\theta})$, the diffusion term $b(t, Y(t), \boldsymbol{\theta})$, model parameters $\boldsymbol{\theta}$, Wiener process $W(t)$, and given initial value $Y_0 := Y(t = 0)$. When the drift and diffusion terms satisfy some regularity conditions (e.g., the global Lipschitz continuity (Karatzas and Shreve 1988, p. 289)), the existence and uniqueness of the solution of (1) are guaranteed. The cumulative distribution function of $Y(t)$, $t \in [0, T]$, $F_{Y(t)}(\cdot)$ is available and the corresponding density function, evolving over time, is described by the Fokker–Planck equation (Risken 1984).

With a discretization in time interval $[0, T]$, $t_i = i \cdot T/N$, $i = 0, \ldots N$, with equidistant time step $\Delta t = t_{i+1} - t_i$, the discrete random variable at time t_i is denoted by $Y(t_i)$. Traditional numerical schemes have been designed based on Itô's lemma, in a similar fashion as the Taylor expansion is used to discretize deterministic ODEs and PDEs. The basic discretization, for each Monte Carlo path, is the Euler–Maruyama scheme (Platen 1999), which reads,

$$\hat{Y}_{i+1} = \hat{Y}_i + a(t_i, \hat{Y}_i, \boldsymbol{\theta})\Delta t + b(t_i, \hat{Y}_i, \boldsymbol{\theta})\sqrt{\Delta t}\hat{X}_{i+1}, \tag{2}$$

where $\hat{Y}_{i+1} := \hat{Y}(t_{i+1})$ is a realization (i.e., a number) from random variable $\tilde{Y}(t_{i+1})$, which represents the numerical approximation to exact solution $Y(t_{i+1})$ at time point t_{i+1}, and a realization $\hat{X}_{i+1}$ is drawn from the random variable X, which here follows the standard normal distribution $\mathcal{N}(0, 1)$. Moreover, $\check{Y}(t_i)$ (a number) will be used as the notation for a realization of $Y(t_i)$.

In addition, the Milstein discretization (Milstein 1975) reads,

$$\hat{Y}_{i+1} = \hat{Y}_i + a(t_i, \hat{Y}_i, \boldsymbol{\theta})\Delta t + b(t_i, \hat{Y}_i, \boldsymbol{\theta})\sqrt{\Delta t}\hat{X}_{i+1} + \frac{1}{2}b'(t_i, \hat{Y}_i, \boldsymbol{\theta})b(t_i, \hat{Y}_i, \boldsymbol{\theta})\Delta t(\hat{X}_{i+1}^2 - 1), \tag{3}$$

where $b'(t_i, \cdot, \boldsymbol{\theta})$ represents the derivative with respect to $\hat{Y}$ of $b(\cdot, \hat{Y}, \boldsymbol{\theta})$. When the drift and diffusion terms are independent of time t, the SDE is called time-invariant.

Two error convergence criteria are commonly used to measure the SDE discretization accuracy; that is, the convergence in the weak and strong sense. Strong convergence, which is of our interest here, is defined as follows.

Definition 1. *Let $Y(t_i)$ be the exact solution of an SDE at time t_i, its discrete approximation $\tilde{Y}(t_i)$ with time step $\Delta t \in \mathbb{R}^+$ converges in the strong sense, with order $\beta_s \in \mathbb{R}^+$, if there exists a constant K such that*

$$\mathbb{E}|\tilde{Y}(t_i) - Y(t_i)| \le K(\Delta t)^{\beta_s}. \tag{4}$$

It is well-known that the Euler–Maruyama scheme (2) has strong convergence $\beta_s = 0.5$, while the Milstein scheme (3) has $\beta_s = 1.0$. When deriving high-order schemes for SDEs, the rules of Itô calculus must be respected (Platen 1999). As a result, there will be eight terms in a Taylor SDE scheme with $\beta_s = 1.5$, and twelve with $\beta_s = 2.0$, and the computational complexity increases. As a consequence, higher order schemes are involved and somewhat expensive. Convergence of the numerical solution for $\Delta t \to 0$ is guaranteed, but the computational costs increase significantly to achieve accurate solutions.

The generic form of the above mentioned numerical schemes to solve the Itô SDE is as follows,

$$\hat{Y}_{i+1}|\hat{Y}_i = \sum_{j=0}^{m-1} \alpha_j(\hat{Y}_i)\hat{X}_{i+1}^j, \tag{5}$$

where m represents the number of polynomial terms, and the coefficients $\alpha_j(\hat{Y}_i)$ are pre-defined and equation-dependent. For the sake of notational convenience, let $\alpha_j = \alpha_j(\hat{Y}_i)$. For example, for the Euler–Maruyama scheme (2), with $m = 2$, we have

$$\begin{cases} \alpha_0 = \hat{Y}_i + a(t_i, \hat{Y}_i, \boldsymbol{\theta})\Delta t, \\ \alpha_1 = b(t_i, \hat{Y}_i, \boldsymbol{\theta})\sqrt{\Delta t}, \end{cases} \tag{6}$$

while for the Milstein scheme, with $m = 3$, it follows that

$$\begin{cases} \alpha_0 = \hat{Y}_i + a(t_i, \hat{Y}_i, \boldsymbol{\theta})\Delta t + \frac{1}{2}b'(t_i, \hat{Y}_i, \boldsymbol{\theta})b(t_i, \hat{Y}_i, \boldsymbol{\theta}), \\ \alpha_1 = b(t_i, \hat{Y}_i, \boldsymbol{\theta})\sqrt{\Delta t}, \\ \alpha_2 = \frac{1}{2}b'(t_i, \hat{Y}_i, \boldsymbol{\theta})b(t_i, \hat{Y}_i, \boldsymbol{\theta}). \end{cases} \tag{7}$$

With these explicit coefficients we arrive at the probability distribution of the random variable,

$$Y_{i+1}|Y_i \approx \tilde{Y}_{i+1}|\tilde{Y}_i \overset{d}{=} \sum_{j=0}^{m-1} \alpha_j X^j. \tag{8}$$

These discrete SDE schemes are based on a series of transformations of the previous realization to approximate the conditional distribution,

$$\mathbb{P}\big[Y(t+\Delta t) < y|Y(t)\big] = F_{Y(t+\Delta t)|Y(t)}(y) \approx F_{\tilde{Y}(t+\Delta t)|\tilde{Y}(t)}(y). \tag{9}$$

A numerical scheme is thus essentially based on the conditional sampling $Y(t+\Delta t)|Y(t)$. The Euler–Maruyama scheme draws from a normal distribution, with a specific mean and variance, to approximate the distribution in the next time point, while the Milstein scheme combines a normal and a chi-squared distribution. Similarly, we can derive the stochastic collocation methods.

2.2. Stochastic Collocation Method

Let us assume two random variables, Y and X, where the latter one is cheaper to sample from (e.g., X is a Gaussian random variable). These two scalar random variables are connected, via,

$$F_Y(Y) \overset{d}{=} U \overset{d}{=} F_X(X), \tag{10}$$

where $U \sim \mathcal{U}([0,1])$ is a uniformly distributed random variable, $F_Y(\bar{y}) := \mathbb{P}[Y \le \bar{y}]$ and $F_X(\bar{x}) := \mathbb{P}[X \le \bar{x}]$ are supposed to be cumulative distribution functions (CDFs). Note that $F_X(X)$ and $F_Y(Y)$ are random variables following the same uniform distribution. $F_Y(\bar{y}_n)$ and $F_X(\bar{x}_n)$ are supposed to be strictly increasing functions, so that the following inversion holds true,

$$\bar{y}_n = F_Y^{-1}(F_X(\bar{x}_n)) =: g(\bar{x}_n). \tag{11}$$

where $\bar{y}_n$ and $\bar{x}_n$ are samples (numbers) from Y and X, respectively. The mapping function, $g(\cdot) = F_Y^{-1}(F_X(\cdot))$, connects the two random variables and guarantees that $F_X(\bar{x}_n)$ equals $F_Y(g(\bar{x}_n))$, in distributional sense and also element-wise. The mapping function should be approximated, i.e., $g(\bar{x}_n) \approx g_m(\bar{x}_n)$, by a function which is computationally cheap. When function $g_m(\cdot)$ is available, we may generate "expensive" samples, $\bar{y}_n$ from Y, by using the cheaper random samples $\bar{x}_n$ from X.

The stochastic collocation Monte Carlo method (SCMC) developed in Grzelak et al. (2019) aims to find an accurate mapping function $g(\cdot)$ in an efficient way. The basic idea is to employ Equation (11) at specific collocation points and approximate the function $g(\cdot)$ by a suitable monotonic interpolation between these points. This procedure, see Algorithm 1, reduces the number of expensive inversions $F_Y^{-1}(\cdot)$ to obtain many samples from $Y(\cdot)$.

```
Algorithm 1: SCMC Method
  Taking an interpolation function of degree $m-1$ (with $m \geq 2$, as we need at least
   two collocation points), as an example, the following steps need to be performed:
  1. Calculate CDF $F_X(x_j)$ on the points $(x_1, x_2, \ldots, x_m)$, which are obtained,
     for example, from Gauss–Hermite quadrature, giving $m$ pairs $(x_j, F_X(x_j))$;
  2. Invert the target CDF $y_j = F_Y^{-1}(F_X(x_j))$, $j = 1, \ldots, m$, and form $m$ pairs $(x_j, y_j)$;
  3. Define the interpolation function, $y = g_m(x)$, based on these $m$ point pairs $(x_j, y_j)$;
  4. Obtain sample $\hat{Y}$ by applying the mapping function $\hat{Y} = g_m(\hat{X})$, where sample
     $\hat{X}$ is drawn from $X$.
```

The SCMC method parameterizes the distribution function by imposing probability constraints at the given collocation points. Taking the Lagrange interpolation as an example, we can expand function $g_m(\cdot)$ in the form of polynomial chaos,

$$Y \approx g_m(X) = \sum_{j=0}^{m-1} \hat{\alpha}_j X^j = \hat{\alpha}_0 + \hat{\alpha}_1 X + \ldots + \hat{\alpha}_{m-1} X^{m-1}. \tag{12}$$

Monotonicity of interpolation is an important requirement, particularly when dealing with peaked probability distributions.

The Cameron-Martin Theorem (Cameron and Martin 1947) states that any distribution can be approximated by a polynomial chaos approximation based on the normal distribution, but also other random variables may be used for X (see, for example, Grzelak et al. 2019).

Clearly, Equation (8) can be compared to Equation (12), as a discretization scheme to approximate the realization in the next time point.

3. Methodology

For our purposes, given $Y(t)$, the conditional variable $Y(t+\Delta t)$ can be written as,

$$Y(t+\Delta t)|Y(t) \approx g_m(X) = \sum_{j=0}^{m-1} \hat{\alpha}_j X^j, \tag{13}$$

where the coefficients $\hat{\alpha}_j \equiv \hat{\alpha}_j(\hat{Y}_i, t_i, t_{i+1}, \boldsymbol{\theta})$ are now functions of realization $\hat{Y}_i$. Equation (13), for large m-values, holds for any $\Delta t = t_{i+1} - t_i$, particularly also for large Δt. As such the scheme can be interpreted as an *almost exact simulation scheme* for an SDE under consideration. By the scheme in (13) we can thus take large time steps in a highly accurate discretization scheme. More specifically, a sample from the known distribution X can be mapped onto a corresponding unique sample of the conditional distribution $Y(t+\Delta t)$ by the coefficient functions.

There are essentially two possibilities for using an ANN in the framework of the stochastic collocation method, the first being to directly learn the (time-dependent) polynomial coefficients, $\hat{\alpha}_j$, in (13), the second to learn the collocation points, y_j. The two methods are equivalent mathematically, but the latter, our method of choice, appears more stable and flexible. Here, we explain how to learn the collocation points, y_i, which is then followed by inferring the polynomial coefficients. When the stochastic collocation points at time $t+\Delta t$ are known, the coefficients in (13) can be easily computed.

An SDE solution is represented by its cumulative distribution at the collocation points, plus a suitable accurate interpolation $g_m(x)$. In other words, the SCMC method forces the distribution functions (the target and the numerical approximation) to strictly match at the collocation points over time. The collocation points are dynamic and evolve with time.

3.1. Data-Driven Numerical Schemes

Calculating the conditional distribution function requires generating samples conditionally on previous realizations of the stochastic process. Based on a general polynomial expression, the conditional sample, in discrete form, is defined as follows,

$$\hat{Y}_{i+1}|\hat{Y}_i = \sum_{j=0}^{m-1} \hat{\alpha}_j\left(\hat{Y}_i, t_i, t_{i+1} - t_i, \boldsymbol{\theta}\right) \hat{X}_{i+1}^j, \tag{14}$$

where $\Delta t = t_{i+1} - t_i$, and the coefficients $\hat{\alpha}_j$, at time t_i, are functions of the variables $\hat{Y}_i, t_i, t_{i+1} - t_i, \boldsymbol{\theta}$, see Equations (6) and (7).

In the case of a Markov process, the future does not dependent on past values. Given $\hat{Y}(t)$, the random variable $\hat{Y}(t + \Delta t)$ only depends on the increment $Y(t + \Delta t) - Y(t)$. The process has independent increments, and the conditional distribution at time t_{i+1} given information up to time t_i only depends on the information at t_i.

Similar to these coefficient functions, the *m conditional* stochastic collocation points at time t_{i+1}, $y_j(t_{i+1})|\hat{Y}_i$, with $j = 0, \ldots, m-1$, can be written as a functional relation,

$$y_j(t_{i+1})|\hat{Y}_i = H_j\left(\hat{Y}_i, t_i, t_{i+1} - t_i, \boldsymbol{\theta}\right). \tag{15}$$

A closed-form expression for function $H(\cdot)$ is generally not available. Finding the conditional collocation points can however be formulated as a regression problem.

It is well-known that neural networks can be utilized as universal function approximators (Cybenko 1989). We then generate random data points in the domain of interest and the ANN should "learn the mapping function $H_j(\cdot)$", in an offline ANN training stage. The SCMC method is here used to compute the corresponding collocation points at each time point, which are then stored to train the ANN, in a supervised learning fashion (see, for example, Goodfellow et al. 2016).

3.2. The Seven-League Scheme

Next, we detail the generation of the stochastic collocation points to create the training data. Consider a stochastic process $Y(\tau)$, $\tau \in [0, \tau_{max}]$, where τ_{max} represents the maximum time horizon for the process that we wish to sample from. When the analytical solution of the SDE is not available (and we cannot use an exact simulation scheme with large time steps), a classical numerical scheme will be employed, based on *tiny constant time increments* $\Delta\tau = \tau_{i+1} - \tau_i$, a discretization in the time-wise direction with grid points $0 < \tau_1 < \tau_2 < \ldots < \tau_N \leq \tau_{max}$, to generate a sufficient number of highly accurate samples at each time point τ_i, to approximate the corresponding cumulative functions highly accurately. Note that the training samples are generated with a fixed time discretization, and in the case of a Markov process, one can easily obtain discrete values for the underlying stochastic process for many possible time increments, e.g., $2\Delta\tau$, $3\Delta\tau$. With the obtained samples, we approximate the corresponding marginal collocation points at time τ_i, as follows,

$$\hat{y}_j(\tau_i) = F_{\tilde{Y}_i}^{-1}(F_X(x_j)) \approx F_{Y_i}^{-1}(F_X(x_j)) \tag{16}$$

where, with integer M_s, $\hat{y}_j(\tau_i)$, $j = 1, \ldots, M_s$, represent the approximate collocation points of $Y(t)$ at time τ_i, and x_j are optimal collocation points of variable X. For simplicity, consider $X \sim \mathcal{N}(0, 1)$, so that the points x_j are known analytically and do not depend on time point τ_i. In the case of a normal distribution, these points are known quadrature points, and tabulated, for example, in (Grzelak et al. 2019). After this first step, we have the set of collocation points, $\hat{y}_j(\tau_i)$, for $i = 1, \ldots, N$ and $j = 1, \ldots, M_s$. Subsequently, the $\hat{y}_j(\cdot)$ from (16) are used as the ground-truth to train the ANN.

In the second step, we determine the *conditional* collocation points. For each time step τ_i and collocation point indexed by j, a *nested Monte Carlo simulation* is then performed to generate the conditional samples. Similar to the first step, we obtain the conditional collocation points from each of these sub-simulations using (16). With M_c being an integer representing the number of conditional collocation points, the above process yields the following set of M_c conditional collocation points,

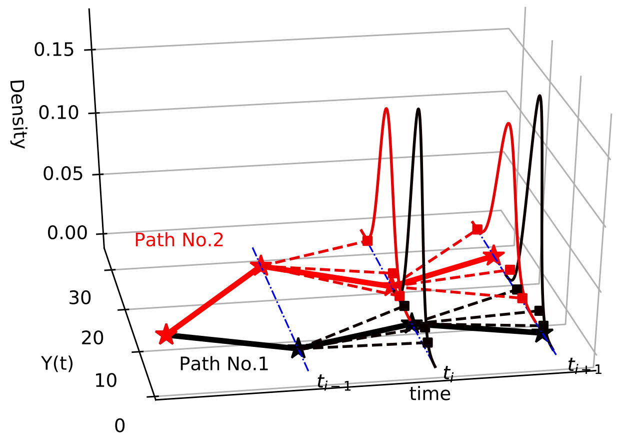
$$\hat{y}_{k|j}(\tau_{i+1}) := \hat{y}_k(\tau_{i+1})|\hat{y}_j(\tau_i) = F^{-1}_{\hat{Y}_{i+1}|\hat{Y}_i=\hat{y}_j(\tau_i)}\left(F_X(x_{k|j})\right), \tag{17}$$

where $x_{k|j}$ is a conditional collocation point, and $i \in \{0, 1, \ldots, N-1\}$, $j \in \{1, \ldots, M_s\}$, $k \in \{1, \ldots, M_c\}$. Note that, in the case of Markov processes, the above generic procedure can be simplified by just varying the initial value Y_0 instead of running a nested Monte Carlo simulation. Specifically, we then set $\hat{Y}_i = \hat{Y}_0$, $\tau_i = \tau_0$ and $\tau_{i+1} = \tau_0 + \Delta\tau$ to generate the corresponding conditional collocation points.

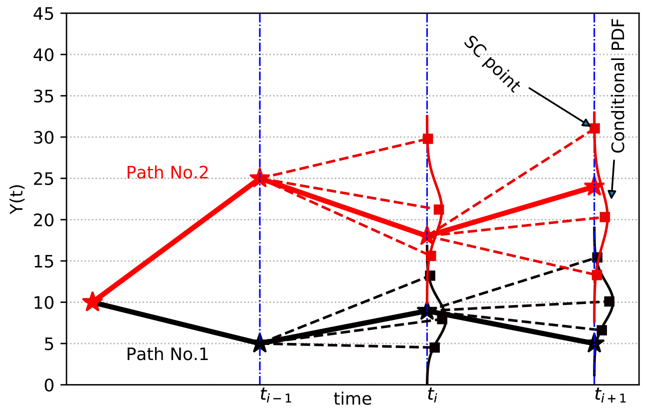
The inverse function, $F^{-1}_{\hat{Y}_{i+1}|\hat{Y}_i}(\cdot)$, is often not known analytically, and needs to be derived numerically. An efficient procedure for this is presented in (Grzelak 2019). Of course, it is well-known that the computation of $F^{-1}(p)$ is equivalent to the computation of the quantile at level p.

We encounter essentially four types of stochastic collocation (SC) points: x_j are called the original SC points, $\hat{x}_j$ are original conditional collocation points, $\hat{y}_j$ are the marginal SC points, and $\hat{y}_k|\cdot$ are the conditional SC points. For example, $\hat{y}_k|\hat{Y}_i$ is conditional on a realization $\hat{Y}_i$. In the context of Markov processes, the marginal SC points $\hat{y}_j$ only depend on the initial value Y_0, thus they are a special type of conditional collocation points, i.e., $\hat{y}_j|Y_0$. When a previous realization happens to be a collocation point, e.g., $\hat{Y}_i = \hat{y}_j$, we have $\hat{y}_{k|j} := \hat{y}_k|\hat{y}_j$, which will be used to develop a variation of the 7L scheme in Section 4.

When the data generation is completed, the ANNs are trained in a supervised-learning fashion, on the generated SC points to approximate the function H in (15), giving us a learned function $\hat{H}$. This is called the training phase. With the trained ANNs, we can approximate new collocation points, and develop a numerical solver for SDEs, which is the Seven-League scheme (7L), see Algorithm 2. Figure 1 gives a schematic illustration of Monte Carlo sample paths that are generated by the 7L scheme.



(**a**) sample paths generated by 7L



(**b**) the 2D projection of Figure 1a

Figure 1. Schematic diagram of the 7L scheme. Here conditional SC points, represented by ■, are conditional on a previous realization, denoted by ★. "Conditional PDF" is the conditional probability density function, defined by these conditional SC points. The density function, which is not required by 7L, is plotted only for illustration purposes.

Algorithm 2: 7L Scheme

```
1. Offline stage: train the ANNs to learn the stochastic collocation points. At this stage, we choose different $\boldsymbol{\theta}$ values, simulate corresponding Monte Carlo paths, with small constant time increments $\Delta\tau = \tau_{i+1} - \tau_i$ in $[0, \tau_{max}]$, generate the $\hat{y}_j$ and $\hat{y}_{k|j}$ collocation points, and learn the relation between input and output. So, we actually "learn" $H_k \approx \hat{H}_k$. See Section 3.3 for the ANN details.
2. Online stage: Partition time interval $[0, T]$, $t_i = i \cdot T/N,\ i = 0, \dots N$, with equidistant time step $\Delta t = t_{i+1} - t_i$. Given a sample $\hat{Y}_i$ at time $t_i$, compute $m$ collocation points at time $t_{i+1}$ using
       $$\hat{y}_j(t_{i+1})|\hat{Y}_i = \hat{H}_j(\hat{Y}_i, t_i, t_{i+1} - t_i, \boldsymbol{\theta}), j = 1, 2, \dots, m, \quad (18)$$
   and form a vector $\hat{\mathbf{y}}_{i+1} = (\hat{y}_1(t_{i+1})|\hat{Y}_i, \hat{y}_2(t_{i+1})|\hat{Y}_i, \dots, \hat{y}_m(t_{i+1})|\hat{Y}_i)$.
3. Compute the interpolation function $g_m(\cdot)$, or calculate the coefficients $\hat{\boldsymbol{\alpha}}_{i+1}$ (if necessary):
       $$A\hat{\boldsymbol{\alpha}}_{i+1} = \hat{\mathbf{y}}_{i+1}, \quad (19)$$
   where $A$ is the corresponding matrix (e.g., the Vandermonde matrix in a polynomial interpolation) and more details on the computation of the original collocation points can be found in Algorithm 1. We will compare monotonic spline, Chebyshev, and the barycentric formulation of Lagrange interpolation for this purpose. See Section 4.2 for a detailed discussion.
4. Sample from $X$ and obtain a sample in the next time point, $\hat{Y}_{i+1}$,
   by $\hat{Y}_{i+1}|\hat{Y}_i = g_m(\hat{X}_{i+1})$, or the coefficient form as follows,
       $$\hat{Y}_{i+1}|\hat{Y}_i = \sum_{j=0}^{m-1} \hat{\alpha}_{i+1,j}\hat{X}_{i+1}^j.$$
5. Return to Step 2 by $t_{i+1} \to t_i$, iterate until terminal time $T$.
6. Repeat this procedure for a number of Monte Carlo paths.
```

Remark 1 (Lagrange interpolation issue)**.** *In the case of classical Lagrange interpolation, Vandermonde matrix A should not get too large, as the matrix would then suffer from ill-conditioning. However, when employing orthogonal polynomials, this drawback is removed. More details can be found in Grzelak et al. (2019).*

When the approximation errors from the ANN and SCMC techniques are sufficiently small, the strong convergence properties of the 7L scheme can be estimated, as follows,

$$\mathbb{E}|\tilde{Y}(t_i) - Y(t_i)| < \epsilon(\Delta\tau) \ll K(\Delta t)^{\beta_s}, \quad (20)$$

where time step $\Delta\tau$ is used to define the ANN training data-set, and the actual time step Δt is used for ANN prediction, with $\Delta\tau \ll \Delta t$. Based on the trained 7L scheme, the strong error, $\epsilon(\Delta\tau)$, thus, does not grow with the actual time step Δt. In particular, let us assume $\Delta\tau = \Delta t/\kappa$, for example $\kappa = 100$, when employing the Euler–Maruyama scheme with time step $\Delta\tau$ during the ANN learning phase, we expect a strong convergence of $O(\sqrt{\Delta\tau})$, which then equals $O(\sqrt{\Delta t/\kappa})$, while the use of the Milstein scheme during training would result in $O(\Delta t/\kappa)$ accuracy. When $\kappa = 100$, the time step during the learning phase is 100 times smaller than Δt, which has a corresponding effect on the overall scheme's accuracy in terms of its strong, pathwise convergence. *The maximum value of the time step Δt in the 7L scheme can be set up to τ_{max} for a Markov process.* With a time step Δt, we solve the SDE in an iterative way until the actual terminal time T, which can be much larger than the training time horizon τ_{max}. An error analysis can be found in Section 5.2.

3.3. The Artificial Neural Network

The ANN to learn the conditional collocation points is detailed in this subsection. Neural networks can be utilized as powerful functions to approximate a nonlinear relationship. In fact, we will employ a rather basic fully-connected neural network configuration for our learning task.

A fully connected neural network, without skip connections, can be described as a composition function, i.e.,

$$\hat{H}(\acute{\mathbf{x}}|\tilde{\mathbf{\Theta}}) = h^{(L)}(\dots h^{(2)}(h^{(1)}(\acute{\mathbf{x}};\tilde{\boldsymbol{\theta}}_1);\tilde{\boldsymbol{\theta}}_2);\dots\tilde{\boldsymbol{\theta}}_{L_A}), \tag{21}$$

where $\acute{\mathbf{x}}$ represents the input variables, $\tilde{\mathbf{\Theta}}$ being the hidden parameters (i.e., weights and biases), L_A the number of hidden layers. We can expand the hidden parameters as,

$$\tilde{\mathbf{\Theta}} = (\tilde{\boldsymbol{\theta}}_1, \tilde{\boldsymbol{\theta}}_2, \dots, \tilde{\boldsymbol{\theta}}_L) = (\mathbf{w}_1, \mathbf{b}_1, \mathbf{w}_2, \mathbf{b}_2, \dots, \mathbf{w}_L, \mathbf{b}_{L_A}), \tag{22}$$

where $\mathbf{w}_\ell$ and $\mathbf{b}_\ell$ represent the weight matrix and the bias vector, respectively, in the ℓ-th hidden layer.

Each hidden-layer function, $h^{(\ell)}(\cdot), \ell = 1, 2, \dots, L_A$, takes input signals from the output of a previous layer, computes an inner product of weights and inputs, and adds a bias. It sends the resulting value in an activation function to generate the output.

Let $z_j^{(\ell)}$ denote the output of the j-th neuron in the ℓ-th layer. Then,

$$z_j^{(\ell)} = \varphi^{(\ell)}\left(\sum_i w_{i,j}^{(\ell)} z_i^{(\ell-1)} + b_j^{(\ell)}\right), \tag{23}$$

where $w_{i,j}^{(\ell)} \in \mathbf{w}_\ell$, $b_j^{(\ell)} \in \mathbf{b}_\ell$, and $\varphi^{(\ell)}$ is a nonlinear transfer function (i.e., activation function). With a specific configuration, including the architecture, the hidden parameters, activation functions, and other specific operations (e.g., drop out), the ANN in (21) becomes a deterministic, complicated, composite function.

Supervised machine learning (Goodfellow et al. 2016) is used here to determine the weights and biases, where the ANN should learn the mapping from a given input to a given output, so that for a new input, the corresponding output will be accurately approximated. Such ANN methodology consists of basically two phases. During the (time-consuming, but offline) training phase the ANN learns the mapping, with many in- and output samples, while in the testing phase, the trained model is used to very rapidly approximate new output values for other parameter sets, in the online stage.

In a supervised learning context, the loss function measures the distance between the target function and the function implied by the ANN. During the training phase, there are many known data samples available, which are represented by input-output pairs $(\acute{\mathbf{X}}, \acute{\mathbf{Y}})$. With a user-defined loss function $\mathrm{L}(\tilde{\mathbf{\Theta}})$, training neural networks is formulated as

$$\arg\min_{\tilde{\mathbf{\Theta}}} \mathrm{L}(\tilde{\mathbf{\Theta}}|(\acute{\mathbf{X}}, \acute{\mathbf{Y}})), \tag{24}$$

where the hidden parameters are estimated to approximate the function of interest in a certain norm. More specifically, in our case, the input, $\acute{\mathbf{x}}$, equals $\{\hat{Y}_i, t_i, t_{i+1} - t_i, \boldsymbol{\theta}\}$, and the output, $\acute{\mathbf{y}}$, represents the collocation points $\hat{\mathbf{y}}_{i+1}$, as in Equation (18). In the domain of interest $\acute{\Omega}$, we have a collection of data points $\{\acute{\mathbf{x}}_k\}$, $k = 1, \dots, M_D$, and their corresponding collocation points $\{\acute{\mathbf{y}}_k\}$, which form a vector of input-output pairs $(\acute{\mathbf{X}}, \acute{\mathbf{Y}}) = \{(\acute{\mathbf{x}}_k, \acute{\mathbf{y}}_k)\}_{k=1,\dots,M_D}$. For example, using the $L2$-norm, the discrete form of the loss function reads,

$$\mathrm{L}(\tilde{\mathbf{\Theta}}|(\acute{\mathbf{X}}, \acute{\mathbf{Y}})) = \sqrt{\frac{1}{M_D}\sum_{k=1}^{M_D}\left(\acute{\mathbf{y}}_k - \hat{H}(\acute{\mathbf{x}}_k|\hat{\mathbf{\Theta}})\right)^2}. \tag{25}$$

One of the popular approaches for training ANNs is to optimize the hidden parameters via backpropagation, for instance, using stochastic gradient descent (Goodfellow et al. 2016).

4. An Efficient Large Time Step Scheme: Compression–Decompression Variant

The 7L scheme employs the ANNs to generate the conditional collocation points for all samples of a previous time point, see Figure 1b. The extensive use of ANNs in the methodology has an impact on the method's computational complexity.

In order to speed up the data-driven 7L scheme procedure, we introduce a compression–decompression (CDC) variant, *in the online validation phase.* Please note that the offline learning phase is identical for both variants. The so-called 7L-CDC scheme, to be developed in this section, only uses the ANNs to determine the conditional collocation points for the optimal collocation points of a previous time point. All other samples will be computed by means of accurate interpolation. The computational complexity is reduced when the chosen interpolation is computationally cheaper than using ANNs.

By the compression–decompression procedure, Monte Carlo sample paths based on SDEs can be recovered from a 3D matrix. We then employ the 7L scheme procedure only to compute the entries of the encoded matrices at time point t_i (see Section 4.1), which leads to a reduction of the computational cost in many cases.

Next, we will explain the process of recovering the sample paths from a known matrix C using the decompression method.

4.1. CDC Variant

With a discretization $\{t_0, t_1, t_2, \dots, t_N\}$, we define a 3D matrix $\hat{C} = \{\hat{C}_0, \hat{C}_1, \dots, \hat{C}_{N-1}\}$, which consists of $N \times (M_s + 1) \times (M_c + 2)$ entries in total. Recall that M_s represents the number of collocation points and M_c the number of conditional collocation points, N the number of grid points over time. M_s and M_c may vary with time points t_i (in case of an adaptive scheme, for example), but we use constant values for M_s and M_c. For each time point t_i, we construct a 2D matrix $\hat{C}_i$,

$$\hat{C}_i = \begin{pmatrix} - & - & \hat{x}_1 & \hat{x}_2 & \dots & \hat{x}_{M_c} \\ x_1 & \hat{y}_1(t_i)|Y_0 & \hat{y}_1(t_{i+1})|\hat{y}_1(t_i) & \hat{y}_2(t_{i+1})|\hat{y}_1(t_i) & \dots & \hat{y}_{M_c}(t_{i+1})|\hat{y}_1(t_i) \\ x_2 & \hat{y}_2(t_i)|Y_0 & \hat{y}_1(t_{i+1})|\hat{y}_2(t_i) & \hat{y}_2(t_{i+1})|\hat{y}_2(t_i) & \dots & \hat{y}_{M_c}(t_{i+1})|\hat{y}_2(t_i) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{M_s} & \hat{y}_{M_s}(t_i)|Y_0 & \hat{y}_1(t_{i+1})|\hat{y}_{M_s}(t_i) & \hat{y}_2(t_{i+1})|\hat{y}_{M_s}(t_i) & \dots & \hat{y}_{M_c}(t_{i+1})|\hat{y}_{M_s}(t_i) \end{pmatrix}_{(M_s+1)\times(M_c+2)}, \quad (26)$$

with x_j, $j = 1, \dots, M_s$, the original SC points, $\hat{y}_j(t_i)|Y_0$ the marginal collocation points (depending on the initial value Y_0), and $\hat{x}_k$, $k = 1, \dots, M_c$, the k-th original conditional SC points, $\hat{y}_k(t_{i+1})|\hat{y}_j(t_i)$ the conditional SC points (depending on the marginal collocation points $\hat{y}_j(t_i)|Y_0$). We thus represent the conditional SC points, $\hat{y}_k(t_{i+1})|\hat{y}_j(t_i)$, by matrix elements $c_{i,j,k}$. The two empty cells in (26) are not addressed in the computation. Moreover, at the last time point, t_N, $\hat{C}_N$ is not needed.

Remark 2 (Time-dependent elements)**.** *As the original collocation points, x_i and $\hat{x}_k$, do not depend on time, we can remove the first row and the first column of matrix $\hat{C}_i$ to obtain a time-dependent version, $C = \{C_0, C_1, \dots, C_{N-1}\}$, with the following elements,*

$$C_i = \begin{pmatrix} \hat{y}_1(t_i)|Y_0 & \hat{y}_1(t_{i+1})|\hat{y}_1(t_i) & \hat{y}_2(t_{i+1})|\hat{y}_1(t_i) & \dots & \hat{y}_{M_c}(t_{i+1})|\hat{y}_1(t_i) \\ \hat{y}_2(t_i)|Y_0 & \hat{y}_1(t_{i+1})|\hat{y}_2(t_i) & \hat{y}_2(t_{i+1})|\hat{y}_2(t_i) & \dots & \hat{y}_{M_c}(t_{i+1})|\hat{y}_2(t_i) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \hat{y}_{M_s}(t_i)|Y_0 & \hat{y}_1(t_{i+1})|\hat{y}_{M_s}(t_i) & \hat{y}_2(t_{i+1})|\hat{y}_{M_s}(t_i) & \dots & \hat{y}_{M_c}(t_{i+1})|\hat{y}_{M_s}(t_i) \end{pmatrix}_{M_s\times(M_c+1)}. \quad (27)$$

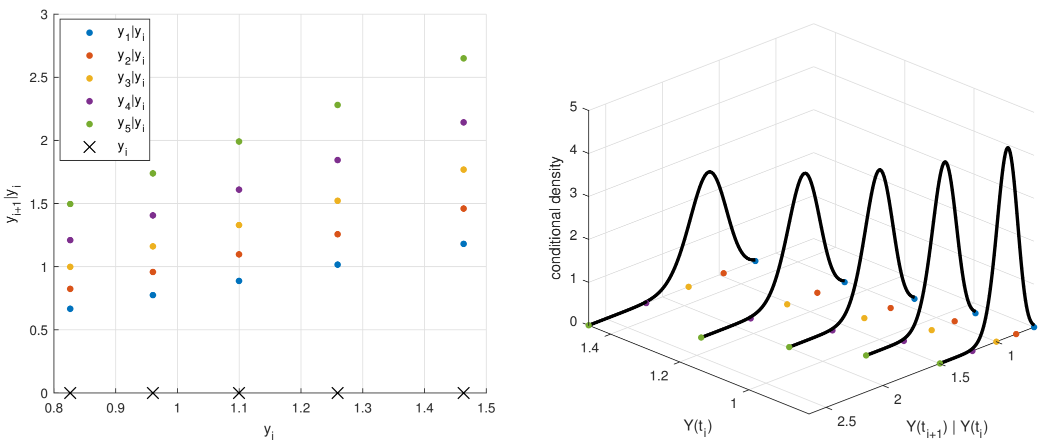
An entry in matrix C_i can be computed by the trained ANNs, as follows,

$$c_{i,j,k} := \hat{y}_k(t_{i+1})|\hat{y}_j(t_i) = \hat{H}_k^{(M_c)}\left(\hat{y}_j(t_i), t_i, t_{i+1} - t_i, \boldsymbol{\theta}\right), \quad i = 0, \dots, N-1, \quad (28)$$

using the marginal SC points,

$$\hat{y}_j(t_i)|Y_0 = \hat{H}_j^{(M_s)}\left(Y_0, t_0, t_i - t_0, \boldsymbol{\theta}\right), \;\; i = 0, \dots, N-1, \tag{29}$$

where $\hat{H}_j^{(\Lambda)}(\cdot), j = 1, \dots, m, \Lambda = \{M_s, M_c\}$, represents the ANN function which approximates the j-th collocation point when $\Lambda = M_s$, and the j-th conditional collocation point when $\Lambda = M_c$. When $M_c = M_s$, $\hat{H}_j^{(M_s)} = \hat{H}_j^{(M_c)}$. Figure 2 shows an example of the distribution of the conditional SC points when $M_c = 3$ and $M_s = 3$. When the matrices have been defined, all sample paths are compressed into a structured matrix. In other words, matrix $\hat{C}$ contains all of the information needed to perform the Monte Carlo simulation of the SDEs, apart from the interpolation technique.



(**a**) marginal and conditional SC points (**b**) conditional SC points and densities

Figure 2. Schematic illustration of matrix C, with five marginal SC points and five conditional SC points. The conditional SC points are dependent on the realization connected to the corresponding marginal SC point.

The resulting matrix C will be *decompressed* to generate Monte Carlo sample paths with the help of an interpolation. The process of decompression is straightforward given a matrix $\hat{C}$. In addition to the interpolation process $g_m(\cdot)$ in SCMC (see Equation (19)), an interpolation $\hat{g}_m(\cdot)$ is needed to compute conditional collocation points for previous realizations, based on the matrix $\hat{C}$.

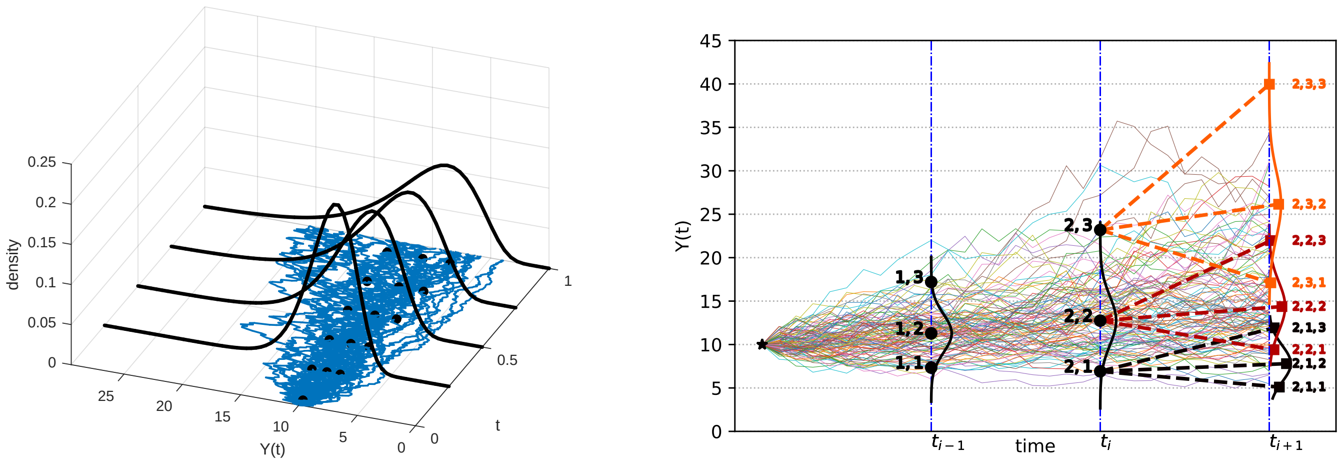
Suppose a vector of samples $\hat{\mathbf{Y}}_i$ at time t_i, and we wish to generate samples of $\hat{\mathbf{Y}}_{i+1}$. For a specific sample $\hat{Y}_i^*$, we need to calculate M_c conditional SC points. To obtain the k-th ($1 \leq k \leq M_c$) conditional SC point, we take marginal collocation points and their k-th conditional collocation points, using functions (28) and (29), to form M_s input-output pairs $\{(\hat{y}_1(t_i)|Y_0, \hat{y}_k(t_{i+1})|\hat{y}_1(t_i)), (\hat{y}_2(t_i)|Y_0, \hat{y}_k(t_{i+1})|\hat{y}_2(t_i)), \dots, \hat{y}_{M_s}(t_i)|Y_0, \hat{y}_k(t_{i+1})|\hat{y}_{M_s}(t_i)\}$. This combination gives us the interpolation function $\hat{g}_m$, through which we can obtain the k-th conditional SC point of $\hat{Y}_i^*$ at time point $i = 0, \dots, N-1$,

$$\hat{y}_k^*(t_{i+1})|\hat{Y}_i^* = \hat{g}_m(\hat{Y}_i^*), \quad k = 1, \dots, M_c. \tag{30}$$

As a result, for each sample $\hat{Y}_i^*$, we obtain M_c interpolation nodes, which form a set of pairs, $\left(\hat{x}_1, \hat{y}_1^*(t_{i+1})|\hat{Y}_i^*\right), \left(\hat{x}_2, \hat{y}_2^*(t_{i+1})|\hat{Y}_i^*\right)$, up to $\left(\hat{x}_k, \hat{y}_{M_c}^*(t_{i+1})|\hat{Y}_i^*\right)$, which are used to determine the interpolation function $g_m(\cdot)$ required by SCMC. Afterwards, to generate a new sample $\hat{Y}_{i+1}^*|\hat{Y}_i^*$, the mapping function g_m defines a conditional sample by taking in a random sample from X,

$$\hat{Y}_{i+1}^*|\hat{Y}_i^* = g_m(\hat{X}_{i+1}).$$

The choice of the appropriate number of (conditional) collocation points is a trade-off between the computational cost and the required accuracy. When the number of collocation points tends to infinity, the 7L-CDC scheme will resemble the 7L scheme from Section 3.1. A schematic picture is presented in Figure 3.



(**a**) paths for marginal SC points (**b**) sample paths by 7L-CDC

Figure 3. Schematic diagram of the 7L-CDC scheme at time t_i. (**a**): marginal SC points, corresponding to Equation (29). (**b**): sample paths generated by 7L-CDC. The triple {2,1,3}, in the picture, represents the third conditional SC point, dependent on the first marginal SC point at time point t_2. The above procedure is also applicable to other time points.

Remark 3 (Computation time)**.** *During the online phase of the method, the total computation time of the large time step schemes consists of essentially two parts, calculation of the conditional SC points, and generating random samples by interpolation (the second part). The difference between the 7L and 7L-CDC schemes is found in the computation of the conditional SC points, the generation of the samples is identical for both schemes.*

In this first part, for the 7L-CDC scheme, the work consists of setting up matrix C by the ANNs and computing the conditional SC points by the interpolation. In matrix C, there are $M_s \times M_c \times N$ elements that are computed by the ANNs, where N represents the number of time points, M_s the number of collocation points and M_c the number of conditional collocation points. Depending on the M_s collocation points, the interpolation is based on M_c conditional collocation points for each path. For the 7L scheme, $M \times M_c \times N$ elements, where M is the total number of paths, are computed by the ANNs. The time ratio between the 7L-CDC and 7L schemes is found to be,

$$\gamma = \frac{t_I M + t_A M_s}{t_A M} = \frac{t_I}{t_A} + \frac{M_s}{M}, \tag{31}$$

with t_A the computational time of the ANN (i.e., the function $\hat{H}(\cdot)$), t_I for the interpolation (i.e., the function $\hat{g}(\cdot)$ in (30)), which is a polynomial function of M_s. Given the fact that the number of sample paths is typically much larger than the number of SC points $M \gg M_s$,

$$\gamma \approx \frac{t_I}{t_A}.$$

When the employed interpolation is computationally cheaper than the ANNs, $\gamma < 1$, so that the 7L-CDC scheme needs fewer computations than the 7L scheme.

4.2. Interpolation Techniques

To define the function $g_m(x)$ in (13) or $\hat{g}(x)$ in (30), we will compare three different interpolation techniques.

A bijective mapping function is obtained by the *monotonic* piecewise cubic Hermite interpolating polynomial (PCHIP) (Fritsch and Carlson 1980). Assuming there are multiple data points, (x_k, y_k), using,

$$h_k := x_{k+1} - x_k,\ d_k := \frac{y_{k+1} - y_k}{x_{k+1} - x_k},$$

the derivatives f_k' at the points x_k are computed as a weighted average,

$$\frac{\hat{w}_1 + \hat{w}_2}{f_k'} = \frac{\hat{w}_1}{d_{k-1}} + \frac{\hat{w}_2}{d_k}, \quad \text{if } d_k \cdot d_{k-1} > 0,$$

where $\hat{w}_1 := 2h_k + h_{k-1}$ and $\hat{w}_2 := h_k + 2h_{k-1}$. At each data point, the first derivative is guaranteed to be continuous, and a cubic spline is used to interpolate between the data points. If $d_k \cdot d_{k-1} \leq 0$, then $f_k' = 0$, PCHIP requires more computations than a Lagrange interpolation, but it results in a monotonic function which is generally advantageous.

The convergence of the stochastic collocation method is not really dependent on the monotonicity of the mapping function, so an interpolation based on *Lagrange polynomials* is possible in practice. The barycentric version of Lagrange interpolation (Berrut and Trefethen 2004), our second interpolation technique, provides a rapid and stable interpolation scheme, which is applied when using Lagrange interpolation in our numerical experiments. With help of the basic Lagrange interpolation expressions, however, we can conveniently perform theoretical analysis.

The third technique is based on choosing the interpolation points carefully (e.g., as the Chebyshev zeros) to achieve a stable interpolation. The *Chebyshev interpolation* (Rivlin 1990) is of the form,

$$g_m(x) = \sum_{j=0}^{m-1} \alpha_j p_j(x) = \alpha_0 + \alpha_1 p_1(x) + \ldots + \alpha_{m-1} p_{m-1}(x), \tag{32}$$

where $p_{m-1}(x)$ are interpolation basis functions, here Chebyshev orthogonal polynomials, up to degree $m-1$. The Chebyshev nodes in the interval $[x_a, x_b]$ are computed as,

$$\tilde{x}_k = x_a + \frac{1 + \cos(\frac{\pi k}{m-1})}{2}(x_b - x_a), k = 0, 1, \ldots, m-1.$$

When the polynomial degree increases, the Chebyshev interpolation retains uniform convergence. In financial mathematics, Chebyshev interpolation has been successfully used, for example, to compute parametric option prices and implied volatility in (Gaß et al. 2018; Glau et al. 2019; Glau and Mahlstedt 2019). When the interpolation points are not Chebyshev nodes (e.g., Gauss quadrature points), the Chebyshev coefficients can be estimated by means of a least squares regression, which is also called the Chebyshev fit. In such case, the coefficients in (14) can be explicitly computed, in contrast to the barycentric Lagrange interpolation.

The selection of a suitable interpolation technique depends on various factors, for instance, speed, monotonicity, availability of coefficients. These three interpolation methods will be compared in the numerical section.

4.3. Pathwise Sensitivity

Often in computations with stochastic variables, we wish to determine the derivatives of the variables of interest, the so-called pathwise sensitivities. This is generally not a trivial exercise in a Monte Carlo setting, see, for example, the discussions in (Capriotti 2010; Giles and Glasserman 2006; Jain et al. 2019; Oosterlee and Grzelak 2019). With our new large time step schemes, we determine the pathwise sensitivities of the computed stochastic variables in a natural way, based on the available information in the (conditional) SC points and the interpolation. In this section, we derive the pathwise sensitivity of the state variable $Y(t)$ with respect to model parameters θ.

The first derivative with respect to parameter θ of the conditional distribution in Equation (13) reads,

$$\frac{\partial Y(t_{i+1})}{\partial \theta} = \frac{\partial g(X)}{\partial \theta} \approx \frac{\partial}{\partial \theta}\left(\sum_{j=1}^{m} \hat{y}_j(t) p_j(X)\right) = \frac{\partial}{\partial \theta}\left(\sum_{j=1}^{m} \hat{H}_j p_j(x)\right) = \sum_{j=1}^{m}\left(\frac{\mathrm{d}\hat{H}_j}{\mathrm{d}\theta} p_j(X)\right), \quad (33)$$

where $\frac{d\hat{H}_j}{d\theta}$ is the total derivative of function $\hat{H}_j(\hat{Y}_i, t_i, t_{i+1} - t_i, \boldsymbol{\theta})$ and $p_j(X)$ are basis functions, which do not depend on the model parameters. For the derivative $\frac{\mathrm{d}\hat{H}_j}{\mathrm{d}\theta}$ in (33) at time t_i, the expression of the ANN (21), given the specific activation function, is available. So, the function $\hat{H}$ is analytically differentiable. As a result, $\frac{\mathrm{d}\hat{H}_j}{\mathrm{d}\theta}$ can be easily computed, by means of automatic differentiation in the machine learning framework. Thus, we arrive at the sensitivity of a sample path with respect to model parameters, as follows,

$$\frac{\partial \hat{Y}_{i+1}}{\partial \theta} = \sum_{j=1}^{m} \frac{\mathrm{d}\hat{H}_j}{\mathrm{d}\theta} p_j(\hat{X}_{i+1}). \quad (34)$$

5. Numerical Experiments

In this section with numerical experiments we will give evidence of the high quality of our numerical SDE solver, by analyzing in detail its components. For this purpose, we mainly focus on the Geometric Brownian Motion SDE, which reads,

$$\mathrm{d}Y(t) = \mu Y(t)\mathrm{d}t + \sigma Y(t)\mathrm{d}W(t), \;\; Y_0 = Y(0), \quad 0 \le t \le T, \quad (35)$$

where the model parameters are the constant drift and volatility coefficients, i.e., $\boldsymbol{\theta} = \{\mu, \sigma\}$, and the initial value is given by Y_0 at time $t = 0$. For (35) a continuous-time analytic expression for the asset price at time t is available, i.e.,

$$Y(t) = Y_0 e^{(\mu - \frac{1}{2}\sigma^2)(t-t_0) + \sigma(W(t) - W(t_0))} \overset{d}{=} Y_0 e^{(\mu - \frac{1}{2}\sigma^2)(t-t_0) + \sigma\sqrt{t-t_0}X}, \quad (36)$$

where $X \sim \mathcal{N}(0,1)$, and $Y(t)$ is governed by the lognormal distribution. The derivative of the stock price with respect to volatility σ is available in closed form, and reads,

$$\frac{\partial Y(t)}{\partial \sigma} \overset{d}{=} Y(t)(-\sigma(t - t_0) + \sqrt{t - t_0}X). \quad (37)$$

This expression will be used as the reference value of the sensitivity obtained from the 7L discretization.

Furthermore, the Ornstein–Uhlenbeck process is explained and also analyzed, in Section 5.3.2. We will employ the large time step discretization, in which the conditional collocation points are computed by the trained ANN, and compare the results of the novel scheme with those obtained by the Milstein SDE discretization.

5.1. ANN Training Details

GBM and the OU process are Markov processes, so the conditional distribution at time t_{i+1} given information up to time t_i only depends on the information at time t_i. The ANN (15) will therefore be used for the conditional collocation stochastic points, with $\boldsymbol{\theta} = \{\mu, \sigma\}$, for GBM, and $\boldsymbol{\theta} = \{\overline{Y}, \sigma, \lambda\}$ for the OU process (as will be discussed in Section 5.3.2).

Regarding the size of the compression–decompression matrix, the more conditional collocation points, the better the accuracy of the 7L-CDC method. A 5×5 matrix size (i.e., five marginal and five conditional SC points) is preferred, taking into account the computing effort and the accuracy. In Grzelak et al. (2019) it has been discussed and shown that highly accurate approximations could already be obtained with a small number of collocation points.

As the first method component, we evaluate the quality of the ANN which defines the collocation points, for the GBM dynamics. For this purpose, M_L random points (i.e., sets of input parameters) are generated by using Latin hypercube sampling (LHS) in the domain of interest for the three parameters (Y_0, μ, σ), see Table 1. As the second step, for each point, a Monte Carlo method is employed to simulate the discretized SDE based on the tiny time step $\Delta\tau$. We use an Euler–Maruyama time discretization for this purpose, with N_τ the number of time points and the time horizon $\tau_{max} = N_\tau \cdot \Delta\tau$. At each time step, $j = 1, \ldots, N_\tau$, the conditional distribution function $F_{Y(t_j)|Y_0,\mu,\sigma}(\cdot)$ is computed, based on the many generated MC paths. This way, the resulting collocation points for the "big time step", $\Delta t = j \cdot \Delta\tau$, are also obtained, to form the required training dataset.

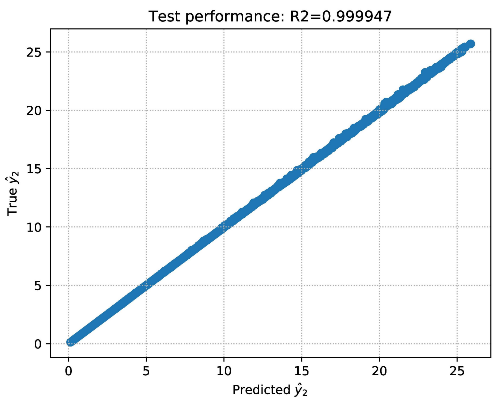
We set $\tau_{max} = 1.6$, $N_\tau = 160$, $M_L = 500$. The amount of training data used is given by $M_{train} = M_L \cdot N_\tau$ = 80,000 samples in total, which are divided into an ANN training (90%) and an ANN testing (10%) set.

Table 1. Training data, $\Delta\tau = 0.01$. Here is an example for training on five SC points.

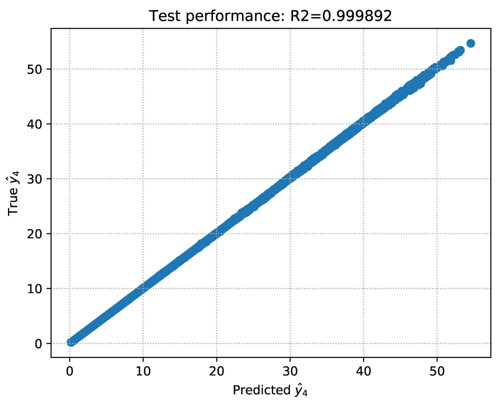
ANN	Parameters	Value Range	Method
input	volatility, σ	[0.05, 0.60]	LHS
	drift, μ	(0.0, 0.10]	LHS
	time, τ_{max}	(0.0, 1.60]	Equidistant
	realization, Y_0	[0.10, 15.0]	LHS
$\hat{H}_1(\cdot)$ output	point, $\hat{y}_1$	$(0.0, 25.65)$	SCMC
$\hat{H}_2(\cdot)$ output	point, $\hat{y}_2$	$(0.0, 25.98)$	SCMC
$\hat{H}_3(\cdot)$ output	point, $\hat{y}_3$	$(0.0, 27.84)$	SCMC
$\hat{H}_4(\cdot)$ output	point, $\hat{y}_4$	$(0.0, 54.67)$	SCMC
$\hat{H}_5(\cdot)$ output	point, $\hat{y}_5$	$(0.0, 154.35)$	SCMC

The ANN hyperparameters have an impact on the errors from the optimization related to training the ANN, as well as on the model performance. The approximation capacity does not only depend on the number of hidden parameters, but also on the network structure (i.e., on the width and depth of the network). In principle, deep neural networks have more powerful expressiveness than shallow neural networks. The fully connected neural network employed will be composed of one input layer, one output layer and four hidden layers. Each hidden layer consists of 50 neurons, with Softplus, i.e., $\varphi(x) = \ln(1 + e^x)$ as the activation function (Nwankpa et al. 2018). Before training the ANN, the hidden parameters are initialized via the Glorot technique (Glorot and Bengio 2010). Training goes in batches. At each iteration, a variant of SGD, the Adam (Kingma and Ba 2014) optimization algorithm, which implements adaptive learning rates, randomly selects a portion of the training samples according to the batch size, to calculate the gradient for updating the hidden parameters. In an epoch, all training samples have been processed by the optimizer. The mean squared error (MSE), which measures the distance between the ground-truth and the model values in supervised learning, is used to update the hidden parameters during training. The measure mean absolute error (MAE), i.e., MAE $= \frac{1}{M_{train}} \sum_j |y_j - \hat{y}_j|$, is also estimated, as the pathwise error of the 7L scheme is related to the maximum absolute difference in the approximated collocation points, $\hat{y}_j$, in Section 5.3, see the derivation in the next section.

The training process starts with a relatively large learning rate (i.e., 10^{-3}) to avoid getting stuck in local optima. After 1000 epochs, the learning rate is reduced to 10^{-4}, followed by training 500 more epochs, to achieve a steady convergence. Afterwards, the trained ANN is evaluated on the testing dataset, with the results presented in Figure 4 (for two of the collocation points) and Table 2. Clearly, the predicted values fit very well with the true values of the stochastic collocation points. This implies that the trained ANNs reach a highly satisfactory generalization, and generate accurate and robust approximation results for all five collocation points.



(**a**) SC point $\hat{y}_2$



(**b**) SC point $\hat{y}_4$

Figure 4. The goodness-of-fit on test dataset. Two scatter plots show the relation between the predicted values and the ground truth.

Table 2. The approximation performance on test data set.

SC Points	$\hat{y}_1$	$\hat{y}_2$	$\hat{y}_3$	$\hat{y}_4$	$\hat{y}_5$
R^2	0.999891	0.999947	0.999980	0.999892	0.999963
MAE	0.026	0.027	0.021	0.071	0.066

5.2. Numerical Error Analysis, the Lagrangian Case

There are essentially two approximation errors in the 7L scheme, a neural network approximation error when generating the collocation points, and an SCMC error when representing the conditional distribution function.

Considering d inputs, the neural network may approximate any function $\zeta_{d,n}$, from the function space $C^{n-1}([0,1]^d)$, where the derivatives up to order $n-1$ are Lipschitz continuous Yarotsky (2017). The input and output variables can be normalized to the unit interval $[0,1]$. With a fixed network architecture during training, the approximation error can be assessed, as follows.

Theorem 1. *From Yarotsky (2017), given any $\hat{\epsilon} \in (0,1)$, there exists a neural network which is capable of approximating any function $\zeta_{d,n}$ with error $\hat{\epsilon}$, based on the following configuration:*

- *at least piece-wise activation functions,*
- *at least $\tilde{c}(\ln(1/\hat{\epsilon})+1)$ hidden layers and $\tilde{c}\hat{\epsilon}^{-d/n}(\ln(1/\hat{\epsilon})+1)$ weights and computation units, where $\tilde{c} := \tilde{c}(d,n)$ depends on the parameters d and n.*

When the architecture is dynamic, the error bound can be further reduced, as shown in Montanelli and Du (2019) and Yarotsky (2017). One of the assumptions is that the ANNs are sufficiently trained, so that the optimization error is negligible.

The error from the SCMC methodology was derived in Grzelak et al. (2019). The optimal collocation points, $x_i, i = 1,\dots,m$, correspond to the zeros of an orthogonal polynomial. In the case of Lagrange interpolation, when the collocation method can be connected to Gauss quadrature, we have

$$\int_{\mathbb{R}} \Psi(x) f_X(x)\mathrm{d}x = \sum_{i=1}^{m} \Psi(x_i)\omega_i + \epsilon_m = \epsilon_m, \tag{38}$$

with $\Psi(x) = (g(x) - g_m(x))^2$, the difference between the target and the SC approximated function, $f_X(x)$ the weight function, and ω_i the quadrature weights. When the Gauss–Hermite quadrature is used with m collocation points. the approximation error of the CDF can be estimated as,

$$\epsilon_m = \frac{m!\sqrt{\pi}}{2^m}\frac{\Psi^{(2m)}(\xi_1)}{(2m)!}, \tag{39}$$

where $\xi_1 \in (-\infty, \infty)$ and the distance function

$$\Psi(x) = (g(x) - g_m(x))^2 \approx \left(\frac{1}{m!}\frac{\partial^m g(x)}{\partial x^m}\bigg|_{x=\xi_2}\prod_{k=1}^{m}(x - x_k)\right)^2,$$

with $\xi_2 \in [x_1, x_{m-1}]$. In other words, the error of approximating the target CDF converges exponentially to zero when the number of corresponding collocation points increases.

At each time point t_i, the process $Y(t_i)$ is approximated using the collocation method, by a polynomial $g_m(X)$, i.e., in the case of classical Lagrange interpolation, using $\ell_j(\bar{x}) = p_j(\bar{x})$,

$$Y(t_i) \approx \tilde{Y}(t_i) = g_m(X) = \sum_{j=1}^{m} y_j(t_i)\ell_j(X), \;\; \ell_j(\bar{x}) = \prod_{k=1}^{m}\frac{X - x_k}{x_j - x_k}, \tag{40}$$

where the collocation points $y_j(t_i) = F^{-1}_{Y(t_i)}(F_X(x_j))$. Because of the use of an ANN, the collocation points are not exact, but they are approximated with $y_j(t_i) - \hat{y}_j(t_i) = \epsilon^A_j$, where $\hat{y}_j(t_i)$ represents the ANN approximated value. The error associated with ϵ^A_j can be estimated as in (Montanelli and Du 2019). The impact of ϵ^A_j on the obtained output distribution needs to be assessed. Let $\tilde{g}_m$ denote the approximate function based on the predicted ANN collocation points $\hat{y}_j(t_i)$, and x a random sample from the standard normal distribution $\mathcal{N}(0,1)$. The approximation error, in the strong sense, is given by

$$\begin{aligned} \mathbb{E}[|g_m(x) - \tilde{g}_m(x)|] &= \mathbb{E}\Big|\sum_{j=1}^{m} y_j(t_i)\ell_j(x) - \sum_{j=1}^{m}\hat{y}_j(t_i)\ell_j(x)\Big| \\ &= \int_{\mathbb{R}}\Big|\sum_{j=1}^{m} y_j(t_i)\ell_j(x) - \sum_{j=1}^{m}\hat{y}_j(t_i)\ell_j(x)\Big| f_X(x)\mathrm{d}x \\ &= \int_{\mathbb{R}}\Big|\sum_{j=1}^{m}\epsilon^A_j\ell_j(x)\Big| f_X(x)\mathrm{d}x. \end{aligned} \tag{41}$$

Note that the $\ell_j(x)$ interpolation functions are identical as they depend solely on the x values. We arrive at the following error related to the ANNs,

$$\begin{aligned} \int_{\mathbb{R}}\Big|\sum_{j=1}^{m}\epsilon^A_j\ell_j(x)\Big| f_X(x)\mathrm{d}x &\leq \int_{\mathbb{R}}\sum_{j=1}^{m}\max\{|\epsilon^A_1|,\ldots,|\epsilon^A_m|\}\ell_j(x)f_X(x)\mathrm{d}x \\ &= \int_{\mathbb{R}}\max\{|\epsilon^A_1|,\ldots,|\epsilon^A_m|\}f_X(x)\mathrm{d}x \\ &= \max\{|\epsilon^A_1|,\ldots,|\epsilon^A_m|\}. \end{aligned} \tag{42}$$

Considering the error introduced by SCMC in (39), the total pathwise error reads

$$\begin{aligned} \mathbb{E}[|g(x) - \tilde{g}_m(x)|] &\leq \mathbb{E}[|g(x) - g_m(x)|] + \mathbb{E}[|g_m(x) - \tilde{g}_m(x)|] \\ &\leq \sqrt{|\epsilon_m|} + \max\{|\epsilon^A_1|,\ldots,|\epsilon^A_m|\}. \end{aligned} \tag{43}$$

In other words, the expected pathwise error can be bounded by the approximation CDF error $\sqrt{|\epsilon_m|}$ plus the largest difference in the ANN approximated collocation points.

Kolmogorov–Smirnov Test

The Kolmogorov–Smirnov test, calculating the supremum of a set of distances, is used to measure the nonparametric distance between two empirical cumulative distribution functions. We perform the two-sample Kolmogorov–Smirnov test, as follows,

$$KS = \sup_z |F_Y(z) - \hat{F}_Y(z)|,$$

where $\hat{F}_Y(\cdot)$ and $F_Y(\cdot)$ are two empirical cumulative distribution functions, one from the 7L-CDC solution and the other one from the reference distribution. We take the analytic solution of the GBM as the reference distribution.

Remark 4 (Time horizon for 7L-CDC)**.** *The information in Table 1 is used to train the mapping function between a realization (including marginal SC points) and its conditional SC points, via Equation (28). For the marginal SC points in Equation (29), however, we need training data up to terminal time T. So, we generate a second dataset in which the time reaches τ_{max} (the terminal time of interest) and the upper value for Y_0 equals 5. These two datasets are merged into one set in order to train the ANNs for the 7L-CDC methodology.*

Figure 5 shows the Kolmogorov–Smirnov test at different time points based on 10,000 samples. We focus on the CDC methodology here, and compare the accuracy with the different interpolation methods in the figure. Clearly, the KS statistic and also the corresponding p-values for the 7L-CDC schemes are much better than those of the Milstein scheme in Figure 5. This is an indication that the CDFs that originate from the 7L-CDC schemes resemble the target CDF much better, with high confidence. In addition, unlike the Milstein scheme, the 7L-CDC schemes exhibit an almost constant difference between the approximated and target CDFs with increasing time.

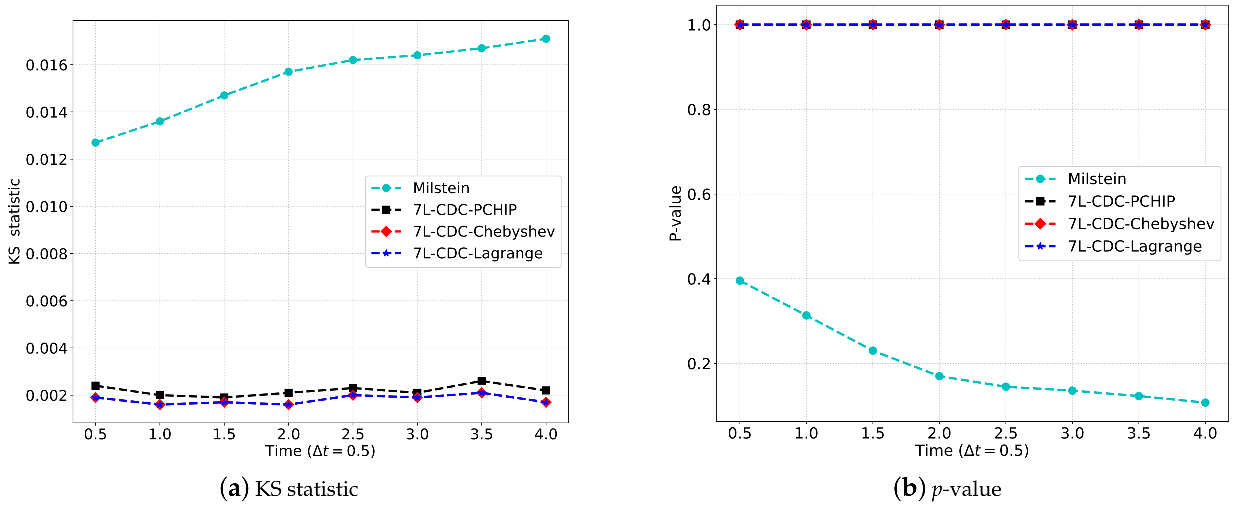


Figure 5. The Kolmogorov–Smirnov test: $\Delta t = 0.5, \mu = 0.1, \sigma = 0.3, Y_0 = 1.0$, with 10,000 samples. When we have a small KS statistic or a large p-value, the hypothesis that the distributions of the two sets of random samples are the same can not be rejected.

We will also analyze the costs of the different interpolation methods within 7L-CDC. The two steps which require interpolation are the computation of the conditional collocation points and the generation of conditional samples. The computational speed of the 7L-CDC scheme depends on the employed interpolation method, see Table 3. In general, to generate a solution with the same strong order in the numerical error, the Milstein scheme will require more computation time, here about 27 s, while the 7L scheme needs 13 s and

7L-CDC (barycentric version) 5 s when $\Delta t = 1.0$. The larger the time step, the more computation time will be saved.

Table 3. The CPU running time (s) to reach the same accuracy (CPU: E3-1240, 3.40 GHz): simulating 10,000 sample paths until terminal time $T = 4.0$, based on 5×5 marginal/conditional SC points. Here, for the 7L scheme, PCHIP is used as the interpolant $g_m(\cdot)$ in Step 3 of Algorithm 1.

Method/Time (s)	**$\Delta t = 1.0$**			**$\Delta t = 2.0$**		
	Create *C*	**Decom. *C***	**Total**	**Create *C***	**Decom. *C***	**Total**
7L-CDC Barycentric	0.054	4.93	4.98	0.027	2.48	2.51
7L-CDC Chebyshev	0.054	9.78	9.83	0.027	4.93	4.96
7L-CDC PCHIP	0.054	11.39	11.44	0.027	5.73	5.76
7L scheme	-	-	12.80	-	-	6.39
Milstein	-	-	27.01	-	-	27.70

Note that, in order to achieve a similar accuracy in the strong sense, the Euler–Maruyama scheme requires a much finer time grid, by a factor of $\kappa = \Delta t/\Delta\tau$, than the 7L scheme. When κ is sufficiently large, the 7L-CDC scheme outperforms the Euler–Maruyama scheme, in terms of both accuracy and speed. For example, in Table 3, $\kappa = 100$ when $\Delta t = 1.0$, and $\kappa = 200$ when $\Delta t = 2.0$. In other words, the "online version" of the Euler–Maruyama discretization is computationally slower than the online phase of the 7L scheme to achieve the same accuracy. Additionally, computational time of the 7L scheme can be further reduced by parallelization, for example, using GPUs .

5.3. Pathwise Error Convergence

In this section, we compare the pathwise errors of our proposed novel discretization with those of the classical discretization schemes.

5.3.1. GBM Process

We analyze here the strong convergence properties of the new methodology for the GBM process. For GBM, the exact path is given by the expression (36). The random number, which is drawn from $X \sim N(0,1)$, is the same for the exact solution (36), the novel schemes (14) and the Milstein scheme (3). The pathwise differences between the numerical schemes and the exact simulation are plotted in Figure 6. When $\Delta t = 0.5$, the 7L-CDC scheme presents superior paths as compared to the Milstein scheme, in terms of its pathwise error comparing to the exact path.

As shown in Figure 7, the 7L-CDC scheme gives rise to flat, almost constant, strong and weak error convergence curves for many different Δt-values, suggesting a small, constant convergence error even with large time steps Δt. The Milstein scheme has the strong order of convergence $O(\Delta t)$, so that a larger time step gives rise to a larger error. When the time step becomes small, more time points are needed to reach a time T, and then the resulting recursive error of the 7L-CDC scheme increases.

The number of conditional collocation points, by which the conditional distribution at a next time point is mostly determined, has a significant contribution to the convergence order of the 7L-CDC scheme. As mentioned, we found empirically that five conditional collocation points are preferable in terms of computing effort versus accuracy. CDC matrix C is then of size $N \times 5 \times 5$; that is, at each time point, there are five collocation points and each of these has five conditional collocation points.

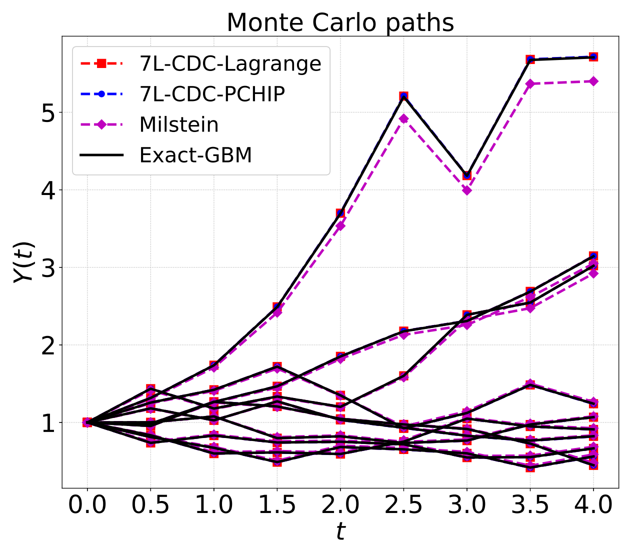
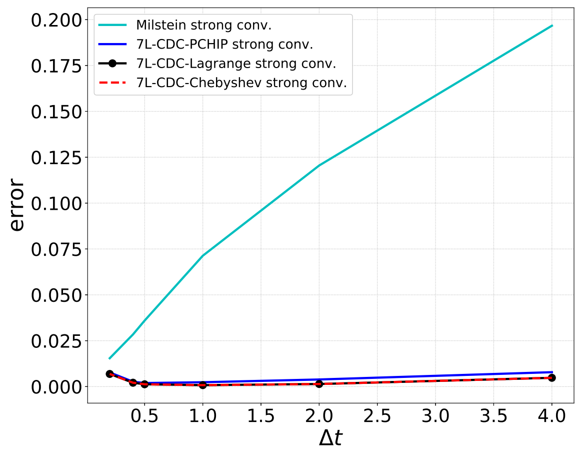
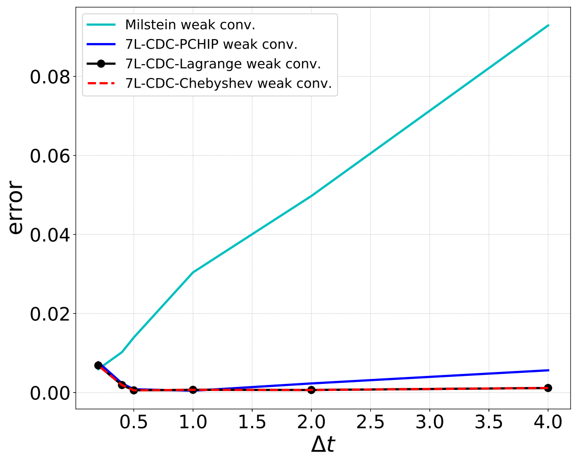


Figure 6. Paths generated by 7L-CDC: time step $\Delta t = 0.5$, GBM with $\sigma = 0.3$, $r = 0.1$, $Y_0 = 1.0$. The paths are with Chebyshev interpolation, which are not plotted, are identical to ones from Lagrange in this case.



(**a**) strong convergence



(**b**) weak convergence

Figure 7. The strong error is estimated as $\frac{1}{M}\sum|\check{Y}_k(T) - \hat{Y}_k(T)|$, see Equation (4) and the weak error by $\frac{1}{M}(\sum \check{Y}_k(T) - \sum \hat{Y}_k(T))$, see (Oosterlee and Grzelak 2019, p. 261) for details on the computation of the convergence rate. There are $M = 1000$ sample paths in total.

5.3.2. Ornstein–Uhlenbeck Process

In the case of Markov processes, any SDE which can be solved by the Euler–Maruyama discretization can be solved by our ANN methodology, with improved strong convergence properties. We also wish to confirm the strong convergence properties for another stochastic process in this section.

The mean reverting Ornstein–Uhlenbeck (OU) process (Uhlenbeck and Ornstein 1930) is defined as,

$$dY(t) = -\lambda(Y(t) - \overline{Y})dt + \sigma dW(t), \quad 0 \le t \le T, \tag{44}$$

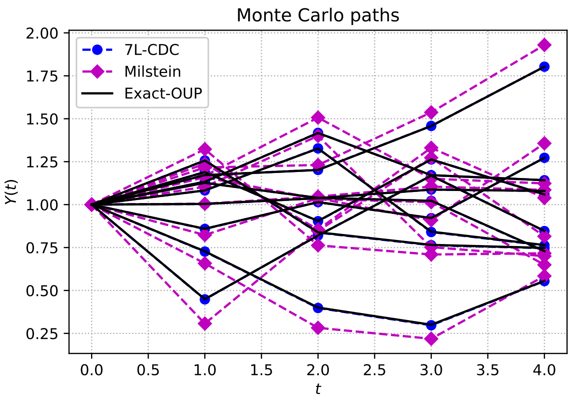
with $\overline{Y}$ the long term mean of $Y(t)$, λ the speed of mean reversion, and σ the volatility. The initial value is Y_0, and the model parameters are $\boldsymbol{\theta} := \{\overline{Y}, \sigma, \lambda\}$. Its analytical solution is given by,

$$Y(t) \stackrel{d}{=} Y_0 e^{-\lambda t} + \overline{Y}(1 - e^{-\lambda t}) + \sigma\sqrt{\frac{1 - e^{-2\lambda t}}{2\lambda}} X, \tag{45}$$

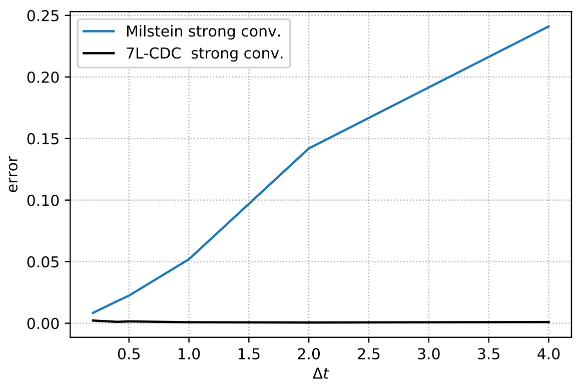
with $t_0 = 0$, $X \sim \mathcal{N}(0, 1)$. Equation (45) is used to compute the reference value to the pathwise error and the strong convergence.

We employ the same data-driven procedure as for GBM to discretize and solve the OU process. In the training phase, the Euler–Maruyama scheme (2) is used to discretize the OU dynamics and generate the dataset. Note that the Milstein and Euler schemes are identical in the case of the OU process. As the OU process is a Markov process, we again can vary Y_0 to find the relation between the conditional SC points and the marginal SC points (i.e., as in Equation (28)). Similar to Table 1, we employ five SC points to learn within the ANN, with $\Delta\tau = 0.01$, $\tau_{max} = 4.1$, $N_\tau = 500$, $M_L = 410$, see Section 5.1 for the details of the training process.

After the training, the obtained ANNs will be applied to solve the OU process with specific parameters and details of our interest. We provide an example in Figure 8, which confirms that the sample paths generated by 7L-CDC are as accurate as the exact solution, and the error, in the sense of strong convergence, stays close to zero even with a large time step.



(**a**) path-wise error ($\Delta t = 1.0$)



(**b**) strong convergence

Figure 8. Paths and strong convergence for the OU process, using $\lambda = 0.5$, $\overline{Y} = 1.0$, $\sigma = 0.3$, $Y_0 = 1.0$. The sample paths with barycentric, Chebyshev and PCHIP interpolation overlap for the 7L-CDC scheme. There are five marginal and five conditional SC points at each time point.

5.4. Applications in Finance

The possibility to take large time steps and still get accurate SDE solutions is certainly interesting in computational finance, as there are several financial products that are updated on a daily basis (think of an over-night interest rate), whereas monitoring of financial contracts and risk management monitoring is typically only done on a weekly, monthly of even yearly basis. In such situations, our novel scheme will be useful. Research into large time step simulations is state-of-the-art in computational finance, see the exact (and almost exact) Monte Carlo simulation papers, e.g., Broadie and Kaya (2006); Leitao et al. (2017) for the SABR and Heston stochastic volatility asset dynamics, respectively.

5.4.1. The Asian Option

Moreover, the strong convergence property of an SDE discretization is important in many cases. When valuing so-called path-dependent options, for example, improved strong convergence enhances the convergence of a Monte Carlo simulation. Options are governed by their pay-off function (i.e., the option value at the final time of the contract,

$t = T$). Here we consider a path-dependent exotic option, the so-called European-style Asian option, which has a payoff that is based on a time-averaged underlying stock price. For example, the pay-off of a fixed strike Asian option is given by

$$V_A(T) = \max\left(A(T) - \tilde{K}, 0\right),$$

where T is the option contract's expiry time, and $\tilde{K}$ is the predetermined strike price. Here, $A(T)$ denotes the discrete arithmetic average of the stock prices over N_b monitoring dates $\{t_1, \ldots, t_k\} \in [0, T]$,

$$A(T) = \frac{1}{N_b} \sum_{k=1}^{N_b} \hat{Y}(t_k),$$

where $\hat{Y}(t_k)$ is the observed stock price at time t_k, $1 \le t_k \le T$. Averaging thus takes place in the time-wise direction, and we consider pricing financial options based on the discrete arithmetic average of a number of stock prices.

We assume here that the underlying stock price follows Geometric Brownian motion, as in Equation (35), under the risk-neutral measure, meaning $\mu \equiv r$, where r is the risk-free interest rate. There is a cash account $M(t)$, governed by $\mathrm{d}M(t) = rM(t)\mathrm{d}t$. The value of European-style Asian option is then given by

$$V_A(t) = \mathrm{e}^{-r(T-t)} \mathbb{E}^{\mathbb{Q}} \left[\max(A(T) - \tilde{K}, 0) \middle| \mathcal{F}(t) \right]. \tag{46}$$

Because the pay-off is clearly a path-dependent quantity for such options, it is expected that an improved strong convergence, obtained with the variant 7L-CDC, will result in superior convergence, as compared to classical numerical discretization schemes.

The relative error is presented, which is defined as

$$\epsilon_{rel} = \left| \frac{V_A^{ref}(t_0) - V_A(t_0)}{V_A^{ref}(t_0)} \right|,$$

where $V_A^{ref}(t_0)$ is based on the exact GBM Monte Carlo simulation. As shown in Table 4, the 7L-CDC scheme gives highly accurate Asian option prices, compared to the Milstein scheme. As the accuracy of Asian option prices depends directly on the accuracy of the realized paths, an increasing number of monitoring dates will give rise to higher accuracy by 7L-CDC.

Table 4. Pricing Asian European-style option with a fixed strike price, using $Y_0 = 1.0$, $\tilde{K} = Y_0$, $r = 0.1$, $T = \Delta t \times N_b$, the number of sample paths M = 100,000.

	Method	**$\Delta t = 1.0$, N_b = 4**	**$\Delta t = 0.5$, N_b = 8**
σ = 0.30	Analytic MC	0.24886257 (0.00%)	0.22403982 (0.00%)
	Milstein MC	0.23077000 (7.27%)	0.21558276 (3.77%)
	7L-CDC	0.24871446 (0.06%)	0.22404571 (0.00%)
σ = 0.40	Analytic MC	0.28515109 (0.00%)	0.25723594 (0.00%)
	Milstein MC	0.26394277 (7.44%)	0.24717425 (3.91%)
	7L-CDC	0.28482371 (0.11%)	0.25647592 (0.30%)

Next, we focus on the Asian option's sensitivity. The sensitivity of the option price with respect to volatility σ is called Vega, which can be computed in a pathwise fashion (see Chapter 7 in Glasserman 2004), as follows,

$$\frac{\partial V}{\partial \sigma} = e^{-rT} \mathbb{E}^{\mathbb{Q}} \left[\sum_{i=1}^{N} \frac{\partial V(T, Y(t_i); \sigma)}{\partial Y(t_i)} \frac{\partial Y(t_i)}{\partial \sigma} | Y_0 \right], \tag{47}$$

where N represents the number of grid points over time. The chain rule is employed to derive the sensitivity. First of all, we compute the gradient of the payoff function with respect to the underlying stock price, by

$$\frac{\partial V(T, Y(t_i))}{\partial Y(t_i)} = \frac{1}{N}\mathbb{1}_{A(T)>\tilde{K}}. \tag{48}$$

Then, the derivative of the stock price at time t_i with respect to the model parameter, $\frac{\partial Y(t_i)}{\partial \sigma}$, can be found with the trained ANNs, as given by Equation (33). Vega can be estimated by,

$$\frac{\partial V}{\partial \sigma} \approx e^{-rT}\frac{1}{N}\mathbb{E}^{\mathbb{Q}}\left[\sum_{i=1}^{N}\left(\mathbb{1}_{A(T)>\tilde{K}}\sum_{j=0}^{m-1}\frac{\mathrm{d}\hat{H}_j}{\mathrm{d}\sigma}p_j(X)\right)|Y_0\right]. \tag{49}$$

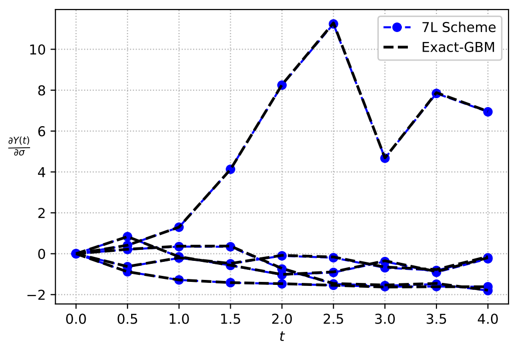
When there are M sample paths, we have,

$$\frac{\partial V}{\partial \sigma} \approx e^{-rT}\frac{1}{M}\frac{1}{N}\left[\sum_{k=1}^{M}\sum_{i=1}^{N}\left(\mathbb{1}_{A(T)>\tilde{K}}\sum_{j=0}^{m-1}\frac{\mathrm{d}\hat{H}_j}{\mathrm{d}\sigma}p_j(\hat{X}_{k,i+1})\right)|Y_0\right]. \tag{50}$$

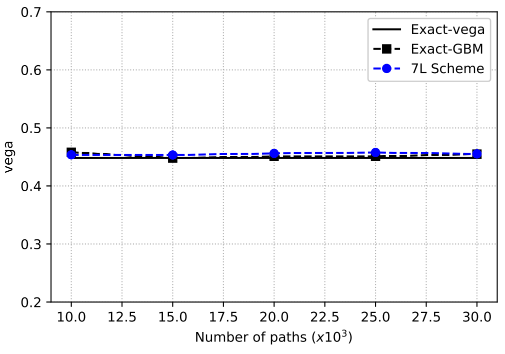
As shown in Table 1, there are four input arguments in the function $\hat{H}_j(\hat{Y}_i, t_{i+1} - t_i, r, \sigma)$ (note that the drift term equals the interest rate, i.e., $\mu = r$, in the risk-neutral world). Since the previous realization $\hat{Y}_i$ is a function of the model parameters (here r and σ), at time t_{i+1}, the total derivative of $\hat{H}_j$ with respect to the volatility in Equation (34) becomes

$$\frac{\mathrm{d}\hat{H}_j}{\mathrm{d}\sigma} = \frac{\partial \hat{H}_j}{\partial \sigma} + \frac{\partial \hat{H}_j}{\partial \hat{Y}_i}\frac{\partial \hat{Y}_i}{\partial \sigma}, \tag{51}$$

where $\frac{\partial \hat{Y}_i}{\partial \sigma}$ is known at the previous time point. Like simulating the Monte Carlo paths, the calculation of this derivative is done iteratively. Figure 9a compares the pathwise sensitivities obtained via Equations (37) and (51). Clearly, the pathwise derivative by the 7L scheme is very similar to the analytical solution. Figure 9b confirms that the ANN methodology computes a highly accurate Asian option Vega by means of the above pathwise sensitivity. Summarizing, the sensitivity with respect to model parameters can highly accurately be obtained from the trained ANNs. As the 7L-CDC scheme is composed of marginal and conditional collocation points, the above procedure of computing the pathwise sensitivity is also applicable to the variant 7L-CDC, by using the chain rule.



(**a**) path-wise sensitivity



(**b**) path-wise vega

Figure 9. Path-wise estimator of Vega: Exact Vega is calculated by means of the central finite difference. The parameters are $Y_0 = 1.0$, $r = 0.05$, $\tilde{K} = Y_0$, $\sigma = 0.3$, $\Delta t = 1.0$, $N_b = 4$, $T = \Delta t \times N_b = 4.0$.

5.4.2. Bermudan Option Valuation

When dealing with so-called Bermudan options, the option contract holder has the right (but not the obligation) to exercise the option contract at a finite number of pre-specified dates up to final time T. At an exercise date, when the holder decides to exercise the Bermudan option, she immediately obtains the current payoff value of the contract. Alternatively, she may also wait until the next exercise opportunity. The Bermudan option can be exercised at the following set of exercise dates, $\{t_0, t_1, \ldots, t_{N_b}\}$, with a constant time difference, $\Delta t = t_i - t_{i-1}$, for any $0 < i \leq N_b$.

In this experiment, we compare the performance of the new 7L-CDC discretization scheme with a classical scheme. Valuation of the Bermudan option will take place by means of the well-known Longstaff–Schwartz Monte Carlo (LSMC) method (Longstaff and Schwartz 2015), a least squares Monte Carlo method. The Longstaff–Schwartz algorithm is presented, for convenience, in the Appendix A.

The difference between a large time step simulation and a classical simulation, like the Milstein scheme, is that a classical scheme requires additional time steps to be taken between the early-exercise dates of the Bermudan option, while with the 7L-CDC scheme, we can perform one-step Monte Carlo simulation without any intermediate grid points between adjacent early-exercise dates.

We also assume here that the underlying stock price follows Geometric Brownian motion, as in Equation (35), under the risk-neutral measure, with $\mu \equiv r$. A Bermudan put option, with risk-free interest rate $r = 0.1$, pay-off function $V(t_j) = \max(\tilde{K} - Y(t_j), 0)$ with strike price $\tilde{K} = 1.1$ and initial stock price $Y_0 = 1.0$, is priced based on $M = 100{,}000$ Monte Carlo paths. The matrix size within the 7L-CDC scheme is set to $N_b \times 5 \times 5$. The terminal time is $T = \Delta t \times M_B$ with a constant time step Δt. The random seed is chosen to be zero when drawing random numbers. We compare the relative errors $|\frac{V^{ref}(t_0) - V(t_0)}{V^{ref}(t_0)}|$, where $V^{ref}(t_0)$ is computed with the help of a Monte Carlo method based on the exact simulation of GBM (36).

As shown in Table 5, the option prices based on the 7L-CDC Monte Carlo simulation are highly satisfactory, and the related error does not increase with larger time steps Δt. In contrast, a larger time step gives rise to significant pricing errors, in the case of the Milstein discretization.

Table 5. Bermudan put option prices based on large time step Monte Carlo simulations.

	Method	$\mathbf{\Delta t = 1.0, N_b = 4}$	$\mathbf{\Delta t = 0.5, N_b = 4}$	$\mathbf{\Delta t = 0.5, N_b = 8}$
	Analytic MC	0.15213858(0.00%)	0.14620214(0.00%)	0.16161876(0.00%)
$\sigma = 0.30$	Milstein MC	0.13872771(8.81%)	0.14065252(3.80%)	0.15429369(4.53%)
	7L-CDC	0.15234901(0.14%)	0.14648443(0.19%)	0.16196264(0.21%)
	Analytic MC	0.21459038(0.00%)	0.19552454(0.00%)	0.22340304(0.00%)
$\sigma = 0.40$	Milstein MC	0.19598488(8.67%)	0.18790933(3.89%)	0.21297732(4.67%)
	7L-CDC	0.21474619(0.07%)	0.19590733(0.20%)	0.22389360(0.22%)

Remark 5. *In principle, a sample value $\hat{Y}_i$ can be any rational number. So, a path value may reach a larger stock price than the prescribed upper bound in Table 1. The stock prices outside the training interval are called* outliers. *Outliers did not appear in the experiments of Table 5. As an alternative method to avoid the appearance of outliers, one may scale the asset price, to remove the dependence on the initial value. For example, GBM can be scaled by $\bar{Y}(t) = \frac{Y(t)}{Y(t_0)e^{rt}}$. Using Itô's lemma, we have a drift-less process, $d\bar{Y}(t) = \bar{Y}(t)\sigma dW$, where the initial value $\bar{Y}_0 = 1.0$. The following formula returns the original variable, $Y(t) = \bar{Y}(t)Y(t_0)e^{rt}$. In such case, scaling guarantees a fixed initial value, for example, $Y_0 = 1.0$.*

6. Conclusions and Outlook

We developed a data-driven numerical solver for stochastic differential equations, by which large time step simulations can be carried out accurately in the sense of strong convergence. With a combination of artificial neural networks and the stochastic collocation Monte Carlo method, a small number of stochastic collocation points are learned by the ANN to approximate a nonlinear function which can be used to compute the unknown collocation points. Theoretical analysis indicates that the numerical error is controllable and does not increase when the simulation time step increases.

There are several advantages to the proposed approach. The powerful expressive ability of neural networks enables the ANNs to accurately approximate stochastic collocation points. The compression–decompression method reduces the computational costs, so that the numerical method can be applied in practice. In finance, the proposed big time step methodology will be highly beneficial for the generation for path-dependent financial option contracts or in risk management applications.

As an outlook, it will be relevant to extend the introduced methodology to solving higher-dimensional or more involved SDE dynamics. We will define multi-dimensional stochastic collocation points (i.e., by means of a tensor) for a multi-dimensional system of SDEs, and choose a Convolutional Neural Network (LeCun et al. 2015) to efficiently process these collocation points. Of course, this may not trivially generalize to truly high-dimensional systems, but approximation of moderate dimensionality should be possible The computational speed can be further improved by parallel computation, for example, on GPUs. Non-Markovian processes may also be solved with a large time step by the proposed ANN method, where the conditional collocation points are dependent on past realizations. Fractional Brownian motion (Mandelbrot and Van Ness 1968) forms a relevant example, which is used for the simulation of rough volatility in finance (Gatheral et al. 2018). In such a context, advanced variants of fully connected neural networks, e.g., recurrent neural networks (RNN) or long short-term memory (LSTM) networks (see a review in Yu et al. 2019), are recommended when approximating the nonlinear transition probability function, for example, Equation (13).

As another outlook, multilevel Monte Carlo (MLMC) methods, as developed by (Giles 2008, 2015), form another interesting research topic for our large time step accurate discretization schemes. It is well-known that the strong convergence properties of SDE discretizations impact the efficiency of the MLMC methods.

Author Contributions: Writing—original draft preparation, S.L.; SCMC method and research advice, L.A.G.; Supervision, C.W.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: S.L. would like to thank the China Scholarship Council (CSC) for the financial support.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Longstaff–Schwartz Algorithm

For convenience, we detail the Longstaff–Schwartz Monte Carlo (LSMC) algorithm here (see Algorithm A1).

Algorithm A1: 7L Scheme Longstaff–Schwartz Algorithm

```
1. Divide the time horizon into $N_b$ intervals.
2. Simulate $M$ stock price paths $\hat{Y}_{i,j}$ ($0 \le i \le N_b, 1 \le j \le M$), using the 7L-CDC methodology;
3. Price the Bermudan option by means of the Longstaff–Schwartz Monte Carlo method:
    (a) At terminal time $T_{N_b}$, calculate the payoff $\hat{V}_{N_b,j} = V\left(Y_{N_b,j}\right)$ for all paths $j$, where $V(\cdot)$ is the payoff function.
    (b) Perform a backward recursion, from $i = N_b - 1$ until $i = 0$ as follows:
    (c) Compute the discounted continuation value at time $t_i$, i.e.,
            $\hat{\eta}_{i,j} := \mathrm{e}^{-r\Delta t}\hat{V}_{i+1,j}$    (A1)
    (d) Perform least squares regression at time $t_i$, based on the cross-sectional information $\hat{Y}_{i,j}$ and $\hat{\eta}_{i,j}$ to estimate the conditional expectation function,
            $\bar{\eta}_i(\hat{Y}) = \sum_{k=1}^{M_k} \beta_k B_k(\hat{Y})$    (A2)
        where $M_k$ is the number of the basis functions $B_k(S)$ (polynomial basis, here, $M_k = 3$), and the coefficients $\beta_k$ are constant over different paths $j$. Note that only in-the-money paths are considered in Equations (A1) and (A2),
    (e) For each path $j$, compare the immediate exercise value $V\left(\hat{Y}_{i,j}\right)$ with the estimated continuation value $\bar{\eta}_i\left(\hat{Y}_{i,j}\right)$: If $V\left(\hat{Y}_{i,j}\right) \ge \bar{\eta}_i\left(\hat{Y}_{i,j}\right)$, then $\hat{V}_{i,j} = V\left(\hat{Y}_{i,j}\right)$; else $\hat{V}_{i,j} = \bar{\eta}_i\left(\hat{Y}_{i,j}\right)$.
4. Calculate the option price $V(t_0)$ at the initial time,
            $V(t_0) = \frac{1}{M}\sum_{j=1}^{M} \hat{V}_{0,j}.$    (A3)
```

Note

[1] With seven-league boots, we are marching through the time-wise direction, see also https://en.wikipedia.org/wiki/Seven-league_boots, accessed on 1 September 2020.

References

Bar-Sinai, Yohai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. 2019. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences of the United States of America* 116: 15344–49. [CrossRef] [PubMed]

Beck, Christian, Sebastian Becker, Philipp Grohs, Nor Jaafari, and Arnulf Jentzen. 2018. Solving stochastic differential equations and Kolmogorov equations by means of deep learning. *arXiv* arXiv:1806.00421.

Berrut, Jean-Paul, and Lloyd N. Trefethen. 2004. Barycentric Lagrange Interpolation. *SIAM Review* 46: 501–17. [CrossRef]

Broadie, Mark, and Özgür Kaya. 2006. Exact simulation of stochastic volatility and other affine jump diffusion processes. *Operations Research* 54: 217–31. [CrossRef]

Cameron, Robert H., and William T. Martin. 1947. The orthogonal development of nonlinear functionals in series of Fourier-Hermite functionals. *Annals of Mathematics* 48: 385–92. [CrossRef]

Capriotti, Luca. 2010. Fast Greeks by Algorithmic Differentiation. *Journal of Computational Finance* 14: 3–35. [CrossRef]

Cybenko, George. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* 2: 303–14. [CrossRef]

Fritsch, Frederick N., and Ralph E. Carlson. 1980. Monotone piecewise cubic interpolation. *SIAM Journal on Numerical Analysis* 17: 238–46. [CrossRef]

Gaß, Maximilian, Kathrin Glau, Mirco Mahlstedt, and Maximilian Mair. 2018. Chebyshev interpolation for parametric option pricing. *Finance and Stochastics* 22: 701–31. [CrossRef]

Gatheral, Jim, Thibault Jaisson, and Mathieu Rosenbaum. 2018. Volatility is rough. *Quantitative Finance* 18: 933–49. [CrossRef]

Giles, Michael B. 2008. Multilevel Monte Carlo Path Simulation. *Operations Research* 56: 607–17. [CrossRef]

Giles, Michael B. 2015. Multilevel Monte Carlo methods. *Acta Numerica* 24: 259–328. [CrossRef]

Giles, Michael B., and Paul Glasserman. 2006. Smoking adjoints: Fast Monte Carlo Greeks. *Risk* 19: 88–92.

Glasserman, Paul. 2004. *Monte Carlo Methods in Financial Engineering*. New York: Springer.

Glau, Kathrin, Paul Herold, Dilip B. Madan, and Christian Pötz. 2019. The Chebyshev method for the implied volatility. *Journal of Computational Finance* 23: 1–31.

Glau, Kathrin, and Mirco Mahlstedt. 2019. Improved error bound for multivariate Chebyshev polynomial interpolation. *International Journal of Computer Mathematics* 96: 2302–14. [CrossRef]

Glorot, Xavier, and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. Paper present at Thirteenth International Conference on Artificial Intelligence and Statistics, Sardinia, Italy, May 13–15, pp. 249–56.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. Cambridge: MIT Press.

Grzelak, Lech A. 2019. The collocating local volatility framework—A fresh look at efficient pricing with smile. *International Journal of Computer Mathematics* 96: 2209–28. [CrossRef]

Grzelak, Lech A., Jeroen Witteveen, Maria Suarez-Taboada, and Cornelis W. Oosterlee. 2019. The stochastic collocation Monte Carlo sampler: Highly efficient sampling from expensive distributions. *Quantitative Finance* 19: 339–56. [CrossRef]

Han, Jiequn, Arnulf Jentzen, and E. Weinan. 2018. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences of the United States of America* 115: 8505–10. [CrossRef]

Jain, Shashi, Álvaro Leitao, and Cornelis W. Oosterlee. 2019. Rolling Adjoints: Fast Greeks along Monte Carlo scenarios for early-exercise options. *Journal of Computational Science* 33: 95–112. [CrossRef]

Karatzas, Ioannis, and Steven E. Shreve. 1988. *Brownian Motion and Stochastic Calculus*. New York: Springer.

Kingma, Diederik P., and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv* arXiv:1412.6980.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521: 436–44. [CrossRef]

Leitao, Álvaro, Lech A. Grzelak, and Cornelis W. Oosterlee. 2017. On a one time-step Monte Carlo simulation approach of the SABR model: Application to European options. *Applied Mathematics and Computation* 293: 461–79. [CrossRef]

Li, Xingjie, Fei Lu, and Felix X. F. Ye. 2021. ISALT: Inference-based schemes adaptive to large time-stepping for locally Lipschitz ergodic systems. *arXiv* arXiv:2102.12669.

Longstaff, Francis A., and Eduardo S. Schwartz. 2015. Valuing American Options by Simulation: A Simple Least-Squares Approach. *The Review of Financial Studies* 14: 113–47. [CrossRef]

Mandelbrot, Benoit B., and John W. Van Ness. 1968. Fractional Brownian Motions, Fractional Noises and Applications. *SIAM Review* 10: 422–37. [CrossRef]

Maziar, Raissi, Paris Perdikaris, and George E. Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics* 378: 686–707.

Milstein, Grigori N. 1975. Approximate integration of stochastic differential equations. *Theory of Probability and Its Applications* 19: 557–62.

Montanelli, Hadrien, and Qiang Du. 2019. New Error Bounds for Deep ReLU Networks Using Sparse Grids. *SIAM Journal on Mathematics of Data Science* 1: 78–92. [CrossRef]

Nwankpa, Chigozie, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. 2018. Activation Functions: Comparison of trends in Practice and Research for Deep Learning. *arXiv* arXiv:1811.03378.

Oosterlee, Cornelis W., and Lech A. Grzelak. 2019. *Mathematical Modeling and Computation in Finance*. Singapore: World Scientific.

Platen, Eckhard. 1999. An introduction to numerical methods for stochastic differential equations. *Acta Numerica* 8: 197–246. [CrossRef]

Risken, Hannes. 1984. *The Fokker-Planck Equation: Methods of Solution and Applications*. Springer Series in Synergetics. New York: Springer.

Rivlin, Theodore J. 1990. *Chebyshev Polynomials: From Approximation Theory to Algebra and Number Theory*. Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts. Hoboken: Wiley.

Sirignano, Justin, and Konstantinos Spiliopoulos. 2018. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics* 375: 1339–64. [CrossRef]

Uhlenbeck, George E., and Leonard Ornstein. 1930. On the Theory of the Brownian Motion. *Physical Review* 36: 823–41. [CrossRef]

Xie, You, Erik Franz, Mengyu Chu, and Nils Thuerey. 2018. TempoGAN: A Temporally Coherent, Volumetric GAN for Super-Resolution Fluid Flow. *ACM Transactions on Graphics* 37: 1–15. [CrossRef]

Yarotsky, Dmitry. 2017. Error bounds for approximations with deep ReLU networks. *Neural Networks* 94: 103–14. [CrossRef]

Yu, Yong, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. 2019. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation* 31: 1235–70. [CrossRef]