Learning to Simulate Sequentially Generated Data via Neural Networks and Wasserstein Training

TINGYU ZHU and HAOYU LIU, Peking University, China
ZEYU ZHENG, University of California, Berkeley, USA

9

We propose a new framework of a neural network-assisted sequential structured simulator to model, estimate, and simulate a wide class of sequentially generated data. Neural networks are integrated into the sequentially structured simulators in order to capture potential nonlinear and complicated sequential structures. Given representative real data, the neural network parameters in the simulator are estimated and calibrated through a Wasserstein training process, without restrictive distributional assumptions. The target of Wasserstein training is to enforce the joint distribution of the simulated data to match the joint distribution of the real data in terms of Wasserstein distance. Moreover, the neural network-assisted sequential structured simulator can flexibly incorporate various kinds of elementary randomness and generate distributions with certain properties such as heavy-tail, without the need to redesign the estimation and training procedures. Further, regarding statistical properties, we provide results on consistency and convergence rate for the estimation procedure of the proposed simulator, which are the first set of results that allow the training data samples to be correlated. We then present numerical experiments with synthetic and real data sets to illustrate the performance of the proposed simulator and estimation procedure.

CCS Concepts: • **Computing methodologies** → *Modeling and simulation;*

Additional Key Words and Phrases: Sequential simulator, neural network, Wasserstein training, statistical properties

ACM Reference format:
Tingyu Zhu, Haoyu Liu, and Zeyu Zheng. 2023. Learning to Simulate Sequentially Generated Data via Neural Networks and Wasserstein Training. *ACM Trans. Model. Comput. Simul.* 33, 3, Article 9 (August 2023), 34 pages.
https://doi.org/10.1145/3583070

1 INTRODUCTION

In many applications, such as finance, transportation, and service systems, stochastic simulation models (simulators) that have a sequential structure are widely used to create sample paths and capture the dynamics of relevant multi-dimensional random objects. A sequential-structured simulator typically involves multiple discrete time periods at a certain resolution and models a stochastic process with multi-dimensional state variables. In each time period, the simulator takes the state from the previous time period as input, and generates, along with some new randomness, a new state passing on the next time period. Such simulators are used to simulate *sequentially*

Authors' addresses: T. Zhu and H. Liu, Peking University, Yiheyuan Rd. 5, Haidian, Beijing, China, 100871; emails: {1800017813, 1800015905}@pku.edu.cn; Z. Zheng, University of California, Berkeley, 4125 Etcheverry Hall, Berkeley, CA, USA; email: zyzheng@berkeley.edu.

1049-3301/2023/08-ART9 $15.00
https://doi.org/10.1145/3583070

generated data, which is slightly more general than time series models. The data is in turn used to evaluate expectations/quantiles of functions of the sample paths or to support decision making tasks. For example, in financial applications, a simulator may be used to simulate sequential data that represents the dynamics of prices and volatilities for multiple correlated assets, possibly as well as other relevant factors that impact the asset prices. The simulated data can then be used to evaluate the risks and performances of a portfolio that are composed of these assets.

For some applications, real data are available to calibrate such sequential-structured simulators. Specifically, representative real data may record the dynamics of some, but maybe not all, dimensions of the stochastic process modeled by the sequential-structured simulator. With real data in hand, there is a natural need to tailor the simulator such that the sequentially generated data from the simulator "matches" real data in the corresponding dimensions. Such tasks have always been challenging in terms of both statistical properties and computational demands, due to large data dimension and/or partial observations. In order to calibrate a sequential-structured simulation model, existing methods largely rely on parametric models with specific distributional assumption, such that maximum likelihood method can be used to estimate the parameters using real data. In general, these methods bring up difficult-to-solve estimation procedures especially when real data only contains partial observations in a subset of dimensions. Moreover, specific distributional assumptions are often needed to compute the likelihood function. In addition to the risk of mis-specification, different distributional assumptions may require completely different optimization, computation, and statistical analysis.

To address such challenges, we propose a new framework of sequential simulation assisted by generative adversarial neural networks and Wasserstein training. At each time step, the neural networks take the state vector of the previous time as input, and generate the next state with some additional randomness known as *elementary randomness*. The elementary randomness is prespecified by users according to domain knowledge, whereas parameters of the neural networks, which aim to capture the potential non-linear and complicated dependence of the dynamics on the previous state, are estimated from data. Such estimation is carried out through a Wasserstein training process, which aims to match the (possibly high-dimensional) joint distribution of the simulated data with that of the real data. More specifically, batches of simulated data and real data are passed to another neural network known as the discriminator, which then produces a loss function indicating the similarity of the underlying distributions of both data. The loss function is then alternately maximized via updating the network parameters of the discriminator and minimized via updating the parameters of the simulator networks. We note that this estimation procedure does not require computing likelihood functions, and does not change significantly if the dimensions increase or the type of elementary randomness changes. With such flexibility, the sequential-structured simulator fitted to real data can provide us with information regarding "what-if" scenarios. By altering some parts of the simulator, such as the elementary randomness and length of the sequences, we can answer questions such as "what happens if the variances increase by 10%" and "what happens if the length of the sequences increases by 10%". Further, such modeling scheme allows us to discuss its statistical aspect, which involves three research questions: What types of underlying sequential-structured simulation model can be consistently learned by such framework? What is the statistical rate of convergence if consistency is achieved? To what extent is correlation allowed to exist between different sequences in the sample set, and how much impact does it have on the convergence rate? To the best of our knowledge, such theoretical guarantees do not exist for general sequential-structured simulators assisted by neural networks, especially when the associated distributions have unbounded support.

The modeling and simulation of sequentially generated data has been introduced and intensively studied in the literature. An important class of models is based on parametric assumptions

to capture the dependence structure in the sequences, of which one most representative example is the **stochastic volatility model (SVM)**. As [29] points out, a key intuition in the SVM literature is that the variations in the level of activity is directed by an underlying stochastic process. As an early example of discrete-time stochastic volatility modeling, [32] models the risky part of returns as a product process, integrating an underlying indicator of volatility which follows a non-zero mean Gaussian linear process. Later, continuous-time SVMs formulated as diffusion processes, such as the Heston model presented by [21], become more favorable in portfolio choice and derivatives pricing. The multivariate generalizations of SVMs are presented by [4]. Estimating such models poses substantial challenge due to difficulties in evaluating the exact likelihood function. It is concluded in [8] that there are mainly three categories of estimation techniques to address the challenge, namely estimators based on the method of moments, such as [25]; estimators based on the maximum likelihood principle, such as [28] and estimators based on an auxiliary model, such as [6]. A most representative application of auxiliary models is model calibration, which uses current information such as the option price in parameter estimation, see [1] for example. However, these techniques can become intractable or computationally demanding when the dimension becomes even moderately high. Moreover, considerable assumptions are required, rendering these techniques vulnerable to model mis-specifications.

An alternative way of modeling sequential data and estimating such models is to use the neural network framework. A representative class of models is the **recurrent neural network (RNN)**, which, along with other variants such as **gated recurrent unit (GRU)** and **long-short-term memory (LSTM)**, aims to capture the transition and dependencies between the state vectors in the time-series. Recently, **variational autoencoder (VAE)** provided by [23] is combined with RNN to model and estimate sequentially generated data with a latent stochastic process, see [9, 17, 24, 35] for examples. We also note that [9] and [35] are among the first that integrate these frameworks with Monte Carlo simulation. Such frameworks in general adopt a likelihood-based statistical inference method, using VAE to learn the posterior distributions of latent process variables whose prior distribution and randomness-generation distribution need to be pre-specified. Another branch of neural network-based sequential modeling, including our work, is based on the **generative adversarial network (GAN)** [18]. A GAN is a generative model consisting of a generator and a discriminator, both of which are often neural networks. The generator produces data of some distribution, and the discriminator compares such distributions with the empirical distribution of the sample set. By alternately maximizing the loss function via updating the discriminator and minimizing the loss function via updating the generator, people train the generator network to produce data with the same underlying distribution as the sample set. What is noteworthy about GAN is that it inherently provides a way to compare (possibly high-dimensional) distributions via capturing the most characteristic features, instead of conducting point-wise comparisons or comparing less informative statistics. This serves as an important foundation for application of GANs in stochastic process modeling, which requires learning distributions from data. Representative works such as [16, 31, 36] and [37] design network architectures and loss functions for various specific purposes, such as incorporating extra information as conditions [16], and capturing long-term dependence [36, 37]. We refer to [7] for an overview of application of GANs in time-series modeling, and [14] for an overview of such application in specifically financial time-series data. As stated in both overviews, instability of training, especially when the sample size is limited and severe randomness is included, and lack of proper measurement of how well the distributions are generated are the main problems suffered by applications of GAN in time-series modeling. Answering to such concerns, we use a more model-based approach instead of completely relying on the neural network architectures to capture the distribution from representative data. Also, we use the **Wasserstein generative adversarial network (WGAN)** training framework with gradient

penalty [19] in the estimation procedure to improve training stability as well as provide a basis for our proof of statistical convergence. Apart from that, we propose our own illustrative measurements in the numerical experiments, which either directly use statistics of distributions or resort to a downstream task to provide assessment from an operational aspect. In terms of the sequential structure, [34] consider a different task from ours, inventory optimization, and proposed an RNN-inspired simulation approach to improve computational capabilities for large-scale inventory management.

The Wasserstein training of our neural network-based simulator is inspired by GAN [18], WGAN [2], and the doubly stochastic WGAN framework by [38]. Representative theoretical works like [5, 11] and [12] serve as fundamentals of the proof of our theorem. We also refer to [13] and [20] for descriptions and efficient estimators for general distribution distance metrics such as Wasserstein distance and Kullback-Liebler divergence. We adopt the use of Wasserstein distance to measure distribution distance in this work. We additionally remark that, in financial applications, the drift term and volatility term are separately and explicitly constructed in our framework, which indicates that the induced stochastic process allows a transition to its risk-neutral distribution. The generated risk-neutral paths can be used to estimate the option prices of underlying assets. Therefore, option data can be incorporated, for instance, by adding the mean square error of estimated option prices into the loss function, to jointly learn the real and risk-neutral dynamics. Moreover, options data can be used to fine-tune the parameters in order to assist the training process on price sequences.

The rest of this paper is organized as follows. Section 2 discusses the model setup of the simulator. Section 3 discusses the estimation framework. Section 3.3 discusses the statistical theory for the estimation method. Section 4 provides numerical experiments.

2 MODEL SETUP

We consider a class of simulators that are used to simulate sequential data. A simulator consists of two functions $\mu(\cdot,\cdot)$ and $\Sigma(\cdot,\cdot)$, which take current information as input to generate information about the next step, and incorporate a sequence of elementary randomness, denoted as $\{\eta_k\}$, which contributes to all the randomness in the simulation process. Such simulators generate the dynamics of a stochastic process $(X_k : k = 0, 1, 2, \ldots)$ that takes value in a d-dimensional multi-dimensional real space. That is, $X_k \in \mathbb{R}^d$ for any k. The simulator sequentially updates $(X_k : k = 0, 1, 2, \ldots)$ according to

$$X_{k+1} = \mu(l_k, X_k) + \Sigma(l_k, X_k)\eta_{k+1}, \quad k = 0, 1, 2, \ldots, \tag{1}$$

in which l_k is a real-valued deterministic label that can be used to represent the time-of-day effect or seasonality associated with time period k. The notion $\mu(\cdot,\cdot)$ is a d-dimensional function of the label and the state of the stochastic process in the previous time period. Similarly, $\Sigma(\cdot,\cdot)$ has the same input variables as $\mu(\cdot,\cdot)$, and outputs a $d \times d'$ matrix. The expressions of $\mu(\cdot,\cdot)$ and $\Sigma(\cdot,\cdot)$ can be further specified to incorporate background knowledge. The notion η_{k+1} is referred to as *elementary randomness*, which represents a d'-dimensional mean-zero random vector that is used by the simulator in the time period $k+1$, and $\eta_k : k = 1, 2, \ldots$ are assumed to be independent and identically distributed.

We consider practical applications in which the probability distribution of the elementary randomness η_k's are specified by the users according to background domain knowledge, whereas both $\mu(\cdot,\cdot)$ and $\Sigma(\cdot,\cdot)$ are unknown functions that need to be estimated from empirical data. For many applications, not all dimensions of X_k and not all time periods of data can be observed. Therefore, we consider a flexible data framework for which only the first $d_1 \le d$ dimension of X_k can be observed at selected time periods. Specifically, we write X_k as $(S_k, Y_k)^\top$, where S_k denotes the d_1-

dimensional observed process, and Y_k denotes the $(d-d_1)$-dimensional latent process that is not observed in empirical data. In terms of generality, suppose that the sequence of S_k's can only be observed at p selected time periods labeled as $0 \le k_1 < k_2 < \cdots < k_p$. Set $\boldsymbol{S} = (S_{k_i} : i = 1, 2, \ldots, p)$. We presume that the empirical data is composed of n copies of $\boldsymbol{S}$, denoted as

$$\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots, \boldsymbol{S}_n,$$

which are n identically distributed copies of $\boldsymbol{S}$. Note that both the number of unobserved points between k_i and k_{i+1} $(i = 1, 2, \ldots, p-1)$ and the dimension $d - d_1$ can be either known or specified by the user as part of the modeling assumptions.

The copies of $\boldsymbol{S}$ do not need to be mutually independent in practice. Our goal is to provide a statistical and computational framework to train (or equivalently, to estimate) the simulator given by (1) such that the sequentially generated data $(X_k : k = 0, 1, 2, \ldots)$ matches the joint distribution of $\boldsymbol{S}$ on the corresponding dimensions and time periods.

Before discussing the statistical and computational framework, we briefly describe two examples to demonstrate the relevance of the class of simulators of interest, given by the form of (1). The first example is given by the **multivariate stochastic volatility model (MSVM)** formulated by a stochastic differential equation, which is widely used within the fields of financial economics and mathematical finance to capture the dynamics of asset prices. Specifically, the vector of state variables X_t follows a multivariate diffusion process,

$$dX_t = \mu_c(X_t)dt + \Sigma_c(X_t)dW_t, \quad t \in [0, T], \tag{2}$$

where $X_t = (S_t, Y_t)^\top$, with S_t denoting the observed price process and Y_t denoting the latent volatility process, and $(W_t : t \in [0, T])$ is a canonical multi-dimensional Brownian motion. Most practical simulation tools for the multi-dimensional diffusion model use the idea of discretization, at a user-specified discretization resolution. Using the Euler-Maruyama discretization scheme [27] for example, once a discretization resolution Δt is selected, the simulation process fits into the general simulator considered in (1). Specifically, we have

$$\begin{aligned} \mu(l_k, X_k) &= X_k + \mu_c(X_k)\Delta t, \\ \Sigma(l_k, X_k) &= \Sigma_c(X_k)\sqrt{\Delta t}, \\ \eta_{k+1} &\sim \mathcal{N}(0, I_{d'}). \end{aligned} \tag{3}$$

Namely, $(X_k : k = 0, 1, 2, \ldots)$ is sequentially generated according to

$$X_{k+1} = X_k + \mu_c(X_k)\Delta t + \Sigma_c(X_k)\sqrt{\Delta t}\eta_{k+1}, \tag{4}$$

where $\eta_k, k = 1, 2, \ldots$ are independent standard d'-dimensional multi-variate normal random variables. We use the subscript c for μ_c and Σ_c to indicate that the label l_k is a constant. We additionally remark that, when the empirical data process follows a stationary pattern, we can always set l_k as a constant, which is often the case in practical applications. Therefore, in the rest of this work we mostly consider stationary cases where $l_k = c$, and $\mu(\cdot, \cdot)$ and $\Sigma(\cdot, \cdot)$ are viewed as functions of X_k only. Besides that the simulator considered in (1) covers the simulation process of MSVM, our data framework also accommodates a practical possibility that the resolution at which data is observed can be lower than the resolution at which the simulation of the stochastic process is conducted.

Not only does the simulator defined by (1) allow simulation of the MSVM, but our data framework also accommodates sequential data with heavy-tailed distributions. As the second example, this simulator sequentially updates $(X_k, k = 0, 1, 2, \ldots)$ according to

$$X_{k+1} = X_k + \mu_h(X_k)\Delta t + \Sigma_h(X_k)\Delta\eta_{k+1}, \tag{5}$$

where Δt can be any given resolution, and $\Delta\eta_k, k = 1, 2, \ldots$ are i.i.d. variables with some given heavy-tailed distribution, such as the t distribution or Pareto distribution. This simulator can be used for data modeling within the fields of spectroscopy, particle motion, finance, and so on, where heavy-tailed behaviors are frequently observed. As a more specific case, we take

$$\mu_h(X_k) = b(X_k, \alpha), \quad \Sigma_h(X_k) = 1, \quad \Delta\eta_{k+1} = L^{\alpha}_{k+1}\Delta t^{1/\alpha}, \quad \alpha \in (0, 2) \tag{6}$$

where the definition of $b(\cdot)$ is specified in [30], and $L^{\alpha}_k : k = 1, 2, \ldots$ is a sequence of i.i.d. standard symmetric α-stable random variables which have heavy tails when $\alpha \in (0, 2)$. This is the discretized version of a stochastic differential equation driven by a symmetric α-stable Lévy process.

3 METHOD

In this section, we use a new framework to estimate the simulator so as to match its simulated data with real data. More specifically, we use **neural networks (NN)** to approximate $\mu(\cdot)$ and $\Sigma(\cdot)$ of the simulator, and update the NN parameters to minimize the distance between the joint distribution of simulated data and the joint distribution of real data. To achieve this, we need to specify how the output distribution is generated by the NN-based simulator, how the distance between the two distributions is formulated and computed, and how the NN parameters are updated according to the computed distance. In the following part of this section, Sections 3.1 and 3.2 provide answers to the first two questions, and formulate the estimation problem into a minimax optimization problem. Section 3.3 answers the third question by discussing the training process to solve the optimization problem.

3.1 Neural Network-integrated Simulator

Recall that $\boldsymbol{S} = (S_{k_1}, S_{k_2}, \ldots, S_{k_p})$ represents the observed sequence. Let π denote the true joint probability distribution of $\boldsymbol{S}$. The training data are n identically distributed copies of $\boldsymbol{S}$, given by $\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots, \boldsymbol{S}_n$. These sequences can be either independent or weakly correlated. Let $\tilde{\pi}$ denote the empirical distribution of the data.

The neural network-based simulator generates a sequence of state vectors $(X_k : k = 1, 2, \ldots)$ according to

$$X_{k+1} = \mu_\theta(X_k) + \Sigma_\phi(X_k)\eta_{k+1}, \tag{7}$$

where $\eta_k, k = 1, 2, \ldots$ are given d'-dimensional random vectors, $\mu_\theta(\cdot)$ is a d-dimensional function of X_k, and $\Sigma_\phi(\cdot)$ is a $d \times d'$-dimensional function of X_k which outputs a $d \times d'$ matrix. Both functions adopt the NN architecture, parameterized by NN parameters θ and ϕ, and are approximations of $\mu(\cdot)$ and $\Sigma(\cdot)$ of the simulator. Specifically, given the number of layers $L \in \mathbb{Z}^+$ and the width of the l-th layer n_l, $l = 1, 2, \ldots, L$, for an input variable $X \in \mathbb{R}^d$, the functional forms of $\mu(X; \theta = (\boldsymbol{W}_\theta, \boldsymbol{b}_\theta))$ and $\Sigma(X; \phi = (\boldsymbol{W}_\phi, \boldsymbol{b}_\phi))$ (expanded forms of μ_θ and Σ_ϕ) are given as

$$\begin{aligned} &\mu_0 = X;\ \mu_k = \sigma(W_{\theta,k} \cdot \mu_{k-1} + b_{\theta,k-1}),\ k = 1, 2, \ldots, L-1; \\ &\mu(X; \theta = (\boldsymbol{W}_\theta, \boldsymbol{b}_\theta)) = W_{\theta,L} \cdot \mu_{L-1} + b_{\theta,L}, \end{aligned} \tag{8}$$

and

$$\begin{aligned} &\Sigma_0 = X;\ \Sigma_k = \sigma(W_{\phi,k} \cdot \Sigma_{k-1} + b_{\phi,k-1}),\ k = 1, 2, \ldots, L-1; \\ &\Sigma(X; \phi = (\boldsymbol{W}_\phi, \boldsymbol{b}_\phi)) = W_{\phi,L} \cdot \Sigma_{L-1} + b_{\phi,L}, \end{aligned} \tag{9}$$

where $(\boldsymbol{W}_\theta, \boldsymbol{b}_\theta)$ and $(\boldsymbol{W}_\phi, \boldsymbol{b}_\phi)$ represent all the parameters in the neural networks, which is the aggregation of $W_{\theta,k}$, $b_{\theta,k}$, $W_{\phi,k}$ and $b_{\phi,k}$ $(k = 1, 2, \ldots, L)$, the parameters of each neural network layer. The notation "$\cdot$" represents a dot product. The dimensionality of $(\boldsymbol{W}_\theta, \boldsymbol{b}_\theta)$ and $(\boldsymbol{W}_\phi, \boldsymbol{b}_\phi)$ can be flexibly adjusted for better simulation outcomes, as long as the dimensionalities of the input and

output of $\mu(\cdot;\theta)$ and the input of $\Sigma(\cdot;\phi)$ match that of X_k, and the output of $\Sigma(\cdot;\phi)$ can be reshaped into a matrix to multiply with the elementary random variables η_{k+1}. To train the neural networks to produce results that fit observations is to search for optimal choices of such parameters. The operator $\sigma(\cdot)$ takes a vector of any dimension as input and is a component-wise operator. We specify the operator $\sigma(\cdot)$ as a *rectified linear activation unit*, or *ReLU* for short. Specifically, for $Z \in \mathbb{R}^d$, we have

$$\sigma(Z) = (\max(Z_1, 0), \max(Z_2, 0), \ldots, \max(Z_d, 0)) . \tag{10}$$

Let X_0 be a given constant or a random vector with a given probability distribution π_0. Recall that $X_k = (S_k, Y_k)$, where S_k is the d_1-dimensional observed process, and Y_k is the $(d-d_1)$-dimensional latent process. We additionally remark that even though $\mu(\cdot)$ and $\Sigma(\cdot)$ of the underlying true model are assumed to be stationary, it is still necessary to generate a full sequence instead of modeling a single step of transition. This is due to our assumption of an existing latent process $(Y_k : k = 1, 2, \ldots)$, which is intractable and has to be sequentially simulated. Finally, the joint probability distribution of the generated d-dimensional observed sequence at the measured data points $\hat{\boldsymbol{S}} = (\hat{S}_{k_1}, \hat{S}_{k_2}, \ldots, \hat{S}_{k_p})$, denoted as $\hat{\pi}$, is taken as the output of the generator. Note that $\hat{\pi}$ and $\hat{\boldsymbol{S}}$ are also functions of θ and ϕ, and are therefore sometimes denoted as $\hat{\pi}(\theta, \phi)$ and $\hat{\boldsymbol{S}}(\theta, \phi)$.

3.2 Wasserstein Distance and Discriminator

Next, we introduce the Wasserstein distance, which is used to quantify the distance between two given distributions. The Wasserstein distance of the generated distribution $\hat{\pi}$ and the real distribution π is given by

$$W(\hat{\pi}, \pi) = \inf_{\gamma \in \Pi(\hat{\pi}, \pi)} \mathbb{E}_{(\hat{\boldsymbol{S}}, \boldsymbol{S}) \sim \gamma}[\|\hat{\boldsymbol{S}} - \boldsymbol{S}\|_2], \tag{11}$$

where $\Pi(\hat{\pi}, \pi)$ denotes the set of all joint distributions of which the marginals are respectively $\hat{\pi}$ and π, and $\|\cdot\|$ denotes the L_2 norm. Since the Wasserstein distance in high dimensions does not have a closed form for computation, we often use the Kantorovich-Rubinstein duality given by

$$W(\hat{\pi}, \pi) = \sup_{\|f\|_L \le 1} \mathbb{E}_{\hat{\boldsymbol{S}} \sim \hat{\pi}}[f(\hat{\boldsymbol{S}})] - \mathbb{E}_{\boldsymbol{S} \sim \pi}[f(\boldsymbol{S})], \tag{12}$$

where $\|f\|_L \le 1$ denotes the class of all 1-Lipschitz functions f, i.e., $|f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)| \le \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2$ for any $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^{d_\pi}$. Computation of the supremum over all 1-Lipschitz functions is also analytically intractable, but we can use a neural network f_ψ to approximate f, and search over all such approximations parameterized by NN parameters ψ. With the same network architecture as the simulator networks, f_ψ has functional form given as

$$\begin{aligned} &f_0 = X;\ f_k = \sigma(W_{\psi,k} \cdot f_{k-1} + b_{\psi,k-1}),\ k = 1, 2, \ldots, L-1; \\ &f(X; \psi = (\boldsymbol{W}_\psi, \boldsymbol{b}_\psi)) = W_{\psi,L} \cdot f_{L-1} + b_{\psi,L}. \end{aligned} \tag{13}$$

In the framework of the classical WGAN, a function with the same purpose as f_ψ is known as the *discriminator*. Our method aims to match the generated distribution to the real distribution, which can be achieved through minimizing the Wasserstein distance of the two distributions. We formulate the estimation method as solving the following minimax optimization problem:

$$\min_{\theta \in \Theta, \phi \in \Phi} \max_{\psi \in \Psi} \mathbb{E}_{\hat{\boldsymbol{S}} \sim \hat{\pi}(\theta, \phi)}[f_\psi(\hat{\boldsymbol{S}})] - \mathbb{E}_{\boldsymbol{S} \sim \pi}[f_\psi(\boldsymbol{S})]. \tag{14}$$

The empirical version of problem (14) is given by

$$\min_{\theta \in \Theta, \phi \in \Phi} \max_{\psi \in \Psi} \frac{1}{n} \sum_{j=1}^{n} f_\psi(\hat{\boldsymbol{S}}_j(\theta, \phi)) - \frac{1}{n} \sum_{j=1}^{n} f_\psi(\boldsymbol{S}_j). \tag{15}$$

3.3 Training

In this section, we discuss the training process for model estimation optimization problem (15). We adopt a classical training strategy to solve the minimax problem, which is to alternately update the parameters of the NN-based discriminator and the simulator. Updating the discriminator increases the difference between the two summation terms of (15), which is then attenuated by updating parameters of the simulator. During this process, the discriminator converges to the supreme f over the class of candidate functions, while the output distribution of the simulator converges to the empirical distribution.

We apply the gradient descent method for the training process, which is based on computing gradients of the objective functions to the model parameters. The gradient $\nabla_{\theta,\phi} f_\psi(\hat{\boldsymbol{S}}(\theta,\phi))$ is evaluated through backpropagation using the chain rule, which involves differentiating the entire process of simulation. The diagram for such computation is illustrated in Appendix A.

sectionStatistical Properties In this section, we discuss the statistical property of the estimation method. We prove that the framework proposed in Sections 3.1 and 3.2 can effectively learn distributions of a wide class of sequential data, if the neural network architectures are properly chosen, and the number of copies of $\boldsymbol{S}$, denoted as n, is large enough. In the following Section 3.4, we formulate the statistical convergence problem and describe the requirements and assumptions.

3.4 Formulation of Problem

In this section, we propose three basic requirements on real data, data preparation, and neural network functional class, and explain the reasons for them. Such explanations also shed light on the main ideas of our proof.

3.4.1 Underlying Distribution of Real Data. Let $\boldsymbol{X}_j = (\boldsymbol{S}_j, \boldsymbol{Y}_j)$ denote the sequence. In this section, we prove that the solution of the optimization problem (15) can generate a distribution $\hat{\pi}_S$ of the observed dimensions $\boldsymbol{S}$ that converges to the underlying real distribution π_S of the observed dimensions. Without loss of generality, we assume in this section that all dimensions and time points of the real data are observed, i.e., we have $\boldsymbol{X} = \boldsymbol{S}$. We make this assumption because the convergence of distribution does not include the unobserved dimensions $\boldsymbol{Y}$. Besides, given the value of $\boldsymbol{S}$, the choice of $\boldsymbol{Y}$ has no effect on the optimality of $\boldsymbol{S}$ as a solution of the optimization problem (15). Suppose that the real data, $\boldsymbol{X}_j = (X_{j,0}, X_{j,1}, \ldots, X_{j,p})$, $j = 1, 2, \ldots, n$, is generated as follows: the sequence starts from an initial distribution $X_0 \sim \pi_0$, where $X_0 \in \mathbb{R}^d$, and sequentially proceeds according to

$$X_{i+1} - X_i = \mu(X_i) + \Sigma(X_i)\xi_i. \quad x_i \in \mathbb{R}^d, i = 1, 2, \ldots, p, \tag{16}$$

where ξ_i is some given elementary randomness. The initial distribution π_0 and the integrated functions (μ, Σ) jointly determine the underlying distribution of the real data, denoted as π. To ensure statistical convergence, we assume that π_0 is a known distribution provided to the simulator, and $(\xi_i : i = 1, 2, \ldots, p)$ have the same distribution as the elementary randomness $(\eta_k : k = 1, 2, \ldots, p)$ of the simulator.

Additionally, we can assume different sequences to be independent and identically distributed, or weakly dependent. It is noteworthy that weak correlation allows for applicational situations where all sequences in the sample set are segmented from one single sequence of time-series data. To characterize weak correlation across sequences, we let the first element X_0 of two arbitrary sequences be correlated, namely, $cov(X_0^{(j)}, X_0^{(l)}) \neq 0$ for some $j, l \in \{1, 2, \ldots, n\}$, where n is the sample size. The specific assumption of constraint on correlation will be formulated in the main theorem, where statistical convergence results for both *i.i.d.* and weakly dependent data will also be presented.

3.4.2 Truncation. Classical theoretical results of neural network approximation require the input of the neural network to have bounded support, while in our framework the sequence $(\eta_k : k = 1, 2, \ldots)$ is often set to follow the Gaussian distribution or some heavy-tailed distribution, resulting in unboundedness of the sequence $(X_k : k = 1, 2, \ldots)$. To address the challenge due to unboundedness, we perform truncation methods on both the empirical data and the simulated data. Specifically, we transform the empirical data into its bounded version $\boldsymbol{X}_j^B$ according to

$$X_{j,i}^B = \begin{cases} X_{j,i}, \text{ if } |X_{j,i}| \le B_1; \\ B_1, \text{ else.} \end{cases} \quad i = 0, 1, 2, \ldots p, \quad j = 1, 2, \ldots, n,$$

and the bounded version of simulated data is simulated with bounded elementary randomness given as

$$\eta_k^B = \begin{cases} \eta_k, \text{ if } |\eta_k| \le B_2; \\ B_2, \text{ else.} \end{cases} \quad k = 1, 2, \ldots$$

We note that the real data and simulated data are truncated in different ways. We perform truncation on elementary randomness, instead of truncating after simulating a whole unbounded sequence, because the approximability of μ_θ and Σ_ϕ can only be ensured within bounded areas. We lose control if an unbounded X_i is generated and fed to the networks during the sequential simulation process. However, to ensure that the simulated sequence is also bounded by B_1, we can find some constant C such that $B_2 = C \cdot B_1$, as both truncation bounds B_1 and B_2 increase to infinity. Therefore, for simplicity of notation, we use a common truncation bound B on the truncated data.

We denote the bounded versions of the empirical data and the simulated data as $\boldsymbol{X}_j^B$ and $\hat{\boldsymbol{X}}_j^B$, and their distributions as π^B and $\hat{\pi}^B$. The empirical optimization problem (15) is then transformed into the bounded version:

$$\min_{\theta \in \Theta, \phi \in \Phi} \max_{\psi \in \Psi} \frac{1}{n} \sum_{j=1}^{n} f_\psi(\hat{\boldsymbol{X}}_j^B(\theta, \phi)) - \frac{1}{n} \sum_{j=1}^{n} f_\psi(\boldsymbol{X}_j^B). \tag{17}$$

The Wasserstein distance between the bounded distributions, denoted as $W(\pi^B, \hat{\pi}^B)$, can be controlled. As the size of NN goes to infinity, the truncation bounds also increase to infinity, which, with certain restrictions on η_k that will be presented in the following specific assumption, results in the convergence of the bounded distribution towards its unbounded version, i.e., $\lim_{B \to \infty} W(\pi, \pi^B) = 0$. This convergence enables the controlling of $W(\pi, \pi^B)$, and thus $W(\pi, \hat{\pi}^B)$.

3.4.3 Restrictions on the Discriminator Class. Since taking the supremum over a whole function class, e.g., the bounded 1-Lipschitz class, is computationally intractable, we replace it with a function class of neural networks with bounded size. In this case, the discriminator is a neural network with parameters to be optimized, and defines a metric $d_{\mathcal{F}}(\cdot, \cdot)$ of distributions. A proper discriminator should define a metric under which convergence is equivalent to Wasserstein convergence. This equivalence guarantees that the discriminator effectively distinguishes different distributions, while also making sure that if the training fails to find a solution with small distance under $d_{\mathcal{F}}(\cdot, \cdot)$, then indeed the distributions are far away under Wasserstein distance. Therefore, we impose certain restrictions on the size and components of the discriminator class, so that its discriminative power is proportional to that of the bounded 1-Lipschitz class. Further, such restrictions are helpful when controlling the error induced by weak correlation.

Specifically, restrictions on the discriminator class are imposed through the following definitions.

Definition 3.1 (function class $\mathcal{F}_\Psi$).

$$\mathcal{F}_\Psi(\kappa, L, P, K) = \Big\{ f : \mathbb{R}^d \to \mathbb{R} \,\Big|\, f \text{ in the form of ReLU neural networks with } L \text{ layers and width bounded by } P, \|W_i\|_\infty \le \kappa, \|b_i\|_\infty \le \kappa, \sum_{i=1}^{L} \|W_i\|_0 + \|b_i\|_0 \le K \Big\}.$$

where W_i and b_i, $i = 1, 2, \ldots, L$ are the weight matrices and bias vectors of the layers.

Definition 3.2 (restricted function class $\mathcal{F}_\Psi$).

$$\mathcal{F}_\Psi(\kappa, L, P, K, \epsilon_f) = \Big\{ f \in \mathcal{F}_\Psi(\kappa, L, P, K) \,\Big|\, \|f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)\| \le \|\boldsymbol{x}_1 - \boldsymbol{x}_2\| + 2\epsilon_f, \forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in [-B, B]^d \Big\}.$$

Throughout our proof, we assume that the discriminator class $\mathcal{F}_\Psi$ is restricted as in Definition 3.2, with parameters $\kappa, L, P, K, \epsilon_f$. In proof of the main theorem, we derive specific order of such parameters with regard to sample size n, to ensure statistical convergence. We remark that the additional $2\epsilon_f$ term in 3.2 serves to allow for approximation, in the sense of infinity norm and within a bounded area, of any 1-Lipschitz function by $\mathcal{F}_\psi(\kappa, L, p, K, \epsilon_f)$, which will be crucial in controlling certain error terms. This is because there is no guarantee of the approximation power of the strictly 1-Lipschitz neural network function class, but with theoretical guarantee that $\mathcal{F}_\psi(\kappa, L, P, K)$, for large enough parameters κ, L, P, K, can approximate any 1-Lipschitz function f within distance ϵ_f, we know that $\mathcal{F}_\psi(\kappa, L, p, K, \epsilon_f)$ is dense enough to approximate any 1-Lipschitz function within a bounded area. Conversely, the fact that $\mathcal{F}_\psi(\kappa, L, P, K, \epsilon_f)$ can be approximated by any 1-Lipschitz function f within distance $2\epsilon_f$ is also a basis for controlling certain error terms.

3.5 Specific Assumption and Theorem

We make the following assumption on the underlying true model of the sample data:

Assumption 1. *The following conditions are satisfied for the generation process of sample data (16):*

(1) *All sequences $\boldsymbol{X}_j$ are independent and identically distributed or weakly dependent. A specified characterization of weak dependence is given as follows:*
Let Π_n denote the joint distribution of $(\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n)$. Let $\{\bar{\boldsymbol{X}}_j\}_{j=1}^n$ be independent and identically distributed variables, where $\bar{\boldsymbol{X}}_j$ has the same distribution as $\boldsymbol{X}_j$. Let $\bar{\Pi}_n$ denote the joint distribution of $(\bar{\boldsymbol{X}}_1, \bar{\boldsymbol{X}}_2, \ldots, \bar{\boldsymbol{X}}_n)$, we have

$$W(\Pi_n, \bar{\Pi}_n) \le O(n^{\frac{1}{2}-\beta}), n \to \infty \tag{18}$$

for some $\beta > 0$.
(2) *$\mu : \mathbb{R}^d \to \mathbb{R}^d$ and $\Sigma : \mathbb{R}^d \to \mathbb{R}^{d \times d'}$ are Lipschitz continuous and bounded on $\mathbb{R}^d$.*
(3) *The tail order of the probability density function of every random variable in the sequence of elementary randomness $(\xi_k : k = 1, 2, \ldots)$ is no more than $x^{-(2+\alpha)}$, for some $\alpha > 0$. Namely, $p_{\xi_k}(x) \le O(x^{-(2+\alpha)}), x \to \infty$, for all $k = 1, 2, \ldots$, where $p_{\xi_k}(x)$ is the density function of ξ_k.*

The first condition is assumed in order to effectively control the statistical error, which we will elaborate on in the following subsection. The two requirements of the second condition are interpreted as follows: the Lipschitz continuous requirement for $\mu(\cdot)$ and $\Sigma(\cdot)$ ensures that they can be sufficiently approximated within a bounded area by neural networks with proper architectures. The bounded requirement for $\mu(\cdot)$ and $\Sigma(\cdot)$ restricts the output, and thus every X_k in the sequence, to a bounded area, when the sequence of elementary randomness $(\eta_k : k = 1, 2, \ldots)$ is also bounded or truncated to its bounded version. The third condition is imposed for controlling the bounding

error, which is defined as the Wasserstein distance between the real distribution and its bounded counterpart, i.e., $W(\pi, \pi^B)$.

The main result of statistical analysis is presented as follows:

Theorem 3.3. *Suppose that n copies of i.i.d. or weakly correlated data are available and that the underlying generation model satisfies Assumption 1. Under appropriate specifications of the neural network architecture, let θ^* and ϕ^* be the parameters that solve the optimization problem given as (17), and let the truncation boundary B increase to infinity along with n, by order $B = O(n^{\frac{2}{3pd+6}}), n \to \infty$. We have*

$$\mathbb{E}\left[W(\pi, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*}))\right] \leq O\left(n^{-\frac{1}{3pd+6}}(\log n)^{\frac{3}{2}} + n^{-\frac{2\alpha}{3pd+6}} + n^{-\beta}\right), n \to \infty. \tag{19}$$

where α and β have the same meanings as in Assumption 1, p is the length of the observed sequence, d is the dimension of the observed process. We hide constant coefficients that are independent of n and B, but relevant to p, d, α, the Lipschitz constants and bounds of $\mu(\cdot)$ and $\Sigma(\cdot)$.

The implication of the three terms, $n^{-1/(3pd+6)}(\log n)^{3/2}$, $n^{-2\alpha/(3pd+6)}$, and $n^{-\beta}$, can be explained as follows:

- $n^{-1/(3pd+6)}(\log n)^{3/2}$ is the balanced statistical error. Roughly speaking, balancing is to decide on an appropriate size for the discriminator class, and two groups of parameters are taken into consideration.
  (1) The truncation bound B, which is also proportional to the domain size of the discriminator function, and thus the size of the discriminator class. It should increase fast enough to ensure the convergence rate of $W(\pi, \pi^B)$.
  (2) Other parameters that control the size of the neural network, such as width, depth, number of neurons, and maximum weight.

  The size of the discriminator class should increase at a proper rate, so that the discriminative power of the discriminator class is proportional to that of the 1-Lipschitz function class, but does not become oversize to induce an uncontrollable statistical error. We refer to Section 3.6.4 for details.
- $n^{-2\alpha/(3pd+6)}$ is the bounding error induced by truncation, which is balanced together with the statistical error term. The order of B is selected to make both error terms convergent at similar rates. The bigger α is, the smaller is the tail order of π, and thus also the bounding error.
- $n^{-\beta}$ is the statistical error induced by weak correlation among data. Note that the bigger β is, the bigger are both the weak correlation and this portion of statistical error.

These terms together demonstrate the impact of the dimensionality of data, tail order, and weak correlation on the statistical convergence rate.

Compared with existing work on GAN theory, our statistical theory discusses situations when the input of the discriminator network is sequentially generated and unbounded. Further, most theoretical work such as [3] and [5] lower bounds the discriminative power and upper bounds the generalization error with regard to the discriminator class, but does not derive the statistical convergence rate of $W(\pi, \pi_n^*)$, where π_n^* denotes the optimal solution of the empirical minimax optimization problem for GAN training, and π is the underlying real distribution.

We additionally remark that the assumptions, as well as the truncation strategies, are imposed only to guarantee statistical convergence in the following theorem, and are sometimes unnecessary in practical applications, especially when we have a moderate tolerance for approximation error. For example, to achieve the numerical results in Section 4, we did not perform truncation on the empirical data or the elementary randomness.

3.6 Analysis

In this section, we briefly describe how Theorem 3.3 is proved. We refer to the appendix for details.

3.6.1 Decomposition of Error. The main framework of proof is to control the Wasserstein distance of real distribution and learned simulated distribution using an oracle inequality, decomposing it into generator approximation error, discriminator approximation error, bounding error, and statistical error.

Adopting the idea of [12], we have

$$W(\pi, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*})) \overset{(i)}{\leq} W(\tilde{\pi}^B, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*})) + \underbrace{W(\pi^B, \tilde{\pi}^B)}_{\text{statistical error}} + \underbrace{W(\pi, \pi^B)}_{\text{bounding error}}, \tag{20}$$

$$W(\tilde{\pi}^B, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*})) = d_{\mathcal{F}_\Psi}(\tilde{\pi}^B, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*})) + \underbrace{W(\tilde{\pi}^B, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*})) - d_{\mathcal{F}_\Psi}(\tilde{\pi}^B, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*}))}_{\text{discriminator approximation error I}}, \tag{21}$$

$$d_{\mathcal{F}_\Psi}(\tilde{\pi}^B, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*})) \overset{(ii)}{\leq} d_{\mathcal{F}_\Psi}(\tilde{\pi}^B, \hat{\pi}^B(\mu_{\theta}, \Sigma_{\phi})) \overset{(iii)}{\leq} d_{\mathcal{F}_\Psi}(\pi^B, \hat{\pi}^B(\mu_{\theta}, \Sigma_{\phi})) + \underbrace{d_{\mathcal{F}_\Psi}(\tilde{\pi}^B, \pi^B)}_{\text{statistical error}}, \tag{22}$$

$$d_{\mathcal{F}_\Psi}(\pi^B, \hat{\pi}^B(\mu_{\theta}, \Sigma_{\phi})) = \underbrace{W(\pi^B, \hat{\pi}^B(\mu_{\theta}, \Sigma_{\phi}))}_{\text{generator approximation error}} + \underbrace{d_{\mathcal{F}_\Psi}(\pi^B, \hat{\pi}^B(\mu_{\theta}, \Sigma_{\phi})) - W(\pi^B, \hat{\pi}^B(\mu_{\theta}, \Sigma_{\phi}))}_{\text{discriminator approximation error II}}. \tag{23}$$

Here we carry out the explanation of the inequalities along with specification of some notations. As a general rule, upper subscript B denotes truncated data bounded by some constant B, hat $\hat{\cdot}$ is a notation for simulated distributions, and tilde $\tilde{\cdot}$ is for the empirical distribution of real data, (θ^*, ϕ^*) and (θ, ϕ) denote solutions of different optimization problems described in the following.

- For two distributions P and Q, we have

$$d_{F_\Psi}(P, Q) = \sup_{f_\psi \in \mathcal{F}_\Psi} \mathbb{E}_{X_P \sim P}[f_\psi(X_P)] - \mathbb{E}_{X_Q \sim Q}[f_\psi(X_Q)].$$

  We remark that $d_{\mathcal{F}_\Psi}(\cdot, \cdot)$, which serves as an approximation to $W(\cdot, \cdot)$, is not necessarily nonnegative, but satisfies the inequality

$$d_{F_\Psi}(P, Q) \leq d_{F_\Psi}(P, R) + d_{F_\Psi}(R, Q),$$

  which serves as the reason for (iii). Also, note that (i) is due to the triangular inequality of Wasserstein distance.
- $\tilde{\pi}^B$ is the bounded empirical distribution, of which distance from the bounded real distribution π^B is due to
  (1) limitation in sample size
  (2) weak correlation among the sample sequences, if we allow for it to exist
- The two pairs of parameters for the neural networks integrated in the generator, namely (θ^*, ϕ^*) and (θ, ϕ), are solutions to different optimization problems defined as follows:

$$(\theta^*, \phi^*) = \arg\min_{\theta, \phi} d_{\mathcal{F}_\Psi}(\tilde{\pi}^B, \hat{\pi}^B(\mu_\theta, \Sigma_\phi)), \tag{24}$$

$$(\theta, \phi) = \arg\min_{\theta, \phi} \|\mu_\theta - \mu\|_{L^\infty([-B,B])} + \|\Sigma_\phi - \Sigma\|_{L^\infty([-B,B])}. \tag{25}$$

  Note that (24) is the reason for (ii).

- μ_θ and Σ_ϕ are likely to be different from the optimal solutions μ_{θ^*} and Σ_{ϕ^*} for the following reasons:
  (1) In the minimax optimization problem, the distribution simulated with μ_{θ^*} and Σ_{ϕ^*} is directed towards $\tilde{\pi}^B$ but not π.
  (2) The difference between the function classes $\mathcal{F}_\Psi = \mathcal{F}_\Psi(\kappa, L, P, K, \epsilon_f)$ and $\mathcal{F}_{\text{Lip}} := \{f : \|f\|_L \leq 1\}$ also induces some difference between the "optimal solutions" and the "optimal networks". This portion of error is bounded by *discriminator approximation error* I and II.

3.6.2 Network Approximation. Before controlling the error terms, we introduce the foundation of proof, which is the deep neural network approximation theory. We first present a theorem of approximating C_L-Lipschitz functions on $[-B, B]^d$. Ideas and proofs of this theorem are adopted from [11, 12] and [15].

Theorem 3.4 (Approximation with Explicit Order Dependence on Bound B). *For*

$$f \in [B, B]^d \text{ s.t. } \|f(\boldsymbol{x}) - f(\boldsymbol{y})\| \leq C_L \|\boldsymbol{x} - \boldsymbol{y}\|,$$

we have a neural network Φ_f *with* $\mathcal{L}(\Phi_f) = O(\log B + \log \delta^{-1})$, $\mathcal{W}(\Phi_f) = O(B^d \delta^{-d})$, $\mathcal{B}(\Phi_f) = O(B\delta^{-1})$ *and* $\mathcal{M}(\Phi_f) = O((\log B + \log \delta^{-1}) \cdot B^d \delta^{-d})$, $B \to \infty$ *and* $\delta \to 0$, *satisfying*

$$\|\Phi_f(\boldsymbol{x}) - f(\boldsymbol{x})\|_{L^\infty([-B,B]^d)} \leq \delta. \tag{26}$$

where notations for the size of ReLU network Φ *are defined as*

- *depth* $\mathcal{L}(\Phi) = L$
- *width* $\mathcal{W}(\Phi) = \max_{l=0,\ldots,L} N_l$, *where* N_l *is the width of the* l*th layer*
- *weight magnitude* $\mathcal{B}(\Phi) = \max_{l=1,\ldots,L} \max\{\|W_l\|_\infty, \|b_l\|_\infty\}$
- *number of neurons* $\mathcal{M}(\Phi) = \sum_{l=1}^{L} \|W_l\| + \|b_l\|$

The proof of Theorem 3.4 consists of two steps: approximating Lipschitz functions with interpolation polynomials, and approximating the polynomials with neural networks. We refer to Appendix B for details.

3.6.3 Controlling Error Terms. In this section, we describe how each error term is controlled. We defer more details to the appendix to complete the proof.

First, for bounding error $W(\pi, \pi^B)$, applying the definition of Wasserstein distance and using the condition that $p_{\eta_k}(x) \leq O(x^{-(2+\alpha)})$, $x \to \infty$, we have

$$W(\pi, \pi^B) \leq O(B^{-\alpha}), \quad B \to \infty. \tag{27}$$

Second, for generator approximation error $W(\pi^B, \hat{\pi}^B(\mu_\theta, \Sigma_\phi))$, which is induced by the errors of approximating μ and Σ with optimal neural networks μ_θ and Σ_ϕ, denoted as ϵ_μ and ϵ_Σ, we apply the law of total expectation on the sequence to derive that the generator approximation error can be bounded as

$$W(\pi^B, \hat{\pi}^B(\mu_\theta, \Sigma_\phi)) \leq O(\epsilon_\mu + \epsilon_\Sigma), \quad \epsilon_\mu \to 0 \text{ and } \epsilon_\Sigma \to 0. \tag{28}$$

Note that, replacing δ in Theorem 3.4 with ϵ_μ and ϵ_Σ, we can derive the required sizes of generator networks μ_θ and Σ_ϕ to achieve levels ϵ_μ, ϵ_Σ of approximation errors.

Next, we control the discriminator approximation error terms. We have the following lemma, the proof of which is given in Appendix D.

Lemma 3.5. *The two function classes* $\mathcal{F}_\Psi(\kappa, L, P, K, \epsilon_f)$ *and* 1*-Lipchitz class* $\mathcal{F}_{Lip}$ *can approximate each other within the bounded area* $[-B, B]^d$, *namely,*

(1) $\forall f \in \mathcal{F}_{Lip}, \exists f_\psi \in \mathcal{F}_\Psi$, *such that* $\|f - f_\psi\|_{L^\infty[-B,B]^d} \le \epsilon_f$;
(2) $\forall f \in \mathcal{F}_\Psi, \exists f_\psi \in \mathcal{F}_{\text{Lip}}$, *such that* $\|f - f_\psi\|_{L^\infty[-B,B]^d} \le 3\epsilon_f$.

Using Lemma 3.5, we have for the discriminator approximation error terms,

$$\begin{aligned}
&W(\tilde{\pi}^B, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*})) - d_{\mathcal{F}_\Psi}(\tilde{\pi}^B, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*})) \\
=& \sup_{\|f\|_L \le 1} [\mathbb{E}_{X\sim\tilde{\pi}^B} f(X) - \mathbb{E}_{X\sim\hat{\pi}^B} f(X)] - \sup_{f_\psi \in \mathcal{F}_\Psi} \left[\mathbb{E}_{X\sim\tilde{\pi}^B} f_\psi(X) - \mathbb{E}_{X\sim\hat{\pi}^B} f_\psi(X)\right] \\
=& \inf_{f_\psi \in \mathcal{F}_\Psi} \sup_{\|f\|_L \le 1} \mathbb{E}_{X\sim\tilde{\pi}^B} \left[f(X) - f_\psi(X)\right] + \inf_{f_\psi \in \mathcal{F}_\Psi} \sup_{\|f\|_L \le 1} \mathbb{E}_{X\sim\hat{\pi}^B} \left[f_\psi(X) - f(X)\right] \\
\le& \inf_{f_\psi \in \mathcal{F}_\Psi} \sup_{\|f\|_L \le 1} 2\|f - f_\psi\|_\infty \le 2\epsilon_f.
\end{aligned} \tag{29}$$

Also,

$$\begin{aligned}
&d_{\mathcal{F}_\Psi}(\pi^B, \hat{\pi}^B(\mu_\theta, \Sigma_\phi)) - W(\pi^B, \hat{\pi}^B(\mu_\theta, \Sigma_\phi)) \\
=& \sup_{f_\psi \in \mathcal{F}_\Psi} \left[\mathbb{E}_{X\sim\pi^B} f_\psi(X) - \mathbb{E}_{X\sim\hat{\pi}^B} f_\psi(X)\right] - \sup_{\|f\|_L \le 1} [\mathbb{E}_{X\sim\pi^B} f(X) - \mathbb{E}_{X\sim\hat{\pi}^B} f(X)] \\
=& \inf_{\|f\|_L \le 1} \sup_{f_\psi \in \mathcal{F}_\Psi} \mathbb{E}_{X\sim\pi^B} \left[f(X) - f_\psi(X)\right] + \inf_{\|f\|_L \le 1} \sup_{f_\psi \in \mathcal{F}_\Psi} \mathbb{E}_{X\sim\hat{\pi}^B} \left[f_\psi(X) - f(X)\right] \\
\le& \inf_{\|f\|_L \le 1} \sup_{f_\psi \in \mathcal{F}_\Psi} 2\|f - f_\psi\|_\infty \le 6\epsilon_f,
\end{aligned} \tag{30}$$

After that, we control the statistical error terms $d_{\mathcal{F}_\Psi}(\tilde{\pi}^B, \pi^B)$ and $W(\pi^B, \tilde{\pi}^B)$. We have

$$\begin{aligned}
\mathbb{E}\left[d_{\mathcal{F}_\Psi}(\tilde{\pi}^B, \pi^B)\right] &= \mathbb{E}\left[\sup_{f_\psi \in \mathcal{F}_\Psi} \frac{1}{n}\sum_{j=1}^n f_\psi(X_j) - \mathbb{E}_{Y\sim\pi^B}[f_\psi(Y)]\right] \\
&\le \mathbb{E}_X \mathbb{E}_{Y_j\sim\pi^B, i.i.d.}\left[\sup_{f\in\mathcal{F}_\Psi} \frac{1}{n}\sum_{j=1}^n [f_\psi(X_j) - f_\psi(Y_j)]\right].
\end{aligned} \tag{31}$$

If X_j are correlated, according to the assumption of weak correlation, we created independent variables $\bar{X}_j$ such that the joint distribution of $(X_1, X_2, \ldots, X_n)$ and $(\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_n)$, denoted as $\gamma(\Pi_n, \bar{\Pi}_n)$, satisfies

$$\gamma^*(\Pi_n, \bar{\Pi}_n) = \arg\inf_{\gamma\in\gamma(\Pi_n,\bar{\Pi}_n)} \mathbb{E}_{(X,\bar{X})\sim\gamma(\Pi_n,\bar{\Pi}_n)}\left[\|X - \bar{X}\|\right].$$

According to the definition of function class $\mathcal{F}_\Psi(\kappa, L, P, K, \epsilon_f)$, we have

$$\begin{aligned}
&\mathbb{E}_X \mathbb{E}_{Y_j\sim\pi^B, i.i.d.}\left[\sup_{f\in\mathcal{F}_\Psi} \frac{1}{n}\sum_{j=1}^n [f_\psi(X_j) - f_\psi(Y_j)]\right] \\
\le& \mathbb{E}\left[\sup_{f\in\mathcal{F}_\Psi} \frac{1}{n}\sum_{j=1}^n [f_\psi(\bar{X}_j) - f_\psi(Y_j)]\right] + \mathbb{E}\left[\frac{1}{n}\sum_{j=1}^n \|X_j - \bar{X}_j\|\right] + 2\epsilon_f.
\end{aligned}$$

[12] suggests that

$$\mathbb{E}\left[\sup_{f_\psi\in\mathcal{F}_\Psi} \frac{1}{n}\sum_{j=1}^n [f_\psi(\bar{X}_j) - f_\psi(Y_j)]\right] \le 2 \inf_{0<\delta<M}\left(2\delta + \frac{12}{\sqrt{n}}\int_\delta^M \sqrt{\log \mathcal{N}(\epsilon, \mathcal{F}_\Psi, \|\cdot\|_\infty)}d\epsilon\right), \tag{32}$$

where $M = \text{diam}(\mathcal{F}_\Psi(\kappa, L, P, K, \epsilon_f)) \le R = O(B)$, and $\mathcal{N}(\epsilon, \mathcal{F}_\Psi, \|\cdot\|_\infty)$ is the covering number [26] of ϵ-balls over the functional class $\mathcal{F}_\Psi$ under distance $\|\cdot\|_\infty$. Further, let $f_\psi, f_{\psi'} \in \mathcal{F}_\Psi(\kappa, L, P, K)$ with

all weight parameters at most h from each other, [12] shows that with bounded support $\|x\|_\infty \le B$, we have

$$\|f_\psi - f_{\psi'}\|_\infty \le hL(PB+2)(\kappa P)^{L-1}.$$

Discretizing each parameter uniformly into κ/h grids yields a δ-covering on $\mathcal{F}_\Psi$, therefore,

$$\mathcal{N}(\delta, \mathcal{F}_\Psi(\kappa, L, P, K), \|\cdot\|_\infty) \le (LP^2)^K \left(\frac{2\kappa}{h}\right)^K,$$

where

$$h = \frac{\delta}{L(PB+2)(\kappa P)^{L-1}}.$$

Further, we have

$$\mathcal{N}(\delta, \mathcal{F}_\Psi(\kappa, L, P, K, \epsilon_f), \|\cdot\|_\infty) \le \mathcal{N}\left(\frac{\delta}{2}, \mathcal{F}_\Psi(\kappa, L, P, K), \|\cdot\|_\infty\right) \le \left(\frac{4L^2(PB+2)(\kappa P)^{L+1}}{\delta}\right)^K. \quad (33)$$

Substituting (33) into (32) and taking $\delta = 1/n$ yields

$$\mathbb{E}\left[\sup_{f\in\mathcal{F}_\Psi} \frac{1}{n}\sum_{j=1}^n [f_\psi(\bar{X}_j) - f_\psi(Y_j)]\right] \le O\left(B\sqrt{n^{-1}KL\log(nL\kappa PB)}\right).$$

Also, by Chebyshev inequality,

$$\mathbb{E}\left[\frac{1}{n}\sum_{j=1}^n \|X_j - \bar{X}_j\|\right] \le \frac{1}{\sqrt{n}} W(\Pi_n, \bar{\Pi}_n) = O\left(n^{-\beta}\right).$$

Similarly, for the Wasserstein statistical error $W(\pi^B, \tilde{\pi}^B)$, we have

$$\mathcal{N}(\delta, \mathcal{F}_{\text{Lip}}, \|\cdot\|_\infty) \le \left(\frac{2B}{\delta}+1\right)^{pd(2B/\delta+1)}.$$

Therefore, taking $\delta^{-1} = n^{(pd-1)/(pd+1)}$, we have

$$2\inf_{0<\delta<B}\left(2\delta + \frac{12}{\sqrt{n}}\int_\delta^B \sqrt{\log \mathcal{N}(\delta, \mathcal{F}_{\text{Lip}}, \|\cdot\|_\infty)}d\epsilon\right) \le O\left(B^{\frac{3}{2}} n^{-\frac{1}{pd+1}}\sqrt{\log(Bn)}\right).$$

And finally,

$$W(\pi^B, \tilde{\pi}^B) \le O\left(B^{\frac{3}{2}} n^{-\frac{1}{pd+1}}\sqrt{\log(nB)} + n^{-\beta}\right) + 2\epsilon_f.$$

3.6.4 Balancing. Finally, we balance the error terms relevant to the discriminator class. Summarizing the four parts of errors, we have

$$\begin{aligned}&\mathbb{E}\left[W(\pi, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*}))\right] \\ &= O\left(B^{-\alpha} + \epsilon_\Sigma + \epsilon_\mu + \epsilon_f + B\sqrt{n^{-1}KL\log(nL\kappa PB)} + B^{\frac{3}{2}} n^{-\frac{1}{pd+1}}\sqrt{\log(Bn)} + n^{-\beta}\right). \quad (34)\end{aligned}$$

Also, since ϵ_f is the approximation tolerance of the discriminator class, we have from Theorem 3.4, $L = O(\log B + \log \epsilon_f^{-1})$, $\kappa = O(B\epsilon_f^{-1})$, $P = O(B^{pd}\epsilon_f^{-pd})$ and $K = O((\log B + \log \epsilon_f^{-1})\cdot B^{pd}\epsilon_f^{-pd})$. To

balance these terms, we mostly pay attention to $B\sqrt{n^{-1}KL\log(nL\kappa PB)}$ and $B^{\frac{3}{2}}n^{-\frac{1}{pd+1}}\sqrt{\log(Bn)}$. To make sure that these two terms converge to 0 as $n \to \infty$, let $B = n^{k_1}$ and $\epsilon_f = n^{-k_2}$, we have

$$\begin{cases} \left(\frac{pd}{2}+1\right)k_1 + \frac{pd}{2}k_2 < \frac{1}{2}. \\ \frac{3}{2}k_1 < \frac{1}{pd+1}, \end{cases}$$

We can set $k_1 = \frac{2}{3pd+6}$ and $k_2 = \frac{1}{3pd+6}$. Also, make the generator network classes large enough so that ϵ_Σ and ϵ_μ are not of leading order, then

$$\mathbb{E}\left[W(\pi, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*}))\right] \le O\left(n^{-\frac{1}{3pd+6}}(\log n)^{\frac{3}{2}} + n^{-\frac{1}{(pd+2)(pd+1)}}(\log n)^{\frac{1}{2}} + n^{-\frac{2\alpha}{3pd+6}} + n^{-\beta}\right).$$

With $pd \ge 3$, we have

$$\mathbb{E}\left[W(\pi, \hat{\pi}^B(\mu_{\theta^*}, \Sigma_{\phi^*}))\right] \le O\left(n^{-\frac{1}{3pd+6}}(\log n)^{\frac{3}{2}} + n^{-\frac{2\alpha}{3pd+6}} + n^{-\beta}\right). \tag{35}$$

and this is achieved by taking $B = O(n^{\frac{2}{3pd+6}})$,

$$L = O(\log n),\ \kappa = O\left(n^{\frac{1}{pd+2}}\right),\ P = O\left(n^{\frac{pd}{pd+2}}\right),\ K = O\left(n^{\frac{pd}{pd+2}}\log n\right),$$

for the discriminator, where $n \to \infty$. For the generator networks μ_θ and Σ_ϕ, we take

$$\epsilon_\mu = \epsilon_\Sigma = O\left(\min\left\{n^{-\frac{1}{3pd+6}}(\log n)^{\frac{3}{2}},\ n^{-\frac{2\alpha}{3pd+6}},\ n^{-\beta}\right\}\right),$$

and the network sizes of μ_θ and Σ_ϕ are accordingly

$$L = O(\log n),\ \kappa = O\left(n^{\frac{2}{3pd+6}}\epsilon_\mu^{-1}\right),\ P = \left(n^{\frac{2pd}{3pd+6}}\epsilon_\mu^{-pd}\right),\ K = O\left(n^{\frac{2pd}{3pd+6}}\epsilon_\mu^{-pd}\log n\right)$$

This result provides insights for selecting the neural network sizes, when the training set is large and high precision of modeling is expected to be achieved.

4 NUMERICAL EXPERIMENTS

In this section, we evaluate the performance of the simulator estimated by our proposed framework, using four sets of synthetic data (Sections 4.1, 4.2, 4.3, and 4.4) and one set of real data (Section 4.5) as the training data.[1] In all experiments, we illustrate the use of the simulator by considering scenarios where the simulator is applied to generate sequences of prices of multiple correlated assets. Our proposed framework then aims to estimate the simulators such that the joint distribution of the simulated data has a close Wasserstein distance compared to that of the training data. To demonstrate the performance of estimated simulators in their practical use, we first consider the task of evaluating the distribution of the **maximal drawdown (MDD)** for all the assets in consideration. In each experiment, we consider the distribution of the MDD of all observed dimensions of the sequential data. We compare the MDD distribution of the simulated data-based portfolio against the MDD distribution of the empirical data-based portfolio. The simulator that simulates the sequential price data is estimated by our proposed method. In this way, we aim to demonstrate the performance of our method from an operational performance point of view. We present the definition of maximal drawdown, which is derived from [10]:

[1] see https://github.com/Goldenbean0521/Sequential-code for code.

Definition 4.1. Let $(H_i \in \mathbb{R} : i = 0, 1, 2, \ldots, p)$ be a portfolio sequence, and let H_i be the portfolio value at time step i. The portfolio *drawdown* at time step i is defined by

$$D_i = \max_{0 \le l \le i} \frac{H_l - H_i}{H_i}, \tag{36}$$

and the *maximal drawdown* of a sequence is the maximum value of D_i over all time steps $i = 0, 1, 2, \ldots, p$, namely, $M = \max_{0 \le i \le p} D_i$.

Apart from that, we demonstrate the ability of the network to capture the correlation among the observed dimensions. Specifically, we use the trained networks to simulate 5,000 copies of sequences, estimate the correlation matrices of all observed dimensions at certain time points, and compare such matrices to that of the real (synthetic) data. Such operation is then replicated 100 times to produce a mean value and a standard deviation of the estimations.

4.1 Multi-dimensional Heston Model

In this subsection, we use a synthesized data set of three stock prices that is generated by a multi-dimensional Heston model.

4.1.1 Underlying Model for Synthetic Data: Multi-dimensional Heston. The observed data is 3-dimensional. For each dimension, the underlying stochastic process is formulated by a stochastic differential equation known as the *Heston model* (see [1], which also presents the values of the model parameters):

$$d\begin{pmatrix} S_t \\ Y_t \end{pmatrix} = \begin{pmatrix} \mu S_t \\ \kappa(\gamma - Y_t) \end{pmatrix} dt + \begin{pmatrix} S_t\sqrt{(1-\rho^2)Y_t} & \rho S_t \sqrt{Y_t} \\ 0 & \sigma\sqrt{Y_t} \end{pmatrix} dW_t \tag{37}$$

where $\mu, \kappa, \gamma, \rho, \sigma$ are given parameters, S_t is the observed price process, Y_t is the latent volatility process, and $W_t = (W_t^S, W_t^Y)^\top$ is the 2-dimensional canonical Brownian motion. We use a matrix L to induce correlation among the three stochastic processes, namely, we let

$$d\begin{pmatrix} X_{1,t} \\ X_{2,t} \\ X_{3,t} \end{pmatrix} = \begin{pmatrix} \mu_{1,t} \\ \mu_{2,t} \\ \mu_{3,t} \end{pmatrix} dt + (L \otimes I_2) \cdot \begin{pmatrix} \Sigma_{1,t} & 0 & 0 \\ 0 & \Sigma_{2,t} & 0 \\ 0 & 0 & \Sigma_{3,t} \end{pmatrix} d\begin{pmatrix} W_{1,t} \\ W_{2,t} \\ W_{3,t} \end{pmatrix}, \tag{38}$$

where

$$X_{i,t} = \begin{pmatrix} S_{i,t} \\ Y_{i,t} \end{pmatrix}, \quad \mu_{i,t} = \begin{pmatrix} \mu_i S_{i,t} \\ \kappa_i(\gamma_i - Y_{i,t}) \end{pmatrix}, \quad \Sigma_{i,t} = \begin{pmatrix} S_{i,t}\sqrt{(1-\rho_i^2)Y_{i,t}} & \rho_i S_{i,t}\sqrt{Y_{i,t}} \\ 0 & \sigma_i\sqrt{Y_{i,t}} \end{pmatrix},$$
$$W_{i,t} = \begin{pmatrix} W_{i,t}^S \\ W_{i,t}^Y \end{pmatrix},$$

and $\otimes$ denotes the Kronecker product.

The model parameters are set as in Table 1. Additionally, we have $\mu = r - d$, and L is the Cholesky decomposition of P, given as

$$LL^\top = P = \begin{pmatrix} 1 & 0.3 & 0.2 \\ 0.3 & 1 & 0.4 \\ 0.2 & 0.4 & 1 \end{pmatrix}.$$

We next describe how the data set is synthesized. The initial values are given by $S_0 \sim \mathcal{N}((100, 100, 100), 70P)$, and $Y_0 = (0.1, 0.1, 0.1)$. There are $n = 5{,}000$ sequences generated in total, each having $p = 15$ transitions, with the weekly frequency $\Delta = 7/365$ as the resolution for observation. We use an Euler discretization of the process, setting 30 sub-intervals between every two

Table 1. Parameters of the Underlying Multi-dimensional Heston Model

	r	d	σ	ρ	κ	γ
d_1	0.04	0.015	0.25	−0.8	3	0.1
d_2	0.04	0.015	0.2	−0.75	2.7	0.11
d_3	0.04	0.015	0.15	−0.85	3.3	0.09

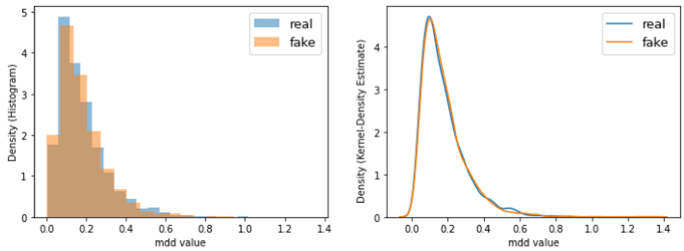

Fig. 1. Maximal draw down distribution, 3-dimensional Heston model, first dimension.

observations, which implies that the resolution for generation is set as $7/(365\times 30)$. This synthetic data set is then used as input data for the discriminator.

4.1.2 Training Process and Results. We specify the structure of the simulator as

$$X_{k+1} = X_k + \mu_\theta(X_k)\Delta t + \Sigma_\phi(X_k)\sqrt{\Delta t}\eta_{k+1} \tag{39}$$

where $\eta_k : k = 1, 2, \ldots$ are independent 6-dimensional canonical normal variables, Δt is set as $0.01/15$, which is not the same as the length of the sub-intervals used in synthetic data generation, but induces only scaling differences that can be eliminated by network parameters. The parameterization of μ_θ is given by $L = 2$, $\tilde{n} = (n_1, n_2) = (100, 6)$. The parameterization of Σ_ϕ is given by $L = 2$, $\tilde{n} = (n_1, n_2) = (100, 36)$. The parameterization of f_ψ is given by $L = 3$, $\tilde{n} = (n_1, n_2, n_3) = (500, 500, 1)$. The initialization of all the parameters of the weight matrices W_l's of μ_θ, Σ_ϕ, f_ψ are given by independent Gaussian random variables with mean 0 and variance 0.1. The vectors b_l's are initialized as constant 3. The gradient penalty coefficient for f_ψ is set as 1, and the batch size for sampling from synthetic data and for simulator generation is set as 256. The initial values S_0 of each simulation process is set to be the same as the initial values of the sample batch, and Y_0 is set as $(0.1, 0.1, 0.1)$. The training process is carried out with 500 iterations using the Adam optimizer [22] with coefficients $\beta_1 = 0.5$ and $\beta_2 = 0.9$. Within each iteration, f_ψ is updated five times. The learning rate of μ_θ, Σ_ϕ and f_ψ decays exponentially from $1e-4$ to $1e-6$. The training takes about 10 minutes using the GPU resource on Google Colab.

For evaluation of training, we first illustrate the comparison between maximal drawdown distribution of the three dimensions. The sizes of the synthesized data set and the simulated data set used for comparison are both 5,000. The results are illustrated in the following Figures 1, 2, and 3.

We also investigate the correlation matrix of the three observed dimensions of data at the 15th observation. By simulating 5,000 sequences using the trained networks μ_θ and Σ_ϕ each time to estimate the correlation matrix, and replicating 100 times to derive the mean value and standard deviation of simulated estimations, we have the following result in Table 2.

4.2 Multi-dimensional Heston Model with Highly Correlated Training Sequences

In this subsection, we demonstrate the performance of our proposed framework when the training sequences are correlated with each other.

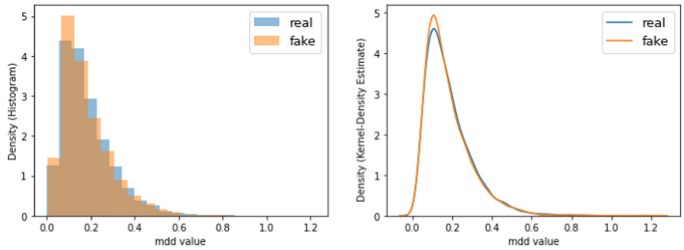

Fig. 2. Maximal draw down distribution, 3-dimensional Heston model, second dimension.

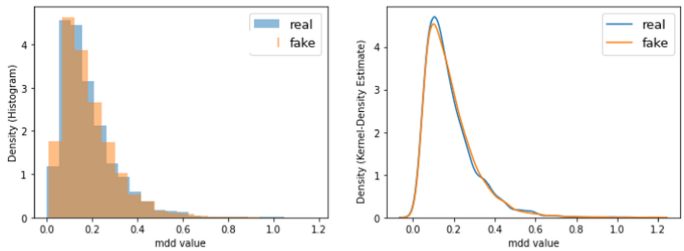

Fig. 3. Maximal draw down distribution, 3-dimensional Heston model, third dimension.

Table 2. Simulated Correlation of the Three Observed Dimensions at Time Step $i = 15$, Multi-dimensional Heston Model

	True value	Sim. mean	Std. dev.
$\text{corr}(S_1, S_2)$	0.3	0.295	0.012
$\text{corr}(S_1, S_3)$	0.2	0.200	0.014
$\text{corr}(S_2, S_3)$	0.4	0.383	0.011

4.2.1 Underlying Model for Synthetic Data: Multi-dimensional Heston. The underlying stochastic process is formulated by the same stochastic differential equation as (37) and (38).

We next describe how the data set is synthesized. The initial values are given by $S_0 \sim \mathcal{N}((100, 100, 100), 70P)$, and $Y_0 = (0.1, 0.1, 0.1)$. There are $n = 5{,}000$ sequences generated in total, each having $p = 15$ transitions, with the weekly frequency $\Delta = 7/365$ as the resolution for observation. We use an Euler discretization of the process, setting 30 sub-intervals between every two observations, which implies that the resolution for generation is set as $7/(365 \times 30)$. Namely, $(X_{i,k} : i = 1, 2, 3; k = 0, 1, 2, \ldots)$ is sequentially generated according to

$$\begin{pmatrix} X_{1,k+1} \\ X_{2,k+1} \\ X_{3,k+1} \end{pmatrix} = \begin{pmatrix} X_{1,k} \\ X_{2,k} \\ X_{3,k} \end{pmatrix} + \begin{pmatrix} \mu_{1,k} \\ \mu_{2,k} \\ \mu_{3,k} \end{pmatrix} \Delta t + (L \otimes I_2) \cdot \begin{pmatrix} \Sigma_{1,k} & 0 & 0 \\ 0 & \Sigma_{2,k} & 0 \\ 0 & 0 & \Sigma_{3,k} \end{pmatrix} d \begin{pmatrix} \sqrt{\Delta t}\eta_{1,k+1} \\ \sqrt{\Delta t}\eta_{2,k+1} \\ \sqrt{\Delta t}\eta_{3,k+1} \end{pmatrix}, \quad (40)$$

where

$$X_{i,k} = \begin{pmatrix} S_{i,k} \\ Y_{i,k} \end{pmatrix}, \quad \mu_{i,k} = \begin{pmatrix} \mu_i S_{i,k} \\ \kappa_i(\gamma_i - Y_{i,k}) \end{pmatrix}, \quad \Sigma_{i,k} = \begin{pmatrix} S_{i,k}\sqrt{(1-\rho_i^2)Y_{i,k}} & \rho_i S_{i,k}\sqrt{Y_{i,k}} \\ 0 & \sigma_i\sqrt{Y_{i,k}} \end{pmatrix},$$

$$\eta_{i,k+1} = \begin{pmatrix} \eta^S_{i,k+1} \\ \eta^Y_{i,k+1} \end{pmatrix}.$$

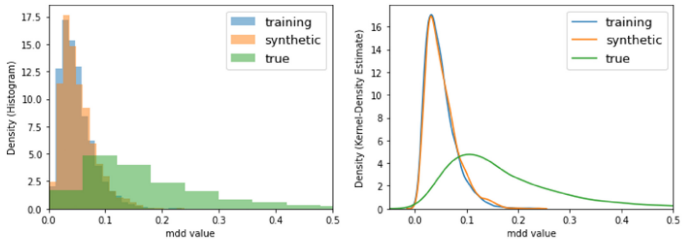

Fig. 4. Maximal draw down distribution, 3-dimensional Heston model with highly correlated training sequences, first dimension.

$\eta^j_{i,k}, i = 1,2,3; k = 1,2,\ldots, j = S, Y$ are independent standard normal random variables. To create correlation among the 5,000 training-sequences, for each i, k, and j, the $\eta^j_{i,k}$'s in the 5,000 sequences are simultaneously generated from a 5,000-dimensional multivariate normal distribution $\mathcal{N}(0, \Sigma)$, where

$$\Sigma = \begin{pmatrix} 1 & \rho & \cdots & \rho \\ \rho & 1 & \cdots & \rho \\ \vdots & \vdots & \ddots & \vdots \\ \rho & \rho & \cdots & 1 \end{pmatrix}.$$

We set $\rho = 0.9$ in our synthetic data set. In other words, for each $i \in \{1,2,3\}, k \in \{1,2,\ldots\}, j \in \{S, Y\}$, the correlation of $\eta^j_{i,k}$'s (i.e., the random noises) in any two different sequences is 0.9, which ensures that the training-sequences are highly correlated with each other. This synthetic data set is then used as input data for the discriminator.

4.2.2 Training Process and Results. We specify the structure of the simulator to have the same form as (39). The parameterization and initialization of the neural networks μ_θ, Σ_ϕ and f_ψ, as well as the iterative optimization process are similar to those of the first experiment. Note that although the training sequences are highly correlated with each other, the sequences in the simulated set generated by our framework are independent. The training takes about 10 minutes.

Since the training sequences are highly correlated with each other, the empirical distribution of the training data might deviate from the real distribution of the underlying process. Considering this difference, we generate another data set with independent sequences (hereafter "uncorrelated set") according to (40). The statistics of the uncorrelated set will be unbiased estimators of the real statistics of the underlying process. For evaluation of training, we compare the distribution of our simulated data set with the distribution of both the training set and the uncorrelated set.

First, we illustrate the comparison among the maximal drawdown distribution of the three dimensions. The sizes of the synthesized data set and the simulated data set used for comparison are both 5,000. We have the following results in Figures 4, 5, and 6. The results indicate that although the empirical distribution of the training data has deviated from the real distribution of the underlying process, the distribution of the synthesized data set is still comparable to the distribution of the training set.

We also investigate the correlation matrix of the three observed dimensions of data at the 15th observation. By simulating 5,000 sequences using the trained networks μ_θ and Σ_ϕ each time to estimate the correlation matrix, and replicating 100 times to derive the mean value and standard deviation of simulated estimations, we have the following result in Table 3.

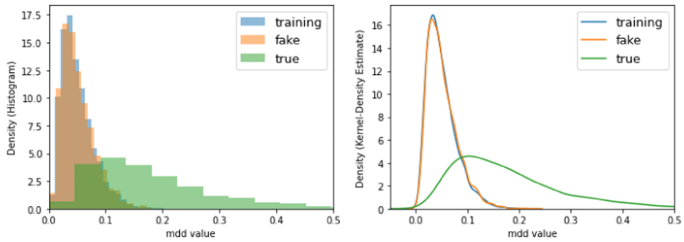

Fig. 5. Maximal draw down distribution, 3-dimensional Heston model with highly correlated training sequences, second dimension.

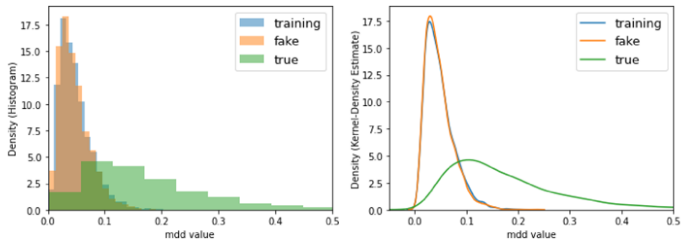

Fig. 6. Maximal draw down distribution, 3-dimensional Heston model with highly correlated training sequences, third dimension.

Table 3. Simulated Correlation of the Three Observed Dimensions at Time Step $i = 15$, Multi-dimensional Heston Model with Highly Correlated Training Sequences

	Training Set	Uncorrelated Set	Sim. mean	Std. dev.
$\text{corr}(S_1, S_2)$	0.28	0.27	0.29	0.008
$\text{corr}(S_1, S_3)$	0.20	0.19	0.17	0.009
$\text{corr}(S_2, S_3)$	0.42	0.42	0.43	0.008

4.3 Multi-dimensional Polynomial Model

In this subsection, we use a synthesized data set generated by a nonlinear SDE-based stochastic process.

4.3.1 Underlying Model for Synthetic Data: Multi-dimensional Polynomial. The underlying stochastic process is formulated by a stochastic differential equation given by:

$$\begin{aligned} &d\,(S_t, Y_t)^\top \\ &= \mu(S_t, Y_t)dt + \Sigma(S_t, Y_t)dW_t \end{aligned} \tag{41}$$

where

$$\begin{aligned} &\mu(S_t, Y_t) \\ &= \left(S_{1,t}^{0.2} + S_{2,t}^{0.2} + 1 \quad S_{2,t}^{0.3} + 0.02 S_{1,t} S_{3,t} \quad S_{3,t}^{0.25} + 0.01 S_{1,t} \quad Y_{2,t} + Y_{3,t} \quad Y_{3,t} + Y_{1,t} \quad Y_{1,t} + Y_{2,t} \right)^\top, \end{aligned}$$

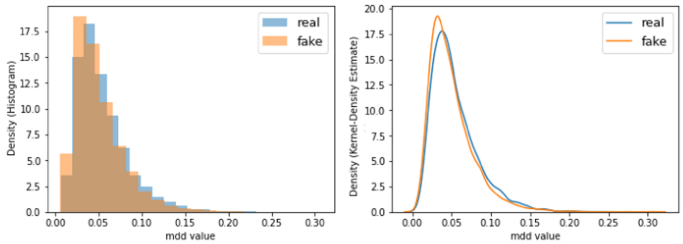

Fig. 7. Maximal draw down distribution, 3-dimensional polynomial model, first dimension.

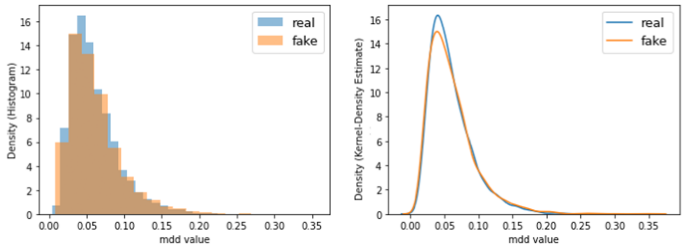

Fig. 8. Maximal draw down distribution, 3-dimensional polynomial model, second dimension.

and

$$\Sigma(S_t, Y_t) = \begin{pmatrix} -2S_{1,t}^{1.2}Y_{1,t} & S_{1,t}S_{2,t}Y_{2,t} & 2S_{1,t}S_{3,t}Y_{3,t} & 0.1S_{1,t}Y_{1,t} & 0.5S_{1,t}Y_{2,t} & 0.7S_{1,t}Y_{3,t} \\ 01.5S_{1,t}S_{2,t}Y_{1,t} & 7S_{2,t}^{1.1}Y_{2,t} & S_{1,t}S_{3,t}Y_{3,t} & 0.1S_{2,t}Y_{1,t} & 0.2S_{2,t}Y_{2,t} & 0.2S_{2,t}Y_{2,t} \\ 3S_{1,t}S_{3,t}Y_{1,t} & S_{2,t}S_{3,t}Y_{2,t} & -Y_{3,t}S_{1,t}S_{3,t}^{1.2} & 0.2S_{3,t}Y_{1,t} & 0.6S_{3,t}Y_{2,t} & 0.3S_{3,t}Y_{3,t} \\ Y_{1,t} & 0 & 0 & Y_{2,t}Y_{3,t} & Y_{3,t}Y_{1,t} & Y_{1,t}Y_{2,t} \\ 0 & Y_{2,t} & 0 & Y_{1,t}Y_{3,t} & Y_{1,t}Y_{2,t} & Y_{3,t}Y_{2,t} \\ 0 & 0 & Y_{3,t} & Y_{2,t}Y_{1,t} & Y_{3,t}Y_{2,t} & Y_{1,t}Y_{3,t} \end{pmatrix}.$$

We next describe how the data set is synthesized. The initial values are given by $S_0 \sim \mathcal{N}((25, 25, 15), 1)$, where all three dimensions are independent, and $Y_0 = (0.1, 0.1, 0.1)$. There are $n = 5{,}000$ sequences generated in total, each having $p = 25$ observed points with frequency $\Delta = 0.01$ as the resolution for observation. We use an Euler discretization of the process, setting 15 sub-intervals between every two observations, which implies that the resolution for generation is set as $\Delta t = 0.00067$. This synthetic data set is then used as input data for the discriminator.

4.3.2 Training Process and Results. We specify the structure of the simulator to have the same form as (39). The parameterization and initialization of the neural networks μ_θ, Σ_ϕ and f_ψ, as well as the iterative optimization process are similar to those of the first experiment. The training takes about 10 minutes.

For evaluation of training, we first illustrate the comparison between maximal drawdown distribution of the three dimensions. The sizes of the synthesized data set and the simulated data set used for comparison are both 5,000. The results are illustrated in the following Figures 7, 8, and 9.

We also investigate the correlation matrix of the three dimensions of data at the 25th observation. By generating 5,000 sequences using the trained networks μ_θ and Σ_ϕ each time to estimate the correlation matrix, and replicating 100 times to derive the mean value and standard deviation of simulated estimations, we have the following results in Table 4. Note that the true value is now estimated from real (synthetic) data.

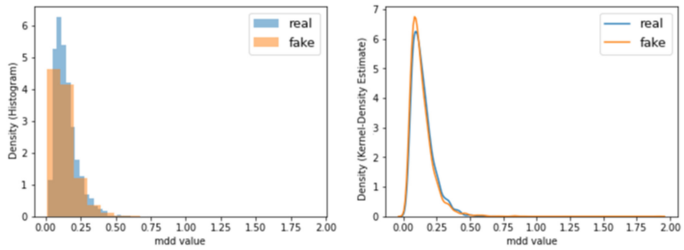

Fig. 9. Maximal draw down distribution, 3-dimensional polynomial model, third dimension.

Table 4. Simulated Correlation of the Three Observed Dimensions at Time Step $i = 15$, Polynomial Model

	Est. true value	Sim. mean	Std. dev.
$\mathrm{cor}(S_1, S_2)$	0.187	0.213	0.014
$\mathrm{cor}(S_1, S_3)$	−0.157	−0.164	0.015
$\mathrm{cor}(S_2, S_3)$	0.442	0.441	0.010

4.4 Multi-dimensional Polynomial Model with Heavy-tailed Noises

In this subsection, we demonstrate the performance of our proposed framework when the distribution of the elementary randomness (i.e., η_k's) are heavy-tailed.

4.4.1 Underlying Model for Synthetic Data: Multi-dimensional Polynomial with Heavy-tailed Randomness. The underlying stochastic process is sequentially generated according to

$$(S_{k+1}, Y_{k+1})^\top = (S_k, Y_k)^\top + \mu(S_k, Y_k)\Delta t + \Sigma(S_k, Y_k)\Delta t \Delta\eta_{k+1}, \tag{42}$$

where $S_k : k = 0, 1, 2, \ldots$ and $Y_k : k = 0, 1, 2, \ldots$ are 3-dimensional vectors; $\mu(S_k, Y_k)$ and $\Sigma(S_k, Y_k)$ have the same form with (41). Specifically, we have

$$\mu(S_k, Y_k) = \left(S_{1,k}^{0.2} + S_{2,k}^{0.2} + 1 \quad S_{2,k}^{0.3} + 0.02 S_{1,k} S_{3,k} \quad S_{3,k}^{0.25} + 0.01 S_{1,k} \quad Y_{2,k} + Y_{3,k} \quad Y_{3,k} + Y_{1,k} \quad Y_{1,k} + Y_{2,k} \right)^\top,$$

and

$$\Sigma(S_k, Y_k) = \begin{pmatrix} -2S_{1,k}^{1.2} Y_{1,k} & S_{1,k} S_{2,k} Y_{2,k} & 2S_{1,k} S_{3,k} Y_{3,k} & 0.1 S_{1,k} Y_{1,k} & 0.5 S_{1,k} Y_{2,k} & 0.7 S_{1,k} Y_{3,k} \\ 01.5 S_{1,k} S_{2,k} Y_{1,k} & 7 S_{2,k}^{1.1} Y_{2,k} & S_{1,k} S_{3,k} Y_{3,k} & 0.1 S_{2,k} Y_{1,k} & 0.2 S_{2,k} Y_{2,k} & 0.2 S_{2,k} Y_{2,k} \\ 3 S_{1,k} S_{3,k} Y_{1,k} & S_{2,k} S_{3,k} Y_{2,k} & -Y_{3,k} S_{1,k} S_{3,k}^{1.2} & 0.2 S_{3,k} Y_{1,k} & 0.6 S_{3,k} Y_{2,k} & 0.3 S_{3,k} Y_{3,k} \\ Y_{1,k} & 0 & 0 & Y_{2,k} Y_{3,k} & Y_{3,k} Y_{1,k} & Y_{1,k} Y_{2,k} \\ 0 & Y_{2,k} & 0 & Y_{1,k} Y_{3,k} & Y_{1,k} Y_{2,k} & Y_{3,k} Y_{2,k} \\ 0 & 0 & Y_{3,k} & Y_{2,k} Y_{1,k} & Y_{3,k} Y_{2,k} & Y_{1,k} Y_{3,k} \end{pmatrix}.$$

$\Delta\eta_k : k = 1, 2, \ldots$ are i.i.d. 6-dimensional vectors. In each $\Delta\eta_k$, the 6 dimensions are i.i.d. variables following t-distribution with 2.5 degrees of freedom.

We next describe how the data set is synthesized. The initial values are given by $S_0 \sim \mathcal{N}((25, 25, 15), 1)$, where all three dimensions are independent, and $Y_0 = (0.1, 0.1, 0.1)$. The resolution for generation is set as $\Delta t = 0.00067$. There are $n = 5{,}000$ sequences generated in total, each having $p = 25$ observed points with frequency $\Delta = 0.01$ as the resolution for observation. Thus,

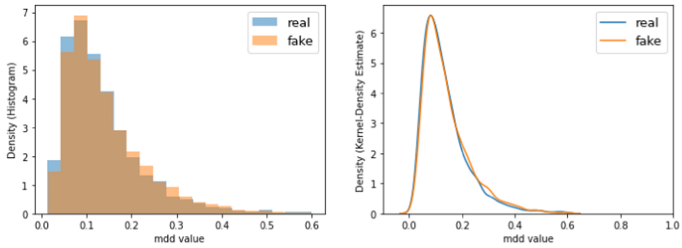

Fig. 10. Maximal draw down distribution, 3-dimensional polynomial model with heavy-tailed noises, first dimension.

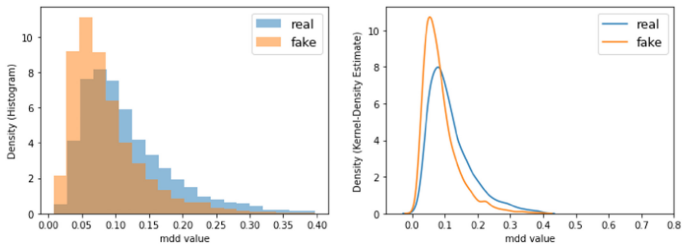

Fig. 11. Maximal draw down distribution, 3-dimensional polynomial model with heavy-tailed noises, second dimension.

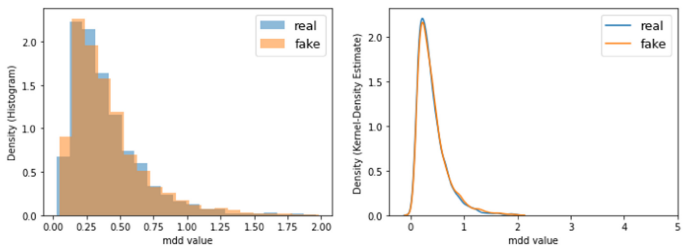

Fig. 12. Maximal draw down distribution, 3-dimensional polynomial model with heavy-tailed noises, third dimension.

there are 15 sub-intervals between every two observations. This synthetic data set is then used as input data for the discriminator.

4.4.2 Training Process and Results. We specify the structure of the simulator to have the same form as (39). The parameterization and initialization of the neural networks μ_θ, Σ_ϕ and f_ψ, as well as the iterative optimization process are similar to those of the first experiment. The training takes about 10 minutes.

For evaluation of training, we first illustrate the comparison between maximal drawdown distribution of the three dimensions. The sizes of the synthesized data set and the simulated data set used for comparison are both 5,000. The results are illustrated in the following Figures 10, 11, and 12.

We also investigate the correlation matrix of the three dimensions of data at the 25th observation. By generating 5,000 sequences using the trained networks μ_θ and Σ_ϕ each time to estimate the correlation matrix, and replicating 100 times to derive the mean value and standard deviation of

Table 5. Simulated Correlation of the Three Observed Dimensions at Time Step $i = 15$, Polynomial Model

	Est. true value	Sim. mean	Std. dev.
$\mathrm{cor}(S_1, S_2)$	0.31	0.29	0.10
$\mathrm{cor}(S_1, S_3)$	−0.24	−0.12	0.14
$\mathrm{cor}(S_2, S_3)$	0.62	0.56	0.15

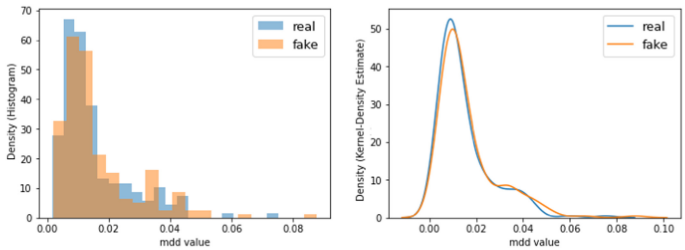

Fig. 13. Maximal draw down distribution, real stock price.

simulated estimations, we have the following results in Table 5. Note that the true value is now estimated from real (synthetic) data.

4.5 Stock Price

4.5.1 The Real Data Set. In this subsection, we use a real data set from a data vendor from a platform *Wind* to test the performance of our estimated simulator. The data set consists of the price variations of a stock (Facebook) from Oct. 8th, 2020 to Mar. 22nd, 2021. The observation frequency is 15 minutes, and 26 data points $(S_i : i = 0, 1, 2, \ldots, 25)$ are recorded for every transaction day. The empirical data is processed as follows. Since stock returns are usually stationary while prices are not, we take logarithm of the data points $(S_i : i = 0, 1, 2, \ldots, 25)$, and derive the log return sequence $(R_i : i = 1, 2, \ldots, 25)$, where $R_i = \log S_i - \log S_{i-1}$. With such transformation, all log return sequences can be regarded as weakly correlated identical copies of an underlying real distribution. After removing the sequences with missing data, we retain $n = 186$ such copies.

4.5.2 Training Process and Results. We specify the structure of the simulator as

$$X_{k+1} = X_k + \mu_\theta(X_k)\Delta t + \Sigma_\phi(X_k)\sqrt{\Delta t}\eta_{k+1} \tag{43}$$

where $X_k = (R_k, Y_k)^\top$, $\eta_k : k = 1, 2, \ldots$ are independent 2-dimensional canonical normal variables. The latent volatility process Y_k is assumed to be 1-dimensional. We first simulate a sequence of $\log \hat{S}_i$ using the sequential simulator, and derive the log return sequence as part of the input of the discriminator. Thus, the discriminator compares distributions of log returns. The resolution is set as the daily frequency with $\Delta t = 1/252$ year. The parameterization of μ_θ is given by $L = 2$, $\tilde{n} = (n_1, n_2) = (50, 2)$. The parameterization of Σ_ϕ is given by $L = 2$, $\tilde{n} = (n_1, n_2) = (80, 4)$. The parameterization of f_ψ is given by $L = 3$, $\tilde{n} = (n_1, n_2, n_3) = (200, 200, 1)$. The neural network initialization of μ_θ, Σ_ϕ and f_ψ, as well as the iterative optimization process are similar to those of the first experiment. The training takes about 1.5 minutes.

We next evaluate the training results. Since the stock price is 1-dimensional, we take itself as the portfolio, i.e., $H_t = S_t$. The comparison between the distribution of the maximal drawdown of real data-based H_t and that of the simulated data-based $\hat{H}_t$ is illustrated in the following Figure 13. The sizes of the real data set and the simulated data set used for comparison are both 186.

5 CONCLUSION

We propose a new framework of a sequential-structured simulator assisted by neural networks and Wasserstein training to model, estimate, and simulate a wide class of sequentially generated data. Neural networks are integrated into the sequentially structured simulators to capture potential nonlinear and complicated sequential structures. Given representative real data, the neural network parameters in the simulator are estimated and calibrated through a Wasserstein training process, which matches the joint distribution of the simulated data and real data in terms of Wasserstein distance. Moreover, the neural network-assisted sequential structured simulator can flexibly incorporate various kinds of elementary randomness and generate distributions with certain properties such as heavy-tail, without the need to redesign the estimation and training procedures. Further, regarding statistical properties, we provide results on consistency and convergence rate for the estimation procedure of the proposed simulator, which are the first set of results that allow the training data samples to be correlated.

ACKNOWLEDGMENTS

The authors are thankful to the anonymous reviewers and editors for their very helpful comments and suggestions, which have significantly benefited this work. The authors thank the participants and organizers of 2021 INFORMS Simulation Society Workshop (I-Sim) for helpful comments and feedback. A preliminary conference version of this work, [33], has appeared in the Proceedings of the Winter Simulation Conference 2021. The theory and analysis in this manuscript are new, compared to the conference version.

APPENDICES

A BACKPROPOGATION DIAGRAM ON GRADIENT EVALUATION

The backpropagation diagram (red dashed line) on the gradient evaluation of $f_\psi(\hat{S}(\theta,\phi))$ with respect to the parameters θ,ϕ in the simulator neural network μ_θ,Σ_ϕ is shown in Figure A.1.

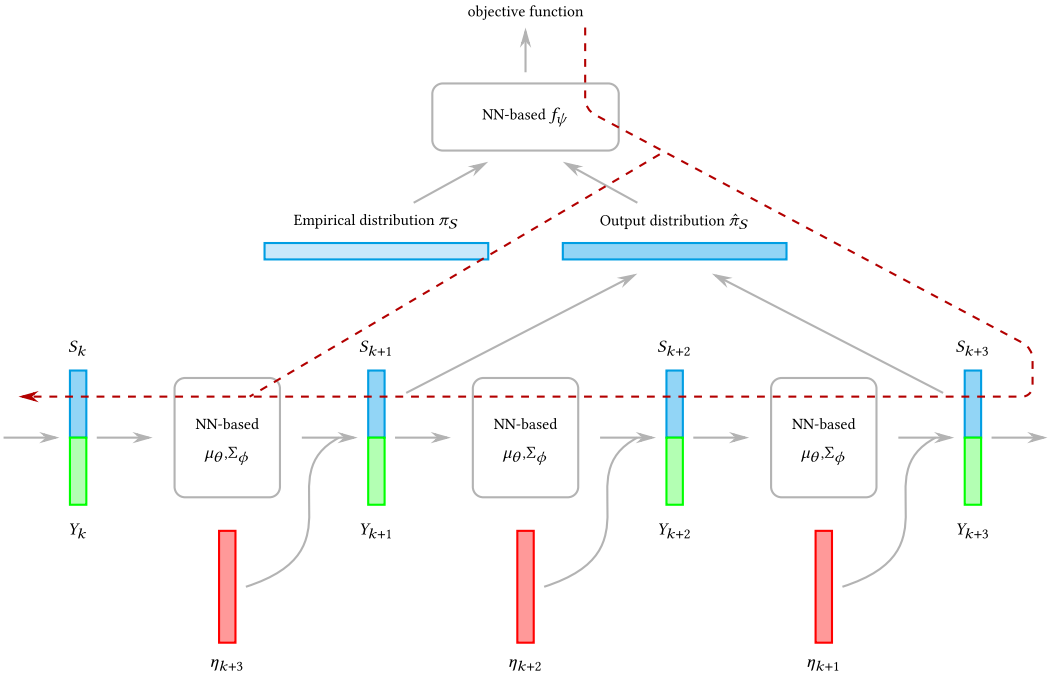

Fig. A.1. Backpropagation diagram (red dashed line) on the gradient evaluation of $f_\psi(\hat{S}(\theta,\phi))$ with respect to the parameters θ,ϕ in the simulator neural network μ_θ,Σ_ϕ.

B PROOF OF THEOREM 3.4

In the following subsections, we present a proof of Theorem 3.4 consisting of two main steps: Approximating Lipschitz functions with interpolation polynomials and approximating the polynomials with neural networks.

B.1 Polynomial Approximation of Lipschitz Functions

We first perform a linear transformation on f, namely, let

$$\Psi : [-B,B]^d \to [0,1]^d \quad \Psi(\boldsymbol{z}) = \frac{1}{2B}(\boldsymbol{z} + B \cdot \mathbf{1}). \tag{44}$$

Let $f^\Psi = f \circ \Psi^{-1}$, we have

$$f^\Psi \in [0,1]^d, \quad \|f^\Psi(\boldsymbol{x}) - f^\Psi(\boldsymbol{y})\| \le 2BC_L\|\boldsymbol{x}-\boldsymbol{y}\|. \tag{45}$$

Without loss of generality, suppose that $\|f^\Psi\|_{L^\infty([0,1]^d)} \le BC_L$. Note that if the $2BC_L$-Lipschitz function f^Ψ can be approximated by a neural network Φ in the sense that $\|\Phi(\boldsymbol{x}) - f^\Psi(\boldsymbol{x})\| \le \delta, \forall \boldsymbol{x} \in [0,1]^d$, we have

- $\Phi \circ \Psi$ can also be expressed in the form of a neural network, where $\mathcal{L}(\Phi \circ \Psi) = \mathcal{L}(\Phi)$
- $\|\Phi \circ \Psi(\boldsymbol{x}) - f^\Psi \circ \Psi(\boldsymbol{x})\| = \|\Phi \circ \Psi(\boldsymbol{x}) - f(\boldsymbol{x})\| \le \delta, \forall \boldsymbol{x} \in [-B,B]^d$

The next step is to approximate an arbitrary $2BC_L$-Lipschitz function f^Ψ on $[0,1]^d$ with a explicitly constructed interpolation polynomial. Our construction and proof are a simplified version of those in [11].

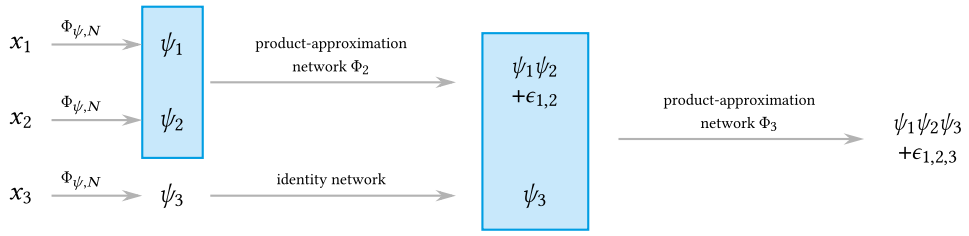

Fig. B.1. Construction of $\Phi_{\zeta_{\boldsymbol{m}}}$.

Define the trapezoid function

$$\psi(x) = \begin{cases} 1 & |x| < 1, \\ 2 - |x| & 1 \le |x| \le 2, \\ 0 & |x| > 2, \end{cases} \qquad x \in \mathbb{R}, \tag{46}$$

and let

$$\zeta_{N,\boldsymbol{m}}(\boldsymbol{x}) = \prod_{k=1}^{d} \psi\left(3N(x_k - \frac{m_k}{N})\right) := \prod_{k=1}^{d} \psi_N(x_k), \tag{47}$$

where $\boldsymbol{m} = (m_1, m_2, \ldots, m_d)$, $m_i \in \{0, 1, \ldots, N\}$. Note that $0 \le \zeta_{N,\boldsymbol{m}}(\boldsymbol{x}) \le 1$ and $\sum_{\boldsymbol{m}} \zeta_{N,\boldsymbol{m}}(\boldsymbol{x}) = 1$, $\forall \boldsymbol{x} \in [0,1]^d$. Further, let

$$P_{N,\boldsymbol{m}} = f^{\Psi}\left(\frac{1}{N} \cdot \boldsymbol{m}\right), \quad \bar{f}_N = \sum_{\boldsymbol{m}} P_{N,\boldsymbol{m}} \zeta_{N,\boldsymbol{m}}(\boldsymbol{x}). \tag{48}$$

We show that $\|\bar{f}_N - f^{\Psi}\|_\infty$ can be bounded as follows:

$$\begin{aligned} \max_{\boldsymbol{x}\in[0,1]^d} |\bar{f}_N(\boldsymbol{x}) - f^{\Psi}(\boldsymbol{x})| &= \max_{\boldsymbol{x}\in[0,1]^d} \left| \sum_{\boldsymbol{m}} \zeta_{N,\boldsymbol{m}}(\boldsymbol{x})(P_{N,\boldsymbol{m}} - f^{\Psi}(\boldsymbol{x})) \right| \\ &\overset{(i)}{\le} \max_{\boldsymbol{x}\in[0,1]^d} \sum_{\boldsymbol{m}:\left|x_k - \frac{m_k}{N}\right| < \frac{1}{N}} \left| P_{N,\boldsymbol{m}} - f^{\Psi}(\boldsymbol{x}) \right| \\ &\le \max_{\boldsymbol{x}\in[0,1]^d} 2^d \max_{\boldsymbol{m}:\left|x_k - \frac{m_k}{N}\right| < \frac{1}{N}} \left| f^{\Psi}\left(\frac{1}{N} \cdot \boldsymbol{m}\right) - f^{\Psi}(\boldsymbol{x}) \right| \\ &\le \max_{\boldsymbol{x}\in[0,1]^d} 2^d \max_{\boldsymbol{m}:\left|x_k - \frac{m_k}{N}\right| < \frac{1}{N}} 2BC_L \left\| \frac{1}{N} \cdot \boldsymbol{m} - \boldsymbol{x} \right\| \\ &\le \frac{2^{d+1}\sqrt{d}BC_L}{N}. \end{aligned} \tag{49}$$

Note that

$$\zeta_{N,\boldsymbol{m}}(\boldsymbol{x}) = 0 \quad \text{if } \exists k \in \{1, 2, \ldots, d\}, \text{ s.t. } \left| x_k - \frac{m_k}{N} \right| \ge \frac{1}{N}, \tag{50}$$

which is due to the definition (47) of $\zeta_{N,\boldsymbol{m}}(\boldsymbol{x})$ and is the reason for (i). Therefore, for $N \ge 2^{d+2}\sqrt{d}BC_L/\delta$, we have $\|\bar{f}_N - f^{\Psi}\|_\infty \le \delta/2$.

B.2 Neural Network Approximation of Polynomials

In this section, we use neural networks to approximate $\bar{f}_N$ constructed in Section B.1. We first introduce the following Theorem B.1 which constructs a neural network to approximate multiplication of two constants, i.e., $\Phi(x, y) \approx xy$.

Theorem B.1 (Neural Network Approximation of the Product Operator, Proposition 3.3 of [15]). *There exists a constant $C > 0$ such that, for all $D \in \mathbb{R}^+$ and $\epsilon \in (0, 1/2)$, there is a network $\Phi \in \mathcal{N}_{2,1}$, with $\mathcal{L}(\Phi) \le C(\log(\lceil D \rceil) + \log(\epsilon^{-1}))$, $\mathcal{W}(\Phi) \le 5$, $\mathcal{B}(\Phi) \le 1$, $\mathcal{M}(\Phi) = O(\mathcal{L}(\Phi))$, $\Phi(0, x) = \Phi(x, 0) = 0$, for all $x \in \mathbb{R}$, and*

$$\|\Phi(x, y) - xy\|_{L^\infty([D,D])} \le \epsilon. \tag{51}$$

Now, using Theorem B.1 as a building block, we approximate $\zeta_{N,\boldsymbol{m}}(\boldsymbol{x})$ defined in Section B.1, which is the product of $\psi_N(x_k) : k = 1, 2, \ldots, d$. One useful way to analyze such approximation is to view the neural network as not just a composition of linear and activation functions, but also a combination of layers with operational connections between the elements of every two adjacent layers.

First, the mapping $x_k \to \psi_N(x_k)$ can be expressed by a neural network. Consider the hat function $h : \mathbb{R} \to [0, 1]$,

$$h(x) = \begin{cases} 2x, & \text{if } 0 \le x \le \frac{1}{2}, \\ 2(1-x), & \text{if } \frac{1}{2} \le x \le 1, \\ 0, & \text{else}, \end{cases} \tag{52}$$

we have

(1) According to [15], $h(x)$ can be expressed by a neural network $\Phi_h(x)$, where $\mathcal{L}(\Phi_h) = 2$, $\mathcal{W}(\Phi_h) = 3$, $\mathcal{B}(\Phi_h) = 4$ and $\mathcal{M}(\Phi_h) = 8$. Also note that the 2-Layer network is given as $W_1 \circ \sigma \circ W_2$, and if a ReLU activation is not involved in the middle, any composition of linear transformations can be compressed into a single layer.

(2) $\psi(x)$ given as (46) can be expressed as follows:

$$\psi(x) = h\left(\frac{1}{2}x\right) + h\left(\frac{1}{2}(x+1)\right) + h\left(\frac{1}{2}(x+2)\right). \tag{53}$$

Therefore, $\psi_N(x_k)$ can be expressed by a neural network $\Phi_{\psi,N}(x_k) \in \mathcal{N}_{1,1}$, where $\mathcal{L}(\Phi)_{\psi,N} = 2$, $\mathcal{W}(\Phi_{\psi,N}) = 9$, and $\mathcal{B}(\Phi_{\psi,N}) = O(N)$. We denote by $\mathcal{N}_{d_1,d_2}$ the set of all ReLU networks with input dimension d_1 and output dimension d_2. Note that N will be balanced with ϵ and δ later on.

The next step is to iteratively approximate the product $\prod_{k=1}^d \psi_k$ with neural network approximators of multiplication. Observe that $\psi_k, k = 1, 2, \ldots, d$ and their products are all bounded by $[0, 1]$. Specifically, by Theorem B.1, we have a network $\Phi_2 \in \mathcal{N}_{2,1}$, with $\mathcal{L}(\Phi_2) = O(\log(d\epsilon^{-1}))$, $\mathcal{W}(\Phi_2) \le 5$, $\mathcal{B}(\Phi_2) \le 1$ and $\mathcal{M}(\Phi_2) = O(\mathcal{L}(\Phi_2))$ satisfying

$$\|\Phi_2(\psi_1, \psi_2) - \psi_1\psi_2\|_{L^\infty([0,1])} \le \frac{\epsilon}{d}. \tag{54}$$

Iteratively, we have a network $\Phi_3 \in \mathcal{N}_{2,1}$, with $\mathcal{L}(\Phi_3) = O(\log(d\epsilon^{-1}))$, $\mathcal{W}(\Phi_3) \le 5$, $\mathcal{B}(\Phi_3) \le 1$ and $\mathcal{M}(\Phi_3) = O(\mathcal{L}(\Phi_3))$ satisfying

$$\begin{aligned} &\|\Phi_3(\Phi_2(\psi_1, \psi_2), \phi_3) - \psi_1\psi_2\psi_3\|_{L^\infty([0,1])} \\ \le &\|\Phi_3(\Phi_2(\psi_1, \psi_2), \psi_3) - \Psi_2(\psi_1, \psi_2)\psi_3\|_{L^\infty([0,1])} + \|\Phi_2(\psi_1, \psi_2)\psi_3 - \psi_1\psi_2\psi_3\|_{L^\infty([0,1])} \le \frac{2\epsilon}{d} + O(\epsilon^2). \end{aligned} \tag{55}$$

In total, we have a network $\Phi_{\zeta_{N,\boldsymbol{m}}} \in \mathcal{N}_{d,1}$ composite of $\Phi_i, i = 2, 3, \ldots d$ and $\Phi_{\psi,N}$s, with $\mathcal{L}(\Phi_{\zeta_{N,\boldsymbol{m}}}) = O(d \log(d\epsilon^{-1}))$, $\mathcal{W}(\Phi_{\zeta_{N,\boldsymbol{m}}}) = O(d)$, $\mathcal{B}(\Phi_{\zeta_{N,\boldsymbol{m}}}) = O(N)$ and $\mathcal{M}(\Phi_{\zeta_{N,\boldsymbol{m}}}) = O(\mathcal{L}(\Phi_{\zeta_{N,\boldsymbol{m}}})$ $\mathcal{W}(\Phi_{\zeta_{N,\boldsymbol{m}}}))$ where

$$\|\Phi_{\zeta_{N,\boldsymbol{m}}}(\boldsymbol{x}) - \zeta_{N,\boldsymbol{m}}(\boldsymbol{x})\|_{L^\infty([0,1]^d)} \le 2\epsilon. \tag{56}$$

The following Figure B.1 illustrates how $\Phi_{\zeta_{N,\boldsymbol{m}}}$ is constructed through combining and paralleling small networks.

Finally, the network $\Phi_{\bar{f}_N} = \sum_{\boldsymbol{m}} P_{N,\boldsymbol{m}} \Phi_{\zeta_{N,\boldsymbol{m}}}$ with $\mathcal{L}(\Phi_{\bar{f}_N}) = O(d\log(d\epsilon^{-1}))$, $\mathcal{W}(\Phi_{\bar{f}_N}) = O(d(N+1)^d)$, $\mathcal{B}(\Phi_{\bar{f}_N}) = O(N + BC_L)$ and $\mathcal{M}(\Phi_{\bar{f}_N}) + O(\mathcal{L}(\Phi_{\bar{f}})\mathcal{W}(\Phi_{\bar{f}_N}))$ satisfies

$$\|\Phi_{\bar{f}_N}(\boldsymbol{x}) - \bar{f}_N(\boldsymbol{x})\|_{L^\infty([0,1]^d)} \le 2^{d+1} BC_L \epsilon. \tag{57}$$

Note that $\forall \boldsymbol{x} \in [0,1]^d$, only 2^d out of the $(N+1)^d$ elements of $\{\zeta_{N,\boldsymbol{m}}\}, \forall \boldsymbol{m}$ are non-zero, and according to Theorem B.1, the approximation error is exactly 0 when zero elements are contained in the multipliers. This explains the term 2^{d+1} on R.H.S. of (57). The term BC_L on R.H.S. of (57) comes from the fact that $|P_{N,\boldsymbol{m}}| = |f^\Psi(\frac{1}{N} \cdot M)| \le BC_L$.

B.3 Balance and Conclusion

To have $\|\bar{f}_N - f^\Psi\|_\infty$ for Section B.1 and $\|\Phi_{\bar{f}_N}(\boldsymbol{x}) - \bar{f}_N(\boldsymbol{x})\|_{L^\infty([0,1]^d)} \le 2^{d+1} BC_L \epsilon \le \frac{\delta}{2}$ for Section B.2, let

$$N = 2^{d+2}\sqrt{d} BC_L \frac{1}{\delta}, \quad \epsilon = \frac{\delta}{2^{d+2} BC_L}, \tag{58}$$

we can replace the orders of the network approximator Φ_f with $\mathcal{L}(\Phi_f) = O(\log B + \log \delta^{-1})$, $\mathcal{W}(\Phi_f) = O(B^d \delta^{-d})$, $\mathcal{B}(\Phi_f) = O(B\delta^{-1})$ and $\mathcal{M}(\Phi_f) = O((\log B + \log \delta^{-1}) \cdot B^d \delta^{-d})$, and Φ_f satisfies

$$\|\Phi_f(\boldsymbol{x}) - f(\boldsymbol{x})\|_{L^\infty([-B,B]^d)} \le \|\bar{f}_N - f^\Psi\|_\infty + \|\Phi_{\bar{f}_N}(\boldsymbol{x}) - \bar{f}_N(\boldsymbol{x})\|_{L^\infty([0,1]^d)} \le \delta. \tag{59}$$

C CONTROLLING BOUNDING ERROR AND GENERATOR APPROXIMATION ERROR

C.1 Bounding Error

In this section, we control the bounding error term $W(\pi, \pi^B)$. With the assumption that

$$P_{\eta_k}(x) \le O(x^{-(\alpha+2)}), \quad \alpha > 0, \tag{60}$$

the bounding error term can be bounded as follows:

$$W(\pi, \pi^B) \le W(\pi, \pi^B) = \inf_{\gamma \in \Gamma(\pi^B, \pi)} \mathbb{E}_\gamma \left(\|\mathbf{X} - \mathbf{Y}\|\right) \le \mathbb{E}\left(\|\mathbf{X} - \mathbf{X}^B\|\right) \le O(B^{-\alpha}). \tag{61}$$

C.2 Generator Approximation Error

In this section, we control the generator approximation error $W(\pi^B, \hat{\pi}^B(\mu_\theta, \Sigma_\phi))$. Note that

$$W(\pi^B, \hat{\pi}^B(\mu_\theta, \Sigma_\phi)) \le \underbrace{W(\hat{\pi}^B(\mu, \Sigma), \hat{\pi}^B(\mu_\theta, \Sigma_\phi))}_{\text{identically bounded generator approximation error}} + \underbrace{W(\pi, \hat{\pi}^B(\mu, \Sigma)) + W(\pi, \pi^B)}_{\text{bounding error}}. \tag{62}$$

As in Section C.1, the bounding error can be controlled by

$$W(\pi, \hat{\pi}^B(\mu, \Sigma)) + W(\pi, \pi^B) \le O(B^{-\alpha}). \tag{63}$$

Next, denote the network approximation error of μ and Σ on $[-B, B]^d$ as ϵ_μ and ϵ_Σ, respectively. We have

$$\begin{aligned} W(\hat{\pi}^B(\mu, \Sigma), \hat{\pi}^B(\mu_\theta, \Sigma_\phi)) &= \inf_{\gamma \in S(\hat{\pi}^B(\mu_\theta, \Sigma_\phi), \hat{\pi}^B(\mu, \Sigma))} \mathbb{E}_{(\hat{\mathbf{X}}^B, \hat{\mathbf{Y}}^B) \sim \gamma} \left[\|\hat{\mathbf{X}}^B - \hat{\mathbf{Y}}^B\|\right] \\ &\le \mathbb{E}_{(\hat{\mathbf{X}}^B, \hat{\mathbf{Y}}^B) \sim \gamma_e} \left[\|\hat{\mathbf{X}}^B - \hat{\mathbf{Y}}^B\|\right], \end{aligned} \tag{64}$$

where, denoting the elementary randomness of the simulator of $\hat{\boldsymbol{X}}^B$ and $\hat{\boldsymbol{Y}}^B$ as $\{\xi_i\}_{i=1}^p$ and $\{\eta_i\}_{i=1}^p$ respectively, we have

$$\mathbb{E}_{\gamma_e} \left[\|\hat{\mathbf{X}}^B - \hat{\mathbf{Y}}^B\|\right] := \mathbb{E}\left[\|\hat{\mathbf{X}}^B - \hat{\mathbf{Y}}^B\| \middle| \xi_i \equiv \eta_i, \forall i\right]. \tag{65}$$

Since

$$\mathbb{E}_{\gamma_e}\left[\|\hat{\mathbf{X}}^B - \hat{\mathbf{Y}}^B\|\right] \le \sum_{k=1}^{p} \mathbb{E}_{\gamma_e}\left[\|\hat{X}_i^B - \hat{Y}_i^B\|\right] := \sum_{i=1}^{p} \mathbb{E}_{\gamma_e}\Delta_i, \tag{66}$$

and by law of total expectation, we have

$$\mathbb{E}_{\gamma_e}\Delta_i = \mathbb{E}_{\gamma_e}\left[\mathbb{E}_{\gamma_e}(\Delta_i|\Delta_{i-1})\right], \tag{67}$$

further,

$$\begin{aligned}
&\mathbb{E}_{\gamma_e}\left[\Delta_i \middle| \Delta_{i-1}, \hat{X}_{i-1}^B, \hat{Y}_{i-1}^B\right] \\
\le&\mathbb{E}_{\gamma_e}\left[\Delta_{i-1} + |\mu(\hat{Y}_{i-1}^B) - \mu_\theta(\hat{X}_{i-1}^B)| + \|\Sigma(\hat{Y}_{i-1}^B) - \Sigma_\phi(\hat{X}_{i-1}^B)\eta_i\| \middle| \Delta_{i-1}, \hat{X}_{i-1}^B, \hat{Y}_{i-1}^B\right] \\
\le&\mathbb{E}_{\gamma_e}\left[\Delta_{i-1} + (\epsilon_\mu + \Delta_{i-1}) + (\epsilon_\Sigma + \Delta_{i-1})\|\eta_i\| \middle| \Delta_{i-1}\right] \\
\le&(2+d)\Delta_{i-1} + (\epsilon_\mu + d\epsilon_\Sigma).
\end{aligned} \tag{68}$$

Therefore,

$$\mathbb{E}_{\gamma_e}\Delta_i \le \frac{(2+d)^i - 1}{1+d}(d\epsilon_\Sigma + \epsilon_\mu), \tag{69}$$

and

$$\sum_{i=1}^{p} \mathbb{E}_{\gamma_e}\Delta_i \le \left(\frac{1}{1+d}\sum_{i=1}^{p}(2+d)^i\right) \cdot (d\epsilon_\Sigma + \epsilon_\mu) = O(\epsilon_\Sigma + \epsilon_\mu). \tag{70}$$

D PROOF OF LEMMA D.1

In this section, we first prove that the two function classes $\mathcal{F}_{NN}(\kappa, L, P, K, \epsilon_f)$ and $\mathcal{F}_{\text{Lip}}$ can approximate each other, namely,

(1) $\forall f \in \mathcal{F}_{\text{Lip}}, \exists f_\psi \in \mathcal{F}_{NN}$, such that $\|f - f_\psi\|_{L^\infty[-B,B]^d} \le \epsilon_f$,
(2) $\forall f \in \mathcal{F}_{NN}, \exists f_\psi \in \mathcal{F}_{\text{Lip}}$, such that $\|f - f_\psi\|_{L^\infty[-B,B]^d} \le 3\epsilon_f$.

For 2, we have the following lemma:

Lemma D.1. *Suppose that $f : \mathbb{R}^d \to \mathbb{R}$, $f \in C([a,b])$ satisfies $|f(x) - f(y)| \le \|x - y\| + 2\epsilon$, $\forall x, y \in [a,b]$. Then there is a 1-Lipschitz function $g : \mathbb{R}^d \to \mathbb{R}$, $g \in C([a,b])$, such that*

$$|f(x) - g(x)| \le 3\epsilon, \quad \forall x \in [a,b]. \tag{71}$$

Proof: Without loss of generality, we assume that $d = 1, a = 0, b = 1$ and $f(0) = 0$. We prove by contradiction. Let

$$x(g) := \sup\{x \in [0,1] : |f(x') - g(x')| \le 2\epsilon, \forall x' < x\}, \tag{72}$$

and

$$g^* = \arg\sup_{\|g\|_L \le 1} x(g). \tag{73}$$

The contradiction assumption suggests that $x^* := x(g^*) < 1$. We first show that $x^* > 0$ and that g^* exists. Note that for $x' < \epsilon$, we have, by definition of the functional class $\mathcal{F}_{\text{NN}}(\kappa, L, P, K, \epsilon)$,

$$|f(x') - f(0)| \le |x'| + \epsilon < 3\epsilon.$$

Therefore, taking $g(x) \equiv f(0)$ yields an approximation of f with precision of 3ϵ on $[0, \epsilon]$, implying that x^* is at least ϵ. To demonstrate the existence of g^*, suppose that $|f(x)| < C$ on $[0,1]$ for some constant C. The 1-Lipschitz functional class bounded by $2C$ on $[0,1]$, denoted as $\mathcal{F}_{\text{Lip}}^C$ is uniformly bounded and equicontinuous. Arzelà-Ascoli theorem suggests that $\mathcal{F}_{\text{Lip}}^C$ is sequentially compact.

Further, for any convergent sequence in $\mathcal{F}_{\text{Lip}}^C$, it can be verified that the limit also lies in $\mathcal{F}_{\text{Lip}}^C$. Therefore, $\mathcal{F}_{\text{Lip}}^C$ is a compact set. It remains to be proved that $x(g)$ is upper semi-continuous on $\mathcal{F}_{\text{Lip}}$, so that the supremum point g^* exists. In fact, for a fixed element $g_0 \in \mathcal{F}_{\text{Lip}}$, for all small enough $\epsilon > 0$, let

$$\delta = \sup_{x(g_0) \le x \le x(g_0)+\epsilon} \left[|f(x) - g_0(x)| - 3\epsilon \right].$$

By definition of $x(g)$, we have $\delta > 0$, and for all $g \in \mathcal{F}_{\text{Lip}}$ such that $\|g - g_0\|_{L^\infty[0,B]^d} < \delta$, we have $x(g) < x(g_0) + \epsilon$. The existence of g^* is proved.

We next return to the contradiction assumption, which suggests that $x^* < 1$. Note that both f and g are continuous. Therefore, by the definition of sup, we have

$$|g^*(x^*) - f(x^*)| = 3\epsilon. \tag{74}$$

Without loss of generality, let $f(x^*) = g^*(x^*) + 3\epsilon$. Also,

$$\forall \Delta > 0, \exists x^* < x_\Delta < \min\{x^* + \delta, 1\}, \text{ s.t. } f(x_\Delta) > g^*(x^*) + (x_\Delta - x^*) + 3\epsilon. \tag{75}$$

Now, we claim that $\exists \delta_0 > 0$,

$$\frac{g^*(x^*) - g^*(x)}{x^* - x} = 1, \quad \forall x \in [x^* - \delta_0, x^*).$$

If this is contradicted, we have some $\delta < \epsilon$, and

$$g^*(x^*) - \delta < g^*(x^* - \delta) < g^*(x^*) + 3\epsilon - \delta.$$

In this case, we can move g^* upward on $[x^* - \delta, x^*]$ by modifying it into

$$\tilde{g}^*(x) = \begin{cases} g^*(x), & x \in [0, x^* - \delta]; \\ g^*(x^* - \delta) + x - (x^* - \delta), & x \in (x^* - \delta, x^*], \end{cases}$$

so that $\tilde{g}^*(x^*) > g^*(x^*)$, and $\|\tilde{g}^*(x^*)\|_L \le 1$ still holds. Also, note that $\forall x \in [0,1]$,

$$\begin{aligned} f(x) &\ge f(x^*) + x - x^* - 3\epsilon = g^*(x^*) + x - x^* > \tilde{g}^*(x) - 3\epsilon, \\ f(x) &\le g^*(x) + 3\epsilon \le \tilde{g}^*(x) + 3\epsilon_f. \end{aligned}$$

so $x(\tilde{g}^*) > x(g^*)$, which contradicts (73). Therefore, the claim is valid.

Let

$$x_1 = \inf_{x \in [0, x^* - \delta_0]} \frac{g^*(x^*) - g^*(x)}{x^* - x} = 1$$

Note that $f(x_\Delta) > g^*(x_1) + (x_\Delta - x_1) + 3\epsilon$, therefore, $f(x_1) > g^*(x_1)$, which implies that $x_1 > 0$.

In this case, we can find $\delta' > 0$, such that

$$g^*(x_1) - \delta' \le g^*(x_1 - \delta') \le g^*(x_1) + 3\epsilon - \delta'.$$

We can similarly define

$$\tilde{g}^*(x) = \begin{cases} g^*(x), & x \in [0, x_1 - \delta']; \\ g^*(x_1 - \delta') + x - (x_1 - \delta'), & x \in (x_1 - \delta', x^*], \end{cases}$$

and verify that $\|\tilde{g}^*(x^*)\|_L \le 1$ and $x(\tilde{g}^*) > x(g^*)$, which also contradicts (73). The proof is complete.

REFERENCES

[1] Yacine Aı, Robert Kimmel, et al. 2007. Maximum likelihood estimation of stochastic volatility models. *Journal of Financial Economics* 83, 2 (2007), 413–452.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*. PMLR, 214–223.

[3] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. 2017. Generalization and equilibrium in generative adversarial nets (GANs). In *International Conference on Machine Learning*. PMLR, 224–232.

[4] Manabu Asai, Michael McAleer, and Jun Yu. 2006. Multivariate stochastic volatility: A review. *Econometric Reviews* 25, 2-3 (2006), 145–175.

[5] Yu Bai, Tengyu Ma, and Andrej Risteski. 2018. Approximability of discriminators implies diversity in GANs. *arXiv preprint arXiv:1806.10586* (2018).

[6] Ravi Bansal, A. Ronald Gallant, Robert Hussey, and George Tauchen. 1994. Computational aspects of nonparametric simulation estimation. In *Computational Techniques for Econometrics and Economic Analysis*. Springer, 3–22.

[7] Eoin Brophy, Zhengwei Wang, Qi She, and Tomas Ward. 2021. Generative adversarial networks in time series: A survey and taxonomy. *arXiv preprint arXiv:2107.11098* (2021).

[8] Carmen Broto and Esther Ruiz. 2004. Estimation methods for stochastic volatility models: A survey. *Journal of Economic Surveys* 18, 5 (2004), 613–649.

[9] Wang Cen, Emily A. Herbert, and Peter J. Haas. 2020. NIM: Modeling and generation of simulation inputs via generative neural networks. In *Proceedings of the 2020 Winter Simulation Conference,* Bae, K., Feng, B., Kim, S., Lazarova-Molnar, S., Zheng, Z., Roeder, T., and Thiesing, R. (Ed.). Institute of Electrical and Electronic Engineers, Inc., Piscataway, New Jersey, 584–595.

[10] Alexei Chekhlov, Stanislav Uryasev, and Michael Zabarankin. 2005. Drawdown measure in portfolio optimization. *International Journal of Theoretical and Applied Finance* 8, 01 (2005), 13–58.

[11] Minshuo Chen, Haoming Jiang, Wenjing Liao, and Tuo Zhao. 2022. Nonparametric regression on low-dimensional manifolds using deep ReLU networks: Function approximation and statistical recovery. *Information and Inference: A Journal of the IMA* 11, 4 (2022), 1203–1253.

[12] Minshuo Chen, Wenjing Liao, Hongyuan Zha, and Tuo Zhao. 2020. Statistical guarantees of generative adversarial networks for distribution estimation. *arXiv preprint arXiv:2002.03938* (2020).

[13] Marco Cuturi. 2013. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in Neural Information Processing Systems* 26 (2013), 2292–2300.

[14] Florian Eckerli. 2021. Generative adversarial networks in finance: An overview. *Available at SSRN 3864965* (2021).

[15] Dennis Elbrächter, Dmytro Perekrestenko, Philipp Grohs, and Helmut Bölcskei. 2021. Deep neural network approximation theory. *IEEE Transactions on Information Theory* 67, 5 (2021), 2581–2623.

[16] Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. 2017. Real-valued (medical) time series generation with recurrent conditional GANs. *arXiv preprint arXiv:1706.02633* (2017).

[17] Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. 2016. Sequential neural models with stochastic layers. *Advances in Neural Information Processing Systems* 29 (2016).

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

[19] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C. Courville. 2017. Improved training of Wasserstein GANs. *Advances in Neural Information Processing Systems* 30 (2017).

[20] Linyun He and Eunhye Song. 2021. Nonparametric Kullback-Liebler divergence estimation using M-spacing. In *2021 Winter Simulation Conference (WSC)*. IEEE.

[21] Steven L. Heston. 1993. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The Review of Financial Studies* 6, 2 (1993), 327–343.

[22] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[23] Diederik P. Kingma and Max Welling. 2013. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114* (2013).

[24] Rui Luo, Weinan Zhang, Xiaojun Xu, and Jun Wang. 2018. A neural stochastic volatility model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[25] Angelo Melino and Stuart M. Turnbull. 1990. Pricing foreign currency options with stochastic volatility. *Journal of Econometrics* 45, 1-2 (1990), 239–265.

[26] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. *Foundations of Machine Learning*. MIT Press.

[27] Eckhard Platen and Nicola Bruti-Liberati. 2010. *Numerical Solution of Stochastic Differential Equations with Jumps in Finance*. Vol. 64. Springer Science & Business Media.

[28] Neil Shephard. 1993. Fitting nonlinear time-series models with applications to stochastic variance models. *Journal of Applied Econometrics* 8, S1 (1993), S135–S152.

[29] Neil Shephard and Torben G. Andersen. 2009. Stochastic volatility: Origins and overview. In *Handbook of Financial Time Series*. Springer, 233–254.
[30] Umut Şimşekli. 2017. Fractional Langevin Monte Carlo: Exploring Lévy driven stochastic differential equations for Markov chain Monte Carlo. In *International Conference on Machine Learning*. PMLR, 3200–3209.
[31] Shuntaro Takahashi, Yu Chen, and Kumiko Tanaka-Ishii. 2019. Modeling financial time-series with generative adversarial networks. *Physica A: Statistical Mechanics and its Applications* 527 (2019), 121261.
[32] Stephen John Taylor. 1982. Financial returns modelled by the product of two stochastic processes-a study of the daily sugar prices 1961-75. *Time Series Analysis: Theory and Practice* 1 (1982), 203–226.
[33] Zhu Tingyu and Zheng Zeyu. 2021. Learning to simulate sequentially generated data via neural networks and Wasserstein training. In *2021 Winter Simulation Conference (WSC)*. IEEE.
[34] Tan Wan and L. Jeff Hong. 2022. Large-scale inventory optimization: A recurrent-neural-networks-inspired simulation approach. *INFORMS Journal on Computing* (2022).
[35] Ruixin Wang, Prateek Jaiswal, and Harsha Honnappa. 2020. Estimating stochastic Poisson intensities using deep latent models. In *2020 Winter Simulation Conference (WSC)*. IEEE, 596–607.
[36] Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. 2020. Quant GANs: Deep generation of financial time series. *Quantitative Finance* 20, 9 (2020), 1419–1440.
[37] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. 2019. Time-series generative adversarial networks. (2019).
[38] Yufeng Zheng, Zeyu Zheng, and Tingyu Zhu. 2020. A doubly stochastic simulator with applications in arrivals modeling and simulation. *arXiv preprint arXiv:2012.13940* (2020).

Received 14 January 2022; revised 4 January 2023; accepted 10 January 2023