Proceedings of the 2021 Winter Simulation Conference
S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, eds.

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

Tingyu Zhu

Yuanpei College
Peking University
No.5 YiHeYuan Road, Haidian District
Beijing, 100871, CHINA

Zeyu Zheng

Department of Industrial Engineering
and Operations Research
University of California, Berkeley
Berkeley, CA 94720, USA

ABSTRACT

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 through a Wasserstein training process, without restrictive distributional assumptions. Moreover, the simulator can flexibly incorporate various kinds of elementary randomness and generate distributions with certain properties such as heavy-tail. Regarding statistical properties, we provide results on consistency and convergence rate for estimation of the simulator. We then present numerical experiments with synthetic and real data sets to illustrate the performance of our estimation method.

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 time period and some new randomness as input, and maps the input to a new state passing on the next time period. Such simulators are used to simulate *sequentially generated data*, which are in turn used to evaluate system performances or 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, representative real data are available that record the dynamics of some but maybe not all dimensions of the stochastic process modeled by the simulator. With the real data in hand, there is a natural need to tailor the simulator such that the sequentially generated data from the simulator "match" the real data in the corresponding dimensions. In this work, we propose a new framework assisted by neural networks and Wasserstein training to address this need. Our framework has two features. First, the framework integrates neural networks (NNs) into the simulator and the NNs are specifically used in each time period to capture the potential non-linear and complicated dependence of the dynamics on the previous state. Other than this, the sequential structure of the simulator and the randomness used by the simulator are chosen by users. In particular, the randomness can be chosen as a Gaussian random vector or a heavy-tailed random vector, depending on the domain knowledge and real need of the application. Second, without imposing any specific parametric distribution assumption, our framework applies Wasserstein training to meet the goal that the (possibly high-dimensional) joint distribution of the simulated data should match that

978-1-6654-3311-2/21/$31.00 ©2021 IEEE

of the real data. Apart from deriving the framework, we discuss its statistical aspect, which involves two 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 the best of our knowledge, such theoretical guarantees do not exist for general sequential-structure simulators assisted by neural networks, especially when the associated distributions have unbounded support.

The modeling and simulation of sequential generated data has been introduced and intensively studied in many literature. An important class of models are 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 Shephard and Andersen (2009) 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, Taylor (1982) 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 formalized as diffusion processes, such as the Heston model presented by Heston (1993), become more favorable in portfolio choice and derivatives pricing. The multivariate generalizations of SVMs are presented by Asai et al. (2006). Estimating such models poses substantial challenge due to difficulties in evaluating the exact likelihood function. Broto and Ruiz (2004) conclude that there are mainly three categories of estimation techniques to address the challenge, namely estimators based on the method of moments, such as Melino and Turnbull (1990); estimators based on the maximum likelihood principle, such as Shephard (1993) and estimators based on an auxiliary model, such as Bansal et al. (1994). A most representative application of auxiliary models is model calibration, which uses current information such as the option price in parameter estimation, see Aı et al. (2007) 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 are recurrent neural networks (RNN), which, along with other variants, aim to capture the transition between the state vectors in the time-series. Recently, variational autoencoders (VAE) provided by Kingma and Welling (2013) are combined with RNNs to model and estimate sequentially generated data with a latent stochastic process, see Luo et al. (2018), Yeo and Melnyk (2019) and Cen et al. (2020) for examples. Such frameworks adopt a likelihood-based statistical inference method, using VAEs to learn the posterior distributions of latent process variables whose prior distribution and randomness-generation distribution need to pre-specified. Different from such frameworks, our work adopts a training process which directly learns the distributions of both the observed and latent variables using only partially observed data. Moreover, our simulator differs from RNN and its variants in that the time-dependence captured by the latent process can be more explicitly explained by the original sequential structured model, and the randomness unincorporated in the neural networks induces a more significant and flexible impact on the simulated distribution. Such deviations from RNN are fundamental, from both computational and theoretical aspects. The Wasserstein training of our neural network-based simulator is inspired by generative adversarial networks (GAN) (Goodfellow et al. (2014)), Wasserstein generative adversarial networks (WGAN) (Arjovsky et al. (2017)) and the doubly stochastic WGAN framework by Zheng and Zheng (2020). Bai et al. (2018), Chen et al. (2019) and Chen et al. (2020) are representative theoretical works in this area, and serve as fundamentals of the proof of our theorem. The general idea of adversarial training is also applicable to model calibration, see Cuchiero et al. (2020) for example. Such methods require additional information such as option price, and are more restricted to estimation of financial models.

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 4 discusses the statistical theory for the estimation method. Section 5 provides numerical experiments.

2 MODEL SETUP

We consider a class of simulators that are used to simulate sequentially generated 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 simulator generates 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 $\mu(\cdot,\cdot)$ that 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 $k+1$.

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 $d_1 \leq d$ 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 $0 \leq k_1 < k_2 < \cdots < k_p$. 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 $\boldsymbol{S}$. 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 $(X_k : k = 0,1,2,\ldots)$ match 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 multi-dimensional diffusion model use the idea of discretization, at a user-specified discretization resolution. Using the Euler-Maruyama discretization scheme 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 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 process is conducted.

The second example is a simulator for sequential data with heavy-tailed distributions. This simulator sequentially updates $(X_k, k = 0,1,2,\ldots)$ according to

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

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

3 METHOD

In this section, we use a new framework to estimate the simulator in consideration 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 the simulated data and the joint distribution of the 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 formalized and computed, and how the NN parameters are updated according to the computed distance. In the following part of this section, Subsection 3.1 provides answers to the first two questions, and formulates the estimation problem into a minimax optimization problem. Subsection 3.2 answers the third question by discussing the training process to solve the optimization problem.

3.1 Framework

Recall that $\boldsymbol{S} = (S_{k_1}, S_{k_2}, \ldots, S_{k_p})$ represents the observed sequence. Let π_S denote 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 $\tilde{\pi}_S$ 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

$$X_{k+1} = \mu_\theta(X_k) + \Sigma_\phi(X_k)\eta_{k+1}, \tag{6}$$

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 neural network (NN) architecture, parameterized by NN parameters θ and ϕ, and are approximations of $\mu(\cdot)$ and $\Sigma(\cdot)$ of the simulator. We refer to Zheng and Zheng (2020) for a detailed description of the functional form. We let X_0 be a given constant or a random vector with a given probability distribution. 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 $\hat{\pi}_S$, is taken as the output of the generator. Note that $\hat{\pi}_S$ and $\hat{\boldsymbol{S}}$ are also functions of θ and ϕ, and are therefore sometimes denoted as $\hat{\pi}_S(\theta, \phi)$ and $\hat{\boldsymbol{S}}(\theta, \phi)$.

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

$$W(\hat{\pi}_S, \pi_S) = \inf_{\gamma \in \Pi(\hat{\pi}_S, \pi_S)} \mathbb{E}_{(\hat{\boldsymbol{S}}, \boldsymbol{S}) \sim \gamma} \mathbb{E}[\| \hat{\boldsymbol{S}} - \boldsymbol{S} \|_2], \tag{7}$$

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

$$W(\hat{\pi}_S, \pi_S) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\hat{\boldsymbol{S}} \sim \hat{\pi}_S}[f(\hat{\boldsymbol{S}})] - \mathbb{E}_{\boldsymbol{S} \sim \pi_S}[f(\boldsymbol{S})], \tag{8}$$

where $\| f \|_L \leq 1$ denotes the class of all 1-Lipschitz functions f, i.e., $|f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)| \leq \| \boldsymbol{x}_1 - \boldsymbol{x}_2 \|_2$ for any $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^{d_\pi}$. The supreme f over all 1-Lipschitz functions is still intractable, but we can use a neural network f_ψ to approximate f, and search over all such approximations parameterized by NN parameters ψ. The detailed description of the functional form of f_ψ is also presented in Zheng and Zheng (2020). In the framework of the classical Wasserstein Generative Adversarial Network (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:

$$\min_{\theta \in \Theta, \phi \in \Phi} \max_{\psi \in \Psi} \mathbb{E}_{\hat{\boldsymbol{S}} \sim \hat{\pi}_S(\theta, \phi)}[f_\psi(\hat{\boldsymbol{S}})] - \mathbb{E}_{\boldsymbol{S} \sim \pi_S}[f_\psi(\boldsymbol{S})]. \tag{9}$$

The empirical version of problem (9) is given by

$$\min_{\theta \in \Theta, \phi \in \Phi} \max_{\psi \in \Psi} \frac{1}{n} \sum_{j=1}^{n} f_\psi(\hat{\boldsymbol{S}}_j(\theta, \phi)) - \frac{1}{n} \sum_{j=1}^{n} f_\psi(\boldsymbol{S}_j). \tag{10}$$

3.2 Training

In this section, we discuss the training process for model estimation optimization problem (10). 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 (10), 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 stochastic gradient descent (SGD) (Gulrajani et al. (2017)) 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. Such computation is illustrated in the following Figure 1.

4 STATISTICAL PROPERTIES

In this section, we discuss the statistical property of the estimation method. We prove that the framework proposed in section 3 can effectively learn distributions of a wide class of sequentially generated data, if the neural network architectures are properly chosen, and the number of copies of $\boldsymbol{S}$, denoted as n,

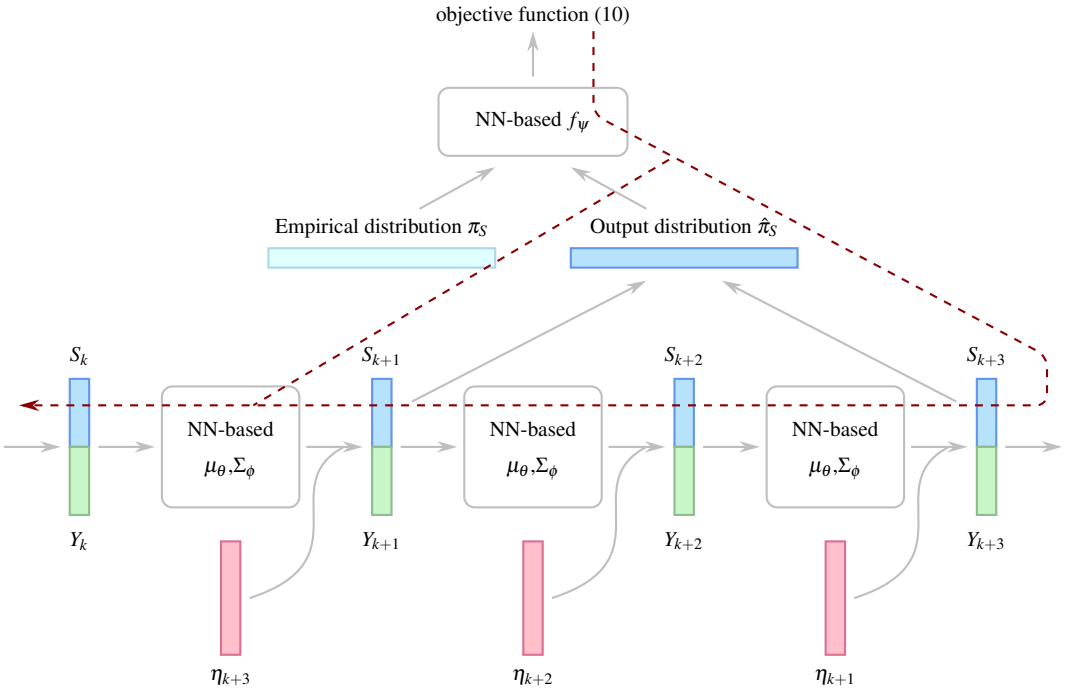


Figure 1: Backpropogation diagram (red dashed line) on the gradient evaluation of $f_\psi(\hat{\boldsymbol{S}}(\theta,\phi))$ with respect to the parameters θ,ϕ in the simulator neural network μ_θ,Σ_ϕ.

is large enough. We assume an underlying true model that sequentially generates the empirical data $\boldsymbol{S}_j = (S_{j,k_0}, S_{j,k_1}, S_{j,k_2}, \ldots, S_{j,k_p}), j = 1,2,\ldots,n$ according to

$$X_{k+1} = \mu(X_k) + \Sigma(X_k)\eta_{k+1}, \quad k = 0,1,2,\ldots \tag{11}$$

where $X_k = (S_k, Y_k)^\top$ is the d-dimensional state process, S_k is the d_1-dimensional observed process, and Y_k is the $(d-d_1)$-dimensional latent process. To simplify the problem, we assume that X_0 is a fixed given constant.

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 generated sequence $(X_k : k = 1,2,\ldots)$. To address the challenge due to unboundedness, we perform clipping methods on both the empirical data and the simulated data. Specifically, we transform the empirical data into its bounded version $\boldsymbol{S}_j^{B_0}$ according to

$$S_{j,k_i}^{B_0} = \begin{cases} S_{j,k_i}, \text{ if } S_{j,k_i} \le B_0; \\ B_0, \text{ else.} \end{cases} \quad i = 0,1,2,\ldots p, \quad j = 1,2,\ldots,n,$$

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

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

where B_0 is a given constant, and B is selected to ensure that each generated X_k falls into the range of $[-B_0, B_0]^d$. Both constants are allowed to increase along with the increase in the width and depth of the neural networks μ_θ, Σ_ϕ and f_ψ. We denote the the bounded versions of the empirical data and the simulated

data as $\boldsymbol{S}_j^{B_0}$ and $\hat{\boldsymbol{S}}_j^B$, and their distributions as $\pi_S^{B_0}$ and $\hat{\pi}_S^B$. The empirical optimization problem (10) is then transformed into the bounded version:

$$\min_{\theta\in\Theta,\phi\in\Phi}\max_{\psi\in\Psi}\frac{1}{n}\sum_{j=1}^{n}f_\psi(\hat{\boldsymbol{S}}_j^B(\theta,\phi))-\frac{1}{n}\sum_{j=1}^{n}f_\psi(\boldsymbol{S}_j^{B_0}). \tag{12}$$

It can be proved that the Wasserstein distance between the bounded distributions, denoted as $W(\pi_S^{B_0},\hat{\pi}_S^B)$, can be controlled. As the size of NN goes to infinity, B_0 and B also increase to infinity, which, with certain restrictions on η_k that will be presented in the following assumption, results in the convergence of the bounded distribution towards its unbounded origin, i.e., $\pi_S^{B_0}\to\pi_S$. This convergence enables the controlling of $W(\pi_S,\pi_S^{B_0})$, and thus $W(\pi_S,\hat{\pi}_S^B)$.

We make the following assumption on the underlying true model and the generated empirical data:

Assumption 1 The following conditions are satisfied for the generation process of empirical data (11):

1. All sequences $\boldsymbol{S}_j$ are independent and identically distributed (i.i.d.).
2. $\mu:\mathbb{R}^d\to\mathbb{R}^d$ and $\Sigma:\mathbb{R}^d\to\mathbb{R}^{d\times d_0}$ 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 $(\eta_k:k=1,2,\ldots)$ is no more than $x^{-\frac{1}{3+\alpha}}$, for some $\alpha>0$. Namely, $p_{\eta_k}(x)=O(x^{-\frac{1}{3+\alpha}})$, for all $k=1,2,\ldots$.

The first condition is assumed in order to effectively control the statistical error (see Chen et al. (2020) for a detailed explanation), which is given as

$$\sup_{\|f\|_L\le 1}\mathbb{E}_{\boldsymbol{S}\sim\pi_S}[f(\boldsymbol{S})]-\mathbb{E}_{\hat{\boldsymbol{S}}\sim\tilde{\pi}_S}[f(\tilde{\boldsymbol{S}})]+\sup_{\psi\in\Psi}\mathbb{E}_{\boldsymbol{S}\sim\pi_S}[f_\psi(\boldsymbol{S})]-\mathbb{E}_{\hat{\boldsymbol{S}}\sim\tilde{\pi}_S}[f_\psi(\tilde{\boldsymbol{S}})] \tag{13}$$

where π_S denotes the real underlying distribution and $\tilde{\pi}_S$ denotes the empirical distribution of the n data points. 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 clipped 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., $W(\pi_S,\pi_S^{B_0})$. We additionally remark that such assumptions, as well as the bounding strategies, are imposed only to guarantee statistical convergence in the following theorem, and are unnecessary in practical applications. For example, there is no need to perform clipping on the empirical data or the elementary randomness, and the empirical observations of $\boldsymbol{S}$ do not need to be mutually independent in practice.

The main result of statistical analysis is presented as follows:

Theorem 1 Suppose that n copies of i.i.d. 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 (12), the clipping boundaries B and B_0 approach infinity along with n, by order $B_0=C_1\cdot B=O(n^{\frac{1}{(2+\alpha)(pd_1+2)}})$. We have

$$\mathbb{E}W(\pi_S,\hat{\pi}_S^B(\theta^*,\phi^*))\le C\cdot n^{-\frac{1+\alpha}{(2+\alpha)(pd_1+2)}}(\log n)^{\frac{3}{2}}, \tag{14}$$

where α has the same meanings as in assumption 1, p is the length of the observed sequence, d_1 is the dimension of the observed process, C and C_1 are both constants that are independent of n, but relevant to pd_1, α, the Lipschitz constants and bounds of $\mu(\cdot)$ and $\Sigma(\cdot)$, and the length of the simulated sequence $(X_k:k=0,1,2,\ldots)$.

Due to length limit, we defer the detailed proofs of the theorem to a future work. The essence of the proof is summarized as follows. We first decompose the error into three parts, namely the bounding error, the network approximation error and the statistical error. The bounding error is controlled using a coupling technique. The discriminator approximation error $\| f - f_\psi \|_\infty$ and generator approximation errors $\| \mu - \mu_\theta \|_\infty$ and $\| \Sigma - \Sigma_\phi \|_\infty$, together referred to as the network approximation error, are controlled using function approximation theory provided by Chen et al. (2019) and Chen et al. (2020). Further, the accumulated error in the sequentially generated data distribution, which is induced by the generator approximation errors, is controlled using the law of total expectations. The statistical error is then balanced with the network approximation errors, using theories of empirical process and network approximation.

We briefly discuss the implications of Theorem 1. Theorem 1 shows that with appropriately chosen neural network architectures, for an arbitrary sequential stochastic model, if the underlying real generation structure has $\mu(\cdot)$, $\Sigma(\cdot)$ and η_k satisfying certain properties, the proposed estimation method can provide consistent estimation of the simulator in that the simulated distribution converges to the underlying real distribution under the Wasserstein distance measurement. To the best of our knowledge, this is the first statistical theory on using neural networks to estimate and simulate sequentially generated data with unbounded elementary randomness.

5 NUMERICAL EXPERIMENTS

In this section, we evaluate the performance of the simulator estimated by our proposed framework, using two sets of synthetic data (Subsection 5.1 and Subsection 5.2) and one set of real data (Subsection 5.3) as the training data. 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 consider the task of evaluating the distribution of the maximal drawdown (MDD) for a portfolio that is formed by a combination of all assets in consideration. In each experiment, we consider the distribution of the MDD of a portfolio which consists 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 first present the definition of maximal drawdown, which is derived from Chekhlov et al. (2005):

Definition 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

$$D_i = \max_{0 \le l \le i} \frac{H_l - H_i}{H_i}, \tag{15}$$

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, $M = \max_{0 \le i \le p} D_i$.

5.1 Multi-dimensional Heston Model

In this subsection, we use a synthesized data set of three stock prices and the maximal drawdown distribution of a portfolio to evaluate the performance of our estimated simulator.

5.1.1 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 A1 et al. (2007), which also presents

	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

Table 1: Parameters of the underlying multi-dimensional Heston model

the values of the model parameters):

$$d\begin{pmatrix} S_t \\ Y_t \end{pmatrix} = \begin{pmatrix} \mu S_t \\ \kappa(\gamma - Y_t) \end{pmatrix} dt + \begin{pmatrix} S_t\sqrt{(1-\rho^2)Y_t} & \rho S_t\sqrt{Y_t} \\ 0 & \sigma\sqrt{Y_t} \end{pmatrix} dW_t \tag{16}$$

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 is the 2-dimensional canonical Brownian motion. We use a matrix L to induce correlation among the three stochastic processes, namely, we let

$$d\begin{pmatrix} X_{1,t} \\ X_{2,t} \\ X_{3,t} \end{pmatrix} = \begin{pmatrix} \mu_{1,t} \\ \mu_{2,t} \\ \mu_{3,t} \end{pmatrix} dt + (L \otimes I_2) \cdot \begin{pmatrix} \Sigma_{1,t} & 0 & 0 \\ 0 & \Sigma_{2,t} & 0 \\ 0 & 0 & \Sigma_{3,t} \end{pmatrix} d\begin{pmatrix} W_{1,t} \\ W_{2,t} \\ W_{3,t} \end{pmatrix}, \tag{17}$$

where

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

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 $\boldsymbol{S}_0 \sim \mathcal{N}((100,100,100), 70 \cdot P)$, and $\boldsymbol{Y}_0 = (0.1, 0.1, 0.1)$. There are $n = 500$ 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 $7/(365 \times 30)$. This synthetic data set is then used as input data for the discriminator.

5.1.2 Training Process and Results

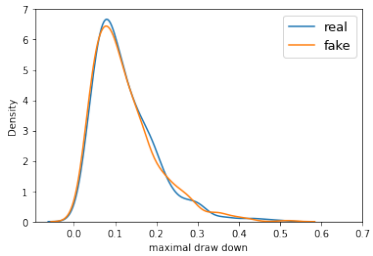
We specify the structure of the simulator as

$$X_{k+1} = X_k + \mu_\theta(X_k)\Delta t + \Sigma_\phi(X_k)\eta_{k+1}\sqrt{\Delta t} \tag{18}$$

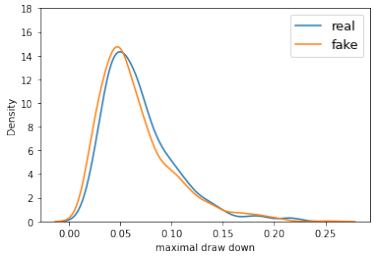
where $\eta_k : k = 1, 2, \ldots$ are independent 6-dimensional canonical normal variables, Δt is set as $7/(365 \times 30)$, which is the same as the length of the sub-intervals used in synthetic data generation. The parameterization of μ_θ is given by $L = 4$, $\tilde{n} = (n_1, n_2, n_3, n_4) = (10, 10, 10, 6)$, the parameterization of Σ_ϕ is given by $L = 4$,$\tilde{n} = (n_1, n_2, n_3, n_4) = (80, 80, 80, 36)$, and the parameterization of f_ψ is given by $L = 4$, $\tilde{n} = (n_1, n_2, n_3, n_4) = (500, 500, 500, 1)$. 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 10, and the batch size for sampling

from synthetic data and for simulator generation is set as 256. The initial values $\boldsymbol{S}_0$ of each simulation process is set to be the same as the initial values of the sample batch, and $\boldsymbol{Y}_0$ is also set as $(0.1, 0.1, 0.1)$. The training process is carried out with 500 iterations using the Adam optimizer (Kingma and Ba (2014)) with coefficients $\beta_1 = 0.5$ and $\beta_2 = 0.9$. Within each iteration, f_ψ is updated 5 times. The learning rate of μ_θ and Σ_ϕ decays exponentially from $5e-4$ to $1e-7$, and the learning rate of f_ψ decays exponentially from $1e-4$ to $1e-7$. The training takes about 34 minutes. The final Wasserstein distance between the simulated distribution and the synthesized empirical distribution is 6.16.

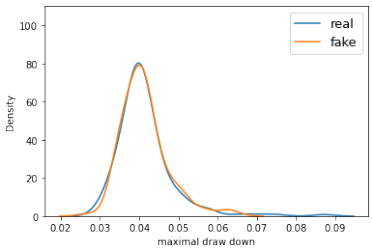
We next evaluate the training results. We define a portfolio $H_t = (S_{1,t} + S_{2,t} + S_{3,t})/3$. The comparison between the distribution of the maximal drawdown of synthesized stock price-based H_t and that of the simulated stock price-based H_t is illustrated in the following figure 2a. The sizes of the synthesized data set and the simulated data set used for comparison are both 500.



(a) Comparison of the smoothed histograms of synthesized and simulated maximal drawdowns in experiment 1.



(b) Comparison of the smoothed histograms of synthesized and simulated maximal drawdowns in experiment 2.



(c) Comparison of the smoothed histograms of real and simulated maximal drawdowns in experiment 3.

Figure 2: numerical results

5.2 Multi-dimensional Polynomial Model

In this subsection, we use a synthesized data set generated by a nonlinear SDE-based stochastic process. We also use the maximal drawdown distribution of a portfolio to evaluate the performance of our estimated simulator.

5.2.1 Underlying Model for Synthetic Data: Multi-dimensional Polynomial

The underlying stochastic process is formulated by a stochastic differential equation given by:

$$d\,(\boldsymbol{S}_t, \boldsymbol{Y}_t)^\top = \mu(\boldsymbol{S}_t, \boldsymbol{Y}_t)dt + \Sigma(\boldsymbol{S}_t, \boldsymbol{Y}_t)dW_t \tag{19}$$

where

$$\mu(\boldsymbol{S}_t, \boldsymbol{Y}_t) = \begin{pmatrix} S_{1,t}^{0.2} + S_{2,t}^{0.2} + 1 & S_{2,t}^{0.3} + 0.02 S_{1,t} S_{3,t} & S_{3,t}^{0.25} + 0.01 S_{1,t} & Y_{2,t} + Y_{3,t} & Y_{3,t} + Y_{1,t} & Y_{1,t} + Y_{2,t} \end{pmatrix}^\top$$

and

$$\Sigma(\boldsymbol{S}_t, \boldsymbol{Y}_t) = \begin{pmatrix} 0.3 S_{1,t}^{1.2} Y_{1,t} & 0.01 S_{1,t} S_{2,t} Y_{2,t} & 0.02 Y_{3,t} S_{1,t} S_{3,t} & 0.1 S_{1,t} Y_{1,t} & 0.5 S_{1,t} Y_{2,t} & 0.7 S_{1,t} Y_{3,t} \\ 0.02 S_{1,t} Y_{2,t} & 0.7 S_{2,t}^{1.1} Y_{3,t} & 0.05 Y_{1,t} S_{2,t} S_{3,t} & 0.1 S_{2,t} Y_{1,t} & 0.2 S_{2,t} Y_{2,t} & 0.2 S_{2,t} Y_{2,t} \\ 0.05 S_{1,t} S_{2,t} Y_{3,t} & 0.08 S_{2,t} S_{2,t} Y_{1,t} & 0.1 S_{3,t}^{0.9} Y_{2,t} & 0.2 S_{3,t} Y_{1,t} & 0.6 S_{3,t} Y_{2,t} & 0.3 S_{3,t} Y_{3,t} \\ Y_{1,t} & 0 & 0 & Y_{2,t} Y_{3,t} & Y_{3,t} Y_{1,t} & Y_{1,t} Y_{2,t} \\ 0 & Y_{2,t} & 0 & Y_{1,t} Y_{3,t} & Y_{1,t} Y_{2,t} & Y_{3,t} Y_{2,t} \\ 0 & 0 & Y_{3,t} & Y_{2,t} Y_{1,t} & Y_{3,t} Y_{2,t} & Y_{1,t} Y_{3,t} \end{pmatrix}$$

We next describe how the data set is synthesized. The initial values are given by $S_{i,0} \sim \mathcal{N}(0,1)$, where all three dimensions are independent and identically distributed, and $\boldsymbol{Y}_0 = (0.1, 0.2, 0.15)$. There are $n = 1000$ 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.

5.2.2 Training Process and Results

We specify the structure of the simulator to have the same form as (18). The neural network parameterization and 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 8 minutes with one Nvidia Quadro P4000 GPU. The final Wasserstein distance between the simulated distribution and the synthesized empirical distribution is 0.29.

We next evaluate the training results.The portfolio H_t has the same definition as in the first experiment. Comparison between the distribution of the maximal drawdown of synthesized data-based H_t and that of the simulated data-based H_t is illustrated in the following figure 2b. The sizes of the synthesized data set and the simulated data set used for comparison are both 1000.

5.3 Stock Price

5.3.1 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 $(\hat{S}_i : i = 0, 1, 2, \ldots, 25)$ are recorded for every transaction day. The empirical data is processed as follows. We first subtract the initial value of each day from intra-daily sequences, i.e., let $S_i = \hat{S}_i - \hat{S}_0$, for $i = 0, 1, 2, \ldots, 25$. Then, we remove S_0, which now equals 0, from every sequence. All sequences now begin with S_1, and are regarded as weakly correlated identical copies of an underlying real distribution. After removing the sequences with missing data, we retain $n = 186$ such copies.

5.3.2 Training Process and Results

We specify the structure of the simulator to have the same form as (18), but set $\eta_k : k = 1, 2, \ldots$ as independent 2-dimensional random variables, each dimension following the t distribution with 2.8 degrees of freedom. The neural network parameterization and 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 5 minutes with one Nvidia Quadro P4000 GPU. The final Wasserstein distance between the simulated distribution and the real distribution is 0.33.

We next evaluate the training results. Since the stock price is 1-dimensional, we take itself as the portfolio, i.e., $H_t = \hat{S}_t$. Note that in order to avoid 0 as the dominator in equation (15), we add back the value of the first observation S_0 of each sequence and set H_t's as the real stock prices. The comparison between the distribution of the maximal drawdown of real data-based H_t and that of the simulated data-based H_t is illustrated in the following figures 2c. The sizes of the real data set and the simulated data set used for comparison are both 186.

6 CONCLUSION

We propose a new framework of a neural network-based sequential structured simulator to model, estimate, and simulate a wide class of sequentially generated data. In our future work, we plan to demonstrate through more numerical experiments and measurements the flexibility of our proposed structure of the simulator. We also plan to further study the simulation of heavy-tailed sequential data.

REFERENCES

Aı, Y., R. Kimmel et al. 2007. "Maximum likelihood estimation of stochastic volatility models". *Journal of Financial Economics* 83(2):413–452.

Arjovsky, M., S. Chintala, and L. Bottou. 2017. "Wasserstein generative adversarial networks". In *International Conference on Machine Learning*, 214–223. PMLR.

Asai, M., M. McAleer, and J. Yu. 2006. "Multivariate stochastic volatility: a review". *Econometric Reviews* 25(2-3):145–175.

Bai, Y., T. Ma, and A. Risteski. 2018. "Approximability of discriminators implies diversity in GANs". *arXiv preprint arXiv:1806.10586*.

Bansal, R., A. R. Gallant, R. Hussey, and G. Tauchen. 1994. "Computational aspects of nonparametric simulation estimation". In *Computational Techniques for Econometrics and Economic Analysis*, 3–22. Springer.

Broto, C., and E. Ruiz. 2004. "Estimation methods for stochastic volatility models: a survey". *Journal of Economic Surveys* 18(5):613–649.

Cen, W., E. A. Herbert, and P. J. Haas. 2020. "Nim: modeling and generation of simulation inputs via generative neural networks". In *Proceedings of the 2020 Winter Simulation Conference, edited by Bae, K., Feng, B., Kim, S., Lazarova-Molnar, S., Zheng, Z., Roeder, T., and Thiesing, R.*, 584–595. Piscataway, New Jersey: Institute of Electrical and Electronic Engineers, Inc.

Chekhlov, A., S. Uryasev, and M. Zabarankin. 2005. "Drawdown measure in portfolio optimization". *International Journal of Theoretical and Applied Finance* 8(01):13–58.

Chen, M., H. Jiang, W. Liao, and T. Zhao. 2019. "Nonparametric Regression on Low-Dimensional Manifolds using Deep ReLU Networks: Function Approximation and Statistical Recovery". *arXiv preprint arXiv:1908.01842*.

Chen, M., W. Liao, H. Zha, and T. Zhao. 2020. "Statistical guarantees of generative adversarial networks for distribution estimation". *arXiv preprint arXiv:2002.03938*.

Cuchiero, C., W. Khosrawi, and J. Teichmann. 2020. "A generative adversarial network approach to calibration of local stochastic volatility models". *arXiv*.

Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. "Generative adversarial networks". *arXiv preprint arXiv:1406.2661*.

Gulrajani, I., F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. 2017. "Improved training of wasserstein GANs". *arXiv preprint arXiv:1704.00028*.

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

Kingma, D. P., and J. Ba. 2014. "Adam: A method for stochastic optimization". *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., and M. Welling. 2013. "Auto-encoding variational bayes". *arXiv preprint arXiv:1312.6114*.

Luo, R., W. Zhang, X. Xu, and J. Wang. 2018. "A neural stochastic volatility model". In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 32.

Melino, A., and S. M. Turnbull. 1990. "Pricing foreign currency options with stochastic volatility". *Journal of Econometrics* 45(1-2):239–265.

Shephard, N. 1993. "Fitting nonlinear time-series models with applications to stochastic variance models". *Journal of Applied Econometrics* 8(S1):S135–S152.

Shephard, N., and T. G. Andersen. 2009. "Stochastic volatility: origins and overview". In *Handbook of Financial Time Series*, 233–254. Springer.

Taylor, S. J. 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* 1:203–226.

Yeo, K., and I. Melnyk. 2019. "Deep learning algorithm for data-driven simulation of noisy dynamical system". *Journal of Computational Physics* 376:1212–1231.

Zheng, Y., and Z. Zheng. 2020. "Doubly Stochastic Generative Arrivals Modeling". *arXiv preprint arXiv:2012.13940*.

AUTHOR BIOGRAPHIES

TINGYU ZHU is an undergraduate student majoring in mathematics from Yuanpei College at Peking University. Her current research interest is in stochastic process. This is her first research project. Her email address is 1800017813@pku.edu.cn.

ZEYU ZHENG is an assistant professor in the Department of Industrial Engineering & Operations Research at University of California Berkeley. He received his Ph.D. in Management Science and Engineering, Ph.D. minor in Statistics and M.A. in economics from Stanford University, and a B.S. in Mathematics from Peking University. He has done research in simulation, nonstationary stochastic modeling and decision making, data analytics, and financial technology. His email address is zyzheng@berkeley.edu and his website can be found at https://zheng.ieor.berkeley.edu/.