LANGEVINIZED ENSEMBLE KALMAN FILTER FOR LARGE-SCALE DYNAMIC SYSTEMS

by

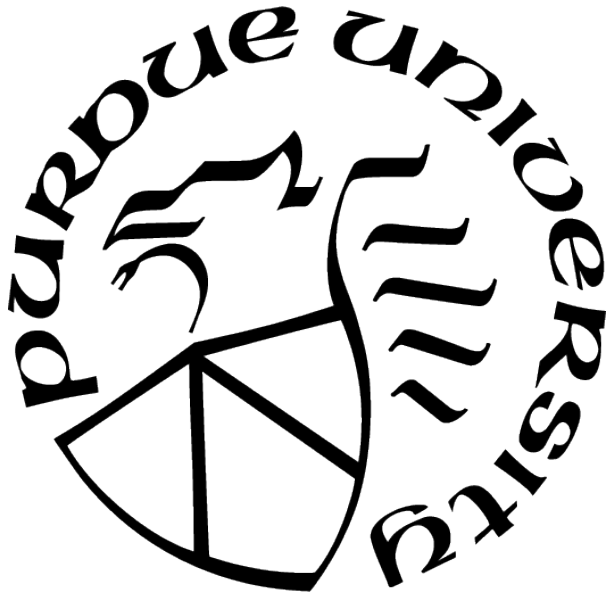
Peiyi Zhang

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy



Department of Statistics

West Lafayette, Indiana

August 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Faming Liang, Chair

Department of Statistics, Purdue University

Dr. Chuanhai Liu

Department of Statistics, Purdue University

Dr. Xiao Wang

Department of Statistics, Purdue University

Dr. Qifan Song

Department of Statistics, Purdue University

Approved by:

Dr. Jun Xie

To my parents, Fengmin Du and Weisong Zhang.

ACKNOWLEDGMENTS

There are many people and things coming to my mind when I am writing the acknowledgements. Undoubtedly, the first person on the list is my advisor, Dr. Faming Liang. This dissertation would not have been possible without his careful support. I would like to express my deepest gratitude to him, for his knowledgeable mentorship, for his interesting thoughts, for his persistent guidance, for his helpful advice, for his sincere care, and for his inspirational encouragement. I am extremely fortunate to have him as my advisor. I really enjoyed and will truly miss our discussion time, when I was led to an exciting world under his supervision. More importantly, his spirit of perseverance and his courage in the face of difficulties will always inspire me.

I would also like to thank Dr. Chuanhai Liu, Dr. Xiao Wang, Dr. Qifan Song, and Dr. Faming Liang for serving on my dissertation committee. I thank Dr. Jun Xie for being a really great advisor when I first came to Purdue. I want to thank all the professors who have taught me at Purdue University. I also want to thank Mary Sigman, Patti Foster, and all other Department of Statistics staff members for their kind assistance and support.

I am so fortunate and happy to discuss problems with my classmates and office-mates. I learned a lot from them, and I am very grateful to them for their helps. I would like to thank my friends, who always support and encourage me in their own ways.

Last but not least, my deepest gratitude goes to my parents for their endless love, support and encouragement. Without them, I would not be able to move forward. To them, I dedicate this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The Ensemble Kalman filter (EnKF) has achieved great successes in data assimilation in atmospheric and oceanic sciences, but its failure in convergence to the right filtering distribution precludes its use for uncertainty quantification. Other existing methods, such as particle filter or sequential importance sampler, do not scale well to the dimension of the system and the sample size of the datasets. In this dissertation, we address these difficulties in a coherent way.

In the first part of the dissertation, we reformulate the EnKF under the framework of Langevin dynamics, which leads to a new particle filtering algorithm, the so-called Langevinized EnKF (LEnKF). The LEnKF algorithm inherits the forecast-analysis procedure from the EnKF and the use of mini-batch data from the stochastic gradient Langevin-type algorithms, which make it scalable with respect to both the dimension and sample size. We prove that the LEnKF converges to the right filtering distribution in Wasserstein distance under the big data scenario that the dynamic system consists of a large number of stages and has a large number of samples observed at each stage, and thus it can be used for uncertainty quantification. We reformulate the Bayesian inverse problem as a dynamic state estimation problem based on the techniques of subsampling and Langevin diffusion process. We illustrate the performance of the LEnKF using a variety of examples, including the Lorenz-96 model, high-dimensional variable selection, Bayesian deep learning, and Long Short-Term Memory (LSTM) network learning with dynamic data.

In the second part of the dissertation, we focus on two extensions of the LEnKF algorithm. Like the EnKF, the LEnKF algorithm was developed for Gaussian dynamic systems containing no unknown parameters. We propose the so-called stochastic approximation-LEnKF (SA-LEnKF) for simultaneously estimating the states and parameters of dynamic systems, where the parameters are estimated on the fly based on the state variables simulated by the LEnKF under the framework of stochastic approximation. Under mild conditions, we prove the consistency of resulting parameter estimator and the ergodicity of the SA-LEnKF. For non-Gaussian dynamic systems, we extend the LEnKF algorithm (Extended LEnKF) by introducing a latent Gaussian measurement variable to dynamic systems. Those two

extensions inherit the scalability of the LEnKF algorithm with respect to the dimension and sample size. The numerical results indicate that they outperform other existing methods in both states/parameters estimation and uncertainty quantification.

1. INTRODUCTION

1.1 Bayesian on-line learning with large-scale dynamic data

Coming with the new century, the integration of computer technology into science and daily life has enabled scientists to collect massive volumes of data, such as satellite data, high-throughput biological assay data and website transaction logs. To address computational difficulty encountered in Bayesian analysis of big data, a variety of scalable MCMC algorithms have been developed, including stochastic gradient MCMC algorithms (see, e.g., [1]–[8]), split-and-merge algorithms (see, e.g., [9]–[12]), mini-batch Metropolis-Hastings algorithms (see, e.g., [13]–[17]), nonreversible Markov process-based algorithms (see, e.g.,[18], [19]), and Bayesian bootstrapping (see, e.g.,[20], [21]).

Although the scalable MCMC algorithms have achieved great successes in Bayesian learning with static data, none of them could be directly applied to dynamic data. In the literature, learning with static data is often termed as static or off-line learning, and learning with dynamic data is often termed as dynamic or on-line learning. Dynamic learning is important and challenging, as dynamic data collection is general, heterogeneous and messy.

Consider a state space model (SSM) of the form

$$\begin{aligned} x_t &= g(x_{t-1}) + u_t, \quad u_t \sim N(0, U_t), \\ y_t &= H_t x_t + \eta_t, \quad \eta_t \sim N(0, \Gamma_t), \end{aligned} \tag{1.1}$$

for $t = 1, 2, \ldots, T$, where $x_t \in \mathbb{R}^p$ and $y_t \in \mathbb{R}^{N_t}$ denote, respectively, the state and observations at stage t; and the dimension p, the number of stages T, and the sample sizes N_t't are all assumed to be very large; and we assume that the model error u_t and observation error η_t are zero-mean Gaussian random variables, and that the covariance matrices U_t and Γ_t and the propagator $g(\cdot)$ and H_t are all fully specified, i.e. containing no unknown parameters. For the dynamic system, the top equation is called the state evolution equation, where $g(\cdot)$ is called state propagator and can be nonlinear; and the bottom equation is called the measurement equation, where the propagator H_t relates the state variable to the measurement variable and yields the expected value of the prediction given the model states and param-

eters. Throughout this dissertation, we let $f(y_t|x_t)$ denote the likelihood function of y_t, let $\boldsymbol{\pi}(x_t|y_{1:t})$ denote the filtering distribution at stage t given the data $\{y_1, y_2, \ldots, y_t\}$, and let $\boldsymbol{\pi}(x_t|y_{1:t-1}) = \int \boldsymbol{\pi}(x_t|x_{t-1})\boldsymbol{\pi}(x_{t-1}|y_{1:t-1})dx_{t-1}$ denote the predictive distribution of x_t given $\{y_1, y_2, \ldots, y_{t-1}\}$.

Among the existing algorithms of state estimation for SSMs, the Kalman filter [22] is perhaps the most famous one, which provides algebraic formulas for recursively updating the mean and variance of the state distribution when the state evolution equation is linear. For the problems with nonlinear state evolution equations, the extended Kalman filter [23] and unscented Kalman filter [24] have been developed. When the state dimension is high, say greater than a few thousands, the Kalman filter recursions becomes computationally infeasible, which requires storage and calculation of matrices of the dimension of the state variable. In this case, the ensemble Kalman filter (EnKF) [25] can be used, which represents the state distribution using an ensemble of equally weighted random samples, and replaces the covariance matrix by the sample covariance computed from the ensemble. The ensemble is propagated forward according to the state evolution equation and re-adjusted according to a linear regression when new data arrive. Unlike the Kalman filter, the EnKF also works when the state evolution equation is nonlinear. Because of its conceptual simplicity and ease of implementation, the EnKF has been widely used in atmospheric and oceanic sciences, where the applications are usually termed as data assimilation. However, unfortunately, as shown by Law, Tembine, and Tempone [26], the EnKF converges only to a mean-field filter, which provides the optimal linear estimator of the conditional mean but not the filtering distribution except in the large sample limit for linear systems. Similar results can be found in Le Gland, Monbet, and Tran [27], Bergou, Gratton, and Mandel [28] and Kwiatkowski and Mandel [29]. Other than the EnKF, classical sequential Monte Carlo or particle filter algorithms (see, e.g., [30], [31]) have also been used to infer the state distribution for model (1.1). However, we note that they lack the scalability necessary for dealing with large-scale dynamic data, which strive to make use of all available data at each processing step and also suffer from the sample degeneracy issue [32] when the state dimension is high and/or the time series is long. How to make Bayesian on-line learning with large-scale dynamic data has posed a great challenge on current statistical methods.

Our goal is to develop a scalable particle filtering algorithm under the big data scenario that the dimension p, the numbers of stages T and the sample sizes N_t's are all very large. Toward this goal, we reformulate the EnKF under the framework of Langevin dynamics. The resulting algorithm inherits the forecast-analysis procedure from the EnKF and the use of mini-batch data from the stochastic gradient Langevin-type algorithms. The former makes the new algorithm scalable with respect to the dimension, and the latter makes it scalable with respect to the sample size. To credit to its two precursors, the new algorithm is coined as Langevinized Ensemble Kalman filter (LEnKF). We prove that the LEnkF converges to the right filtering distributions in Wasserstein distance under the scenario that the number of stages T and the sample sizes N_t's are all large. We illustrate the performance of the LEnKF using a variety of examples, including the Lorenz-96 model [33], Bayesian deep learning, and Long Short Term Memory (LSTM) network learning.

1.2 Bayesian on-line learning with large-scale dynamic data and unknown parameters

Recall that in the dynamic system (1.1), we assume that the covariance matrices U_t and Γ_t and the propagator $g(\cdot)$ and H_t are all fully specified, i.e. containing no unknown parameters. However, in practice, unknown parameters are very common. Consider a more general SSM of the form

$$\begin{aligned} x_t &= g(x_{t-1}, \alpha) + u_t, \quad u_t \sim N(0, U_t(\zeta_x)), \\ y_t &= H_t(\beta) x_t + \eta_t, \quad \eta_t \sim N(0, \Gamma_t(\zeta_y)), \end{aligned} \tag{1.2}$$

where t indexes the stages of the system, $x_t \in \mathbb{R}^p$ represents the system state at stage t, $y_t \in \mathbb{R}^{N_t}$ represents the system observations at stage t, and $\boldsymbol{\theta} = (\alpha, \beta, \zeta_x, \zeta_y)$ are unknown parameters which are assumed to be invariant with respect to stage t. Both the model error u_t and the observation error η_t are zero-mean Gaussian random variables, and their covariances are parameterized by ζ_x and ζ_y, respectively. The nonlinear propagator $g(\cdot)$ contains the parameter vector α, and the linear propagator $H_t(\beta)$ contains the parameter vector β. The propagator $H_t(\beta)$ relates the state variable to the measured variable and yields

the expected value of the prediction given the model state and parameters. The problem that we are considering is to simultaneously estimate the state variables $\{x_1, x_2, \ldots, x_t, \ldots\}$ and the parameters $\boldsymbol{\theta}$ under the high-dimensional, big data and long series scenario; that is, the dimension of the state variable x_t can be very high, the number of observations contained in y_t can be very large, and the number of stages T can be very large. Under this scenario, as briefly reviewed in Section 1.1, many of the existing methods become impractical, which might not scale well with respect to the state dimension, the sample size, or the length of the time series. In addition, they might suffer from a memory issue when storage and/or calculation of the covariance matrix of the state variable is involved.

Towards simultaneous estimation of the states and parameters for the model (1.2), the maximum likelihood method (see, e.g., [34], [35]) and sequential Bayesian method (see, e.g.,[36], [37]) have been developed. However, these methods are full likelihood-based. Since evaluation of the full likelihood function requires a complete scan of all available data and to integrate out all state variables, these methods do not scale well to high-dimensional and big data problems. Quite recently, Aicher, Putcha, Nemeth, *et al.* [38] proposed a particle buffered stochastic gradient MCMC method, where the states are estimated using the particle filter and the parameters are estimated using a stochastic gradient MCMC algorithm. Although the algorithm is scalable to big data problems by employing the subsampling technique in the stochastic gradient MCMC step, it is not scalable to high-dimensional and long series problems because the particle filter used therein suffers from the sample degeneracy issue [32].

When the EnKF is used for state estimation, the state augmentation method (see, e.g.,[39]–[41]) is often used for parameter estimation, which augments the state vector by the model parameters and then constructs a new EnKF for the augmented model. Since the EnKF itself suffers from a convergence issue, so does the state augmentation method. Empirically, this method often works reasonably well for the model parameters that enter additively in the state evolution equation, but it can be problematic for the stochastic and multiplicative parameters. The former refers to the parameters controlling the variance of the system, i.e., ζ_x and ζ_y in model (1.2), and the latter refers to the parameters that are multiplied by the state variables. For stochastic parameters, the state augmentation method

often converges to arbitrary values that depend on the initial condition and noise realization [42]. For multiplicative parameters, the state augmentation method often leads to one realization of the model that is dynamically unstable [43].

We propose the so-called stochastic approximation-Langevinized EnKF (SA-LEnKF) for simultaneously estimating the states and parameters of the dynamic system (1.2), where the parameters are estimated on the fly based on state variables simulated by the LEnKF algorithm under the framework of stochastic approximation [44]. The proposed algorithm is general; it work well for all types of parameters: additive, multiplicative and stochastic. Under mild conditions, we establish the consistency of the resulting parameter estimator and the ergodicity of the SA-LEnKF. Consequently, the proposed method can be efficiently used in uncertainty quantification for dynamic systems. As an advantage inherited from the LEnKF, the proposed method can work well for long series, large scale and high-dimensional dynamic systems.

1.3 Bayesian on-line learning with large-scale dynamic non-Gaussian data

Another extension of State space models is for non-Gaussian data. Consider a general form of nonlinear and/or non-Gaussian SSM:

$$\begin{aligned} x_t &= g(x_{t-1}) + u_t, \quad u_t \sim N(0, U_t), \\ \boldsymbol{z}_t &\sim f(\cdot|x_t), \end{aligned} \tag{1.3}$$

where $x_t \in \mathbb{R}^p$ and $\boldsymbol{z}_t \in \mathbb{R}^{N_t}$ are called, respectively, the state and observation/measurement at stage t, p is the dimension of the state variable, and N_t is the sample size at stage t. The distribution $f(\cdot)$ in the measurement equation deviates from Gaussian. Let $\boldsymbol{z}_{1:t} = (\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_t)$ denote the collection of observations up to stage t. A major goal of the study of SSMs is to infer the filtering distribution $\boldsymbol{\pi}(x_t|\boldsymbol{z}_{1:t})$. Here, we assume that the model (1.3) contains no unknown parameters. Otherwise, as discussed at Section 1.2, SA-LEnKF or the state-augmentation scheme can be employed to simultaneously estimate the states and parameters.

The model (1.3) has been studied in the literature for over half a century. When both the state propagator and the mean function of $f(\cdot)$ can be nonlinear, and $f(\cdot)$ deviates from Gaussian, the particle filter has been used to infer the filtering distribution for model (1.3). However, as discussed in Section 1.1, the particle filter becomes impractical when the state dimension is high and/or the total number of stages is large, as it suffers from the sample degeneracy issue under these scenarios, and it's not scalable with respect to the sample size N_t's due to its Metropolis sampling nature, where a likelihood function needs to evaluated with all available data when a particle is generated at each stage. Other than the particle filter, Katzfuss, Stroud, and Wikle [45] proposed to use Markov chain Monte Carlo (MCMC), e.g., the Metropolis-Hasting algorithm [46], [47] and Gibbs sampler [48], and EnKF in a combined manner, where the Gaussian measurement variables required by the EnKF were imputed using a MCMC algorithm at each stage. However, since the EnKF doesn't converge to the right filtering distribution, so doesn't their algorithm. Moreover, their algorithm is not scalable with respect to the sample size as the algorithm always performs in the scale of entire dataset.

We extend the LEnKF algorithm to nonlinear and/or non-Gaussian systems by introducing a latent Gaussian measurement variable to the model (1.3), called Extended Langevinized EnKF (Extended LEnKF). The proposed algorithm can converge to the right filtering distribution as the number of stages becomes large, while inheriting the scalability of the LEnKF with respect to the dimension and the sample size.

1.4 Dissertation organization

We organize the rest of the dissertation as follows. Chapter 2 provides a brief review of a few preliminary topics, including the EnKF algorithm [25], the state-augmented EnKF algorithm, and the MCMC-EnKF algorithm proposed by Katzfuss, Stroud, and Wikle [45].

In chapter 3, we start with an introduction to the LEnKF algorithm, which is developed under model (1.1). We then reformulate the Bayesian inverse problem as a dynamic state estimation problem based on the techniques of subsampling and Langevin diffusion process, and describe details of the algorithm for linear/nonlinear inverse and data assimilation prob-

lems, and study the algorithm's convergence. Chapter 4 evaluates the performance of the LEnKF algorithm for both inverse and data assimilation problems using a variety of examples, including the Lorenz-96 model, high-dimensional variable selection, Bayesian deep learning, and Long Short-Term Memory (LSTM) network learning with dynamic data.

Starting from chapter 5, we switch to extensions the LEnKF algorithm, i.e., the SA-LEnKF (developed under model (1.2)) and the Extended LEnKF (developed under model (1.3)). We present algorithms' details in Section 5.1.1 and Section 5.2.1 and study their convergence in Section 5.1.2 and Section 5.2.2. Chapter 6 includes a collection of empirical results, including synthetic studies and real data analysis, to demonstrate the efficacy and practicality of our algorithms.

Last, we conclude in chapter 7 with a summary and discussion of the dissertation.

2. PRELIMINARIES

2.1 Ensemble Kalman filter

Consider the dynamic system (1.1). To estimate the state variables $x_1, x_2, \ldots, x_T$, where T denotes the total number of stages, Evensen [25] proposed the EnKF algorithm:

Algorithm 1: Ensemble Kalman filter

(i) Initialization: Initial an ensemble $x_0^{a,1}, x_0^{a,2}, \ldots, x_0^{a,m}$, where m denotes the ensemble size.

for *t=1,2,…,T* **do**

(ii) Forecast: For $\mathrm{i} = 1, 2, \ldots, m$, draw $u_t^{\mathrm{i}} \sim N(0, U_t)$ and calculate

$$x_t^{f,\mathrm{i}} = g(x_{t-1}^{a,\mathrm{i}}) + u_t^{\mathrm{i}}.$$

Calculate the sample covariance matrix of $x_t^{f,1}, \ldots, x_t^{f,m}$ and denote it by C_t.

(ii) Analysis: For $\mathrm{i} = 1, 2, \ldots, m$, draw $\eta_t^{\mathrm{i}} \sim N(0, \Gamma_t)$ and calculate

$$x_t^{a,\mathrm{i}} = x_t^{f,\mathrm{i}} + \hat{K}_t(y_t - H_t x_t^{f,\mathrm{i}} - \eta_t^{\mathrm{i}}) \stackrel{\Delta}{=} x_t^{f,\mathrm{i}} + \hat{K}_t(y_t - y_t^{f,\mathrm{i}}),$$

where $\hat{K}_t = C_t H_t^T (H_t C_t H_t^T + \Gamma_t)^{-1}$ forms an estimator for the Kalman gain matrix $K_t = S_t H_t^T \ (H_t S_t H_t^T + \Gamma_t)^{-1}$ and S_t denotes the covariance matrix of x_t^f.

The rationale underlying the EnKF can be explained as follows. Let x_t^f and x_t^a denote a generic sample obtained at the forecast and analysis step, respectively. The forecast step is to use the forecasted samples $\{x_t^{f,1}, \ldots, x_t^{f,m}\}$ to approximate the predictive distribution $\boldsymbol{\pi}(x_t|y_{1:t-1})$. Let μ_t' and S_t denote the mean and variance of $\boldsymbol{\pi}(x_t|y_{1:t-1})$, respectively. Hence, one can rewrite x_t^f as

$$x_t^f = \mu_t' + w_t,$$

where w_t is a random error with mean 0 and variance S_t. If $\boldsymbol{\pi}(x_t|y_{1:t-1})$ is Gaussian, by the identity $K_t = S_t H_t^T (H_t S_t H_t^T + \Gamma_t)^{-1} = (I - K_t H_t) S_t H_t^T \Gamma_t^{-1} = (H_t^T \Gamma_t^{-1} H_t + S_t^{-1})^{-1} H_t^T \Gamma_t^{-1}$, one can show

$$x_t^a = \left[\mu_t' + K_t(y_t - H_t \mu_t')\right] + \left[(I - K_t H_t) w_t - K_t \eta_t\right] = \mu_t + \mathrm{e}_t,$$

where $\mu_t = \mu_t' + K_t(y_t - H_t\mu_t')$ is the mean of $\boldsymbol{\pi}(x_t|y_{1:t}) \propto \boldsymbol{\pi}(y_t|x_t)\boldsymbol{\pi}(x_t|y_{1:t-1})$, and $\mathrm{e}_t = (I - K_tH_t)w_t - K_t\eta_t$ is a Gaussian random error with mean 0 and variance $\mathrm{Var}(\mathrm{e}_t) = (I - K_tH_t)S_t$; that is, x_t^a is a sample following the filtering distribution $\boldsymbol{\pi}(x_t|y_{1:t})$.

The EnKF has two attractive features which make it extremely successful in dealing with high-dimensional data assimilation problems such as those encountered in reservoir modeling [49], oceanography [50], and weather forecasting [51]. First, it approximates each filtering distribution $\boldsymbol{\pi}(x_t|y_{1:t})$ using an ensemble of particles. Since the ensemble size m is typically much smaller than p, it leads to dimension reduction and computational feasibility compared to the Kalman filter (see e.g., [52]). In particular, it approximates S_t by C_t, and the storage for the matrix C_t is replaced by particles and thus much reduced. Second, in generating particles from each filtering distribution, it avoids covariance matrix decomposition compared to conventional particle filters. It is known that an LU-decomposition of the covariance matrix has a computational complexity of $O(p^3)$. Instead, the EnKF employs a forecast-analysis procedure to generate particles, which has a computational complexity of $O(\max\{p^2N_t, N_t^3\} + mpN_t)$ for m particles at stage t. That is, the forecast-analysis procedure reduces the computational complexity of the algorithm when m and N_t are smaller than p. This explains why the EnKF is so efficient for high-dimensional problems.

Despite its great successes, the performance of the EnKF is sub-optimal. As shown by Law, Tembine, and Tempone [26], it converges only to a mean-field filter, which provides the optimal linear estimator of the conditional mean but not the filtering distribution except in the large sample limit for linear systems. Similar results can be found in Le Gland, Monbet, and Tran [27], Bergou, Gratton, and Mandel [28] and Kwiatkowski and Mandel [29].

As an extension, Iglesias, Law, and Stuart [53] applied the EnKF to solve the inverse problem, which is to find the parameter z given observations of the form

$$y = \mathcal{G}(z) + \eta, \tag{2.1}$$

where $\mathcal{G}(\cdot)$ is the forward response operator mapping the unknown parameter z to the space of observations, $\eta \sim N(0, \Gamma)$ is Gaussian random noise, and y is observed data. With the

state augmentation approach, they defined the new state vector as $x^T = (z^T, \mathcal{G}(z)^T)$ and an artificial dynamic system as

$$\begin{aligned} x_t &= x_{t-1}, \\ y_t &= Hx_t + \eta_t, \end{aligned} \tag{2.2}$$

where $H = (0, I)$ such that $HX = \mathcal{G}(z)$ holds, $y_t \equiv y$ for all $t = 1, 2, \ldots$, and $\eta_t \sim N(0, \Gamma)$. However, as mentioned previously, the EnKF does not converge to the filtering distribution, so the posterior distribution $\pi(z|y)$ cannot be well approximated by the ensemble, and thus uncertainty of the estimate of z cannot be correctly quantified. Numerically, Ernst, Sprungk, and Starkloff [54] demonstrated that for nonlinear inverse problems the large sample limit does not lead to a good approximation to the posterior distribution.

2.2 State-Augmented EnKF for simultaneous state and parameter estimation

Consider the dynamic system (1.2), where $\alpha, \beta, \zeta_x, \zeta_y$ are unknown parameters which are assumed to be invariant with respect to stage t. Let $\boldsymbol{\theta}_t = (\alpha_t, \beta_t, \zeta_{x,t}, \zeta_{y,t})$ be a $d_{\boldsymbol{\theta}}$-dimensional vector containing all unknown parameters at stage t. Towards simultaneous estimation of state x_t and $\boldsymbol{\theta}$, state augmentation schema can be applied. Define the augmented state vector z_t with dimension $d_x + d_{\boldsymbol{\theta}}$ as

$$z_t = \begin{bmatrix} \boldsymbol{\theta}_t \\ x_t \end{bmatrix}.$$

Then the dynamic system (1.2) can be rewritten as

$$\begin{aligned} z_t = \begin{bmatrix} \boldsymbol{\theta}_t \\ x_t \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\theta}_{t-1} \\ g(x_{t-1}, \alpha_{t-1}) \end{bmatrix} + \begin{bmatrix} \epsilon_t \\ u_t \end{bmatrix}, \\ y_t &= H_t^*(\beta_{t-1}) z_t + v_t, \end{aligned} \tag{2.3}$$

where $\epsilon_t \sim N(0, \sigma^2)$ for some pre-specified constant σ, $u_t \sim N(0, U(\eta_{x,t-1}))$, $v_t \sim N(0, V(\eta_{y,t-1}))$, $H_t^*(\beta_{t-1}) = (\mathbf{0}_t, H_t(\beta_{t-1}))$, and $\mathbf{0}_t$ is a $d_y \times d_{\boldsymbol{\theta}}$-dimensional matrix of zero. The State-Augmented EnKF is described as follow:

Algorithm 2: State-Augmented Ensemble Kalman Filter

(i) Initialization: Initialize an ensemble $z_0^{a,1}, z_0^{a,2}, \ldots, z_0^{a,m}$, where m denotes the ensemble size.

for *t=1,2,…, T* **do**

(ii) Forecast: For $\mathrm{i} = 1, 2, \ldots, m$, draw $u_t^{\mathrm{i}} \sim N_{d_x}(0, U(\zeta_{x,t-1}^{a,\mathrm{i}}))$ and $\epsilon_t^{\mathrm{i}} \sim N_{d_{\boldsymbol{\theta}}}(0, \sigma_{\boldsymbol{\theta}}^2)$, calculate

$$z_t^{f,\mathrm{i}} = \begin{bmatrix} \boldsymbol{\theta}_t^{f,\mathrm{i}} \\ x_t^{f,\mathrm{i}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_{t-1}^{a,\mathrm{i}} \\ g(x_{t-1}^{a,\mathrm{i}}, \alpha_{t-1}^{a,\mathrm{i}}) \end{bmatrix} + \begin{bmatrix} \epsilon_t^{\mathrm{i}} \\ u_t^{\mathrm{i}} \end{bmatrix},$$

and calculate the sample covariance matrix of $z_t^{f,1}, \ldots, z_t^{f,m}$ and denote it by C_t.

(iii) Analysis: For $\mathrm{i} = 1, 2, \ldots, m$, draw $v_t^{\mathrm{i}} \sim N_{d_y}(0, V(\zeta_{y,t-1}^{f,\mathrm{i}}))$ and calculate

$$z_t^{a,\mathrm{i}} = z_t^{f,\mathrm{i}} + \hat{K}_t^{\mathrm{i}}(y_t - H_t^*(\beta_{t-1}^{f,\mathrm{i}}) z_t^{f,\mathrm{i}} - v_t^{\mathrm{i}}) \triangleq z_t^{f,\mathrm{i}} + \hat{K}_t^{\mathrm{i}}(y_t - y_t^{f,\mathrm{i}}),$$

where $\hat{K}_t^{\mathrm{i}} = C_t H_t^{*T}(\beta_{t-1}^{f,\mathrm{i}})(H_t^*(\beta_{t-1}^{f,\mathrm{i}}) C_t H_t^{*T}(\beta_{t-1}^{f,\mathrm{i}}) + V(\zeta_{y,t-1}^{f,\mathrm{i}}))^{-1}$ forms an estimator for the Kalman gain matrix $K_t = S_t H_t^T \ (H_t S_t H_t^T + V_t)^{-1}$ and S_t denotes the covariance matrix of z_t^f.

As discussed in Section 1.2, state-augmented EnKF suffers from a convergence issue, and it can be problematic for estimation of stochastic and multiplicative parameters.

2.3 Markov chain Monte Carlo Ensemble Kalman filter for non-Gaussian Data

Consider the dynamic system (1.3), where the distribution $f(\cdot)$ deviates from Gaussian and the model contains no unknown parameters. This section outlines the MCMC-EnKF algorithm for non-Gaussian data proposed by Katzfuss, Stroud, and Wikle [45] as Algorithm 3.

Algorithm 3 uses MCMC and EnKF in a combined manner, where the Gaussian measurement variables required by the EnKF were imputed using a MCMC algorithm at the imputation step at each stage. As discussed in Section 1.3, MCMC-EnKF doesn't converge to the right filtering distribution and is not scalable with respect to the sample size.

Algorithm 3: MCMC-EnKF for Data Assimilation

(i) Initialization: Start with an initial ensemble $\boldsymbol{x}_0^{a,1}, \boldsymbol{x}_0^{a,2}, \ldots, \boldsymbol{x}_0^{a,m}$ drawn from the prior distribution $\boldsymbol{\pi}(\boldsymbol{x}_0)$, where m denotes the ensemble size.

for *t=1,2,…,T* **do**

(ii) Forecast: For $\mathrm{i} = 1, 2, \cdots, m$, draw $w_t^{\mathrm{i}} \sim N_p(0, U_t)$, calculate

$$\boldsymbol{x}_t^{f,\mathrm{i}} = g(\boldsymbol{x}_{t-1}^{a,\mathrm{i}}) + w_t^{\mathrm{i}}, \tag{2.4}$$

and calculate the sample covariance matrix of $\boldsymbol{x}_t^{a,1}, \boldsymbol{x}_t^{a,2}, \ldots, \boldsymbol{x}_t^{a,m}$ and denote it by C_t.

(iii) Imputation: Draw $\boldsymbol{y}_t^{\mathrm{i}} \sim \boldsymbol{\pi}(\boldsymbol{y}|\boldsymbol{x}_t^{f,\mathrm{i}}, \boldsymbol{z}_t) \propto \rho(\boldsymbol{z}_t|\boldsymbol{y}) f(\boldsymbol{y}|\boldsymbol{x}_t^{f,\mathrm{i}})$, where $\boldsymbol{y}_t^{\mathrm{i}}|\boldsymbol{x}_t^{f,\mathrm{i}} \sim N(H_t\boldsymbol{x}_t^{f,\mathrm{i}}, V_t)$.

(iv) Analysis: For $\mathrm{i} = 1, 2, \cdots, m$, draw $v_t^{\mathrm{i}} \sim N_n(0, V_t)$ and set

$$\boldsymbol{x}_t^{a,\mathrm{i}} = \boldsymbol{x}_t^{f,\mathrm{i}} + \hat{K}_t(\boldsymbol{y}_t^{\mathrm{i}} - H_t\boldsymbol{x}_t^{f,\mathrm{i}} - v_t^{\mathrm{i}}) \stackrel{\Delta}{=} \boldsymbol{x}_t^{f,\mathrm{i}} + \hat{K}_t(\boldsymbol{y}_t^{\mathrm{i}} - \boldsymbol{y}_t^{f,\mathrm{i}}), \tag{2.5}$$

where $\hat{K}_t = C_t H_t^T \left(H_t C_t H_t^T + V_t\right)^{-1}$ form an estimator for Kalman Gain Matrix $K_t = S_t H_t^T \left(H_t S_t H_t^T + V_t\right)^{-1}$ and S_t denotes the covariance matrix of $\boldsymbol{x}_t$

3. LANGEVINIZED ENSEMBLE KALMAN FILTER

In this chapter, we propose the LEnKF algorithm by reformulating the EnKF under the framework of Langevin dynamics, which is a scalable particle filtering algorithm under the big data scenario that the dimension p, the numbers of stages T and the sample sizes N_t's are all very large. The resulting algorithm inherits the forecast-analysis procedure from the EnKF and the use of mini-batch data from the stochastic gradient Langevin-type algorithms. The former makes the new algorithm scalable with respect to the dimension, and the latter makes it scalable with respect to the sample size. We prove that the LEnkF converges to the right filtering distributions in Wasserstein distance under the scenario that the number of stages T and the sample sizes N_t's are large.

The LEnKF algorithm is developed under the dynamic system (1.1), which is of central importance in this chapter as it itself models data assimilation problems with linear measurement equations (studied in Section 3.2.1) and data assimilation problems with nonlinear measurement equations (studied in Section 3.2.2), and the inverse problems (studied in Section 3.1) can also be converted to it via appropriate transformations. Section 3.3 gives detailed proofs about the LEnKF algorithm.

3.1 Langevinized Ensemble Kalman filter for inverse problems

To motivate the development of the LEnKF, we first consider a linear inverse problem and then extend it to nonlinear inverse and data assimilation problems.

3.1.1 Linear inverse problem

Consider a Bayesian inverse problem for the linear regression

$$y = Hx + \eta, \tag{3.1}$$

where $\eta \sim N(0, \Gamma)$ for some covariance matrix Γ, $y \in \mathbb{R}^N$, and $x \in \mathbb{R}^p$ is an unknown continuous parameter vector. To accommodate the case that N is extremely large, we assume that y can be partitioned into $B = N/n$ independent and identically distributed

blocks $\{y_1, \ldots, y_B\}$, where each block is of size n and has the covariance matrix V such that $\Gamma = \text{diag}[V, \cdots, V]$.

Let $\boldsymbol{\pi}(x)$ denote the prior density function of x, which is assumed to be differentiable with respect to x. Let $\boldsymbol{\pi}(x|y)$ denote the posterior distribution. To develop an efficient algorithm for simulating from $\boldsymbol{\pi}(x|y)$, which is scalable with respect to both the sample size N and the dimension p, we reformulate the model (3.1) as a state-space model through subsampling and Langevin diffusion:

$$\begin{aligned} x_t &= x_{t-1} + \epsilon_t \frac{n}{2N} \nabla \log \boldsymbol{\pi}(x_{t-1}) + w_t, \\ y_t &= H_t x_t + v_t, \end{aligned} \tag{3.2}$$

where $w_t \sim N(0, \frac{n}{N}\epsilon_t I_p) = N(0, \frac{n}{N} Q_t)$, i.e., $Q_t = \epsilon_t I_p$, y_t denotes a block randomly drawn from $\{y_1, \ldots, y_B\}$, $v_t \sim N(0, V_t)$ with $V_t = V$, and H_t is a submatrix of H extracted with the corresponding y_t. In the state-space model, at each stage t, the state (i.e., the parameters of model (3.1)) evolves according to an Euler-discretized Langevin equation of the prior distribution, and the measurement varies with subsampling. As shown in Theorem 3.3.1 of Section 3.3, the filtering distribution of the state-space model converges to the target posterior $\boldsymbol{\pi}(x|y)$ as $t \to \infty$, provided that ϵ_t decays to zero in an appropriate rate and the matrix V satisfies some regularity conditions. To simulate from dynamic system (3.2), we propose Algorithm 4, which makes use of both techniques, subsampling and the forecast-analysis procedure and is thus scalable with respect to both the sample size N and the dimension p.

Remark 3.1. *The LEnKF is different from the existing formulation of EnKF for inverse problems [53] in three aspects: (i) It reformulates the Bayesian inverse problem as a state-space model, where the state (i.e., parameters) evolves according to a Langevin diffusion process converging to the prior $\boldsymbol{\pi}(x)$ and the measurement varies with subsampling; the subsampling technique enables the algorithm scalable with respect to the sample size N; (ii) the measurement noise is drawn from a variance inflated distribution $N(0, 2V_t)$ in the analysis step; and (iii) $Q_t = \epsilon_t I_p$ is a designed diagonal matrix with the learning rate $\epsilon_t = O(t^{-\varpi})$ for some $0 < \varpi < 1$.*

Algorithm 4: LEnKF for Linear Inverse Problems

(i) Initialization: Initialize an ensemble $\{x_0^{a,1}, x_0^{a,2}, \ldots, x_0^{a,m}\}$, where m is the ensemble size.

for *t=1,2,…,T* **do**

(ii) Subsampling: Draw without replacement a mini-batch data, denoted by (y_t, H_t), of size n from the full dataset of size N. Set $Q_t = \epsilon_t I_p$, $R_t = 2V_t$, and the Kalman gain matrix $K_t = Q_t H_t^T (H_t Q_t H_t^T + R_t)^{-1}$.

for *i=1,2,…,m* **do**

(iii) Forecast: Draw $w_t^{\mathrm{i}} \sim N_p(0, \frac{n}{N} Q_t)$ and calculate

$$x_t^{f,\mathrm{i}} = x_{t-1}^{a,\mathrm{i}} + \epsilon_t \frac{n}{2N} \nabla \log \boldsymbol{\pi}(x_{t-1}^{a,\mathrm{i}}) + w_t^{\mathrm{i}}. \tag{3.3}$$

(iv) Analysis: Draw $v_t^{\mathrm{i}} \sim N_n(0, \frac{n}{N} R_t)$ and calculate

$$x_t^{a,\mathrm{i}} = x_t^{f,\mathrm{i}} + K_t(y_t - H_t x_t^{f,\mathrm{i}} - v_t^{\mathrm{i}}) \stackrel{\Delta}{=} x_t^{f,\mathrm{i}} + K_t(y_t - y_t^{f,\mathrm{i}}). \tag{3.4}$$

The convergence of Algorithm 4 is established in Theorem 3.3.1. An informal restatement of the theorem is given in Proposition 3.1, which facilitates discussions for the property of the LEnKF algorithm.

Proposition 3.1. *(Convergence of LEnKF) Let x_t^a denote a generic member of the ensemble produced by Algorithm 4 in the analysis step of stage t. If the eigenvalues of $\Sigma_t = \frac{n}{N}(I - K_t H_t)$ are uniformly bounded with respect to t, $\log \boldsymbol{\pi}(x)$ is differentiable with respect to x, and the learning rate $\epsilon_t = O(t^{-\varpi})$ for some $0 < \varpi < 1$, then $\lim_{t\to\infty} W_2(\tilde{\boldsymbol{\pi}}_t, \boldsymbol{\pi}_*) = 0$, where $\tilde{\boldsymbol{\pi}}_t$ denotes the empirical distribution of x_t^a, $\boldsymbol{\pi}_* = \boldsymbol{\pi}(x|y)$ denotes the target posterior distribution, and $W_2(\cdot,\cdot)$ denotes the second-order Wasserstein distance.*

In the proof of Theorem 3.3.1, it is shown that

$$x_t^a = x_{t-1}^a + \frac{\epsilon_t}{2} \Sigma_t \widehat{\nabla} \log \boldsymbol{\pi}(x_{t-1}^a | y_t) + \mathrm{e}_t, \tag{3.5}$$

where e_t is a zero mean Gaussian random error with covariance $\mathrm{Var}(\mathrm{e}_t) = \epsilon_t \Sigma_t$, and $\widehat{\nabla} \log \boldsymbol{\pi}(x_{t-1}^a | y_t) = \frac{N}{n} H_t^T V_t^{-1}(y_t - H_t x_{t-1}^a) + \nabla \log \boldsymbol{\pi}(x_{t-1}^a)$ denotes an unbiased estimate of $\nabla \log \boldsymbol{\pi}(x_{t-1}^a | y_t)$. That is, the LEnKF forms a new type of stochastic gradient Riemannian Langevin dynamics (SGRLD) algorithm (see, e.g., [3], [55], [56]), where the Fisher information matrix is adapted

with the mini-batch of data by noting that $\epsilon_t \Sigma_t = \frac{n}{N}(I - K_t H_t) Q_t$ is exactly the inverse of the Fisher information matrix of the distribution $\boldsymbol{\pi}(x_t^a | x_{t-1}^a, y_t)$. It is known that use of the Fisher information, which rescales parameter updates according to the geometry of the target distribution, can generally improve the convergence of SGMCMC especially when the target distribution exhibits pathological curvature and contains some saddle points (see, e.g., [6], [57]).

Since we set the learning rate $\epsilon_t = O(t^{-\varpi})$ for some $0 < \varpi < 1$ in Algorithm 4, by Theorem 2 of Song, Sun, Ye, *et al.* [58], $\{x_t^a : t = t_0 + 1, \ldots, T\}$ can be treated as equally weighted samples, where t_0 represents the burn-in period. That is, for any Lipschitz function $\rho(x)$, the posterior mean $E_\pi \rho(x) = \int \rho(x) \boldsymbol{\pi}(x|y)$ can be estimated by

$$\widehat{E_\pi \rho(x)} = \frac{1}{(T - t_0)m} \sum_{t=t_0+1}^{T} \sum_{\mathrm{i}=1}^{m} \rho(x_t^{a,\mathrm{i}}), \tag{3.6}$$

which converges to $E_\pi \rho(x)$ in probability as $T \to \infty$. Alternatively, we can apply a weighted averaging scheme as suggested by Chen, Ding, and Carin [59] and Teh, Thiery, and Vollmer [60] for estimating $E_\pi \rho(x)$. As for a conventional SGLD algorithm, we can also set ϵ_t to a small constant. In this case, the convergence of x_t^a to the posterior distribution is up to an approximation error even when $t \to \infty$.

It is interesting to point out that when the dimension of x is high, LEnKF can be much more efficient than directly implementing (3.5). The latter requires an LU-decomposition of Σ_t, which has a computational complexity of $O(p^3)$, in generating the random error e_t. While the LEnKF gets around this issue with the forecast-analysis procedure. As shown in the proof of Theorem 3.3.1, we have $\mathrm{e}_t = (I - K_t H_t) w_t - K_t v_t$ and $\mathrm{Var}(\mathrm{e}_t) = \epsilon_t \Sigma_t$. The computational complexity of the forecast-analysis procedure is $O(\max\{n^2 p, n^3\} + mnp)$ for generating m particles per iteration, where the first term represents the cost for calculating K_t, the second term represents the total cost of m chains for forecasting and analysis, and the cost for calculating K_t is counted as the overhead at each iteration. This procedure is even faster than in the original EnKF algorithm as Q_t is diagonal. For high-dimensional problems, we typically set $n \ll p$, so the total computational complexity of the LEnKF is $O((n^2 p + mnp)T)$, which implies that the algorithm is scalable with respect to both the

sample size N and the dimension p. In contrast, if (3.5) is directly simulated as a SGRLD algorithm, the total computational complexity will be $O((p^3 + np^2)mT)$ for mT iterations, where $O(np^2)$ represents the cost for computing $\Sigma_t \widehat{\nabla} \log \boldsymbol{\pi}(x_{t-1}^a | y_t)$. Here we note that the particles in the same ensemble of the LEnKF are not independent, as they are generated based on the same randomly selected mini-batch of data at each stage. However, they are independent if the full data is used at each iteration.

Finally, we note that conventional SGRLD algorithms lack the scalability necessary for high-dimensional problems as computation of the Fisher information matrix can be very costly. For this reason, preconditioned SGLD [6] approximates the Fisher information matrix using a diagonal matrix estimated based on the current gradient information only. The LEnKF forms a new type of SGRLD algorithm, where the forecast-analysis procedure enables the Fisher information efficiently used in the simulation.

3.1.2 Nonlinear inverse problem

Consider a Bayesian inverse problem for the nonlinear regression

$$y = \mathcal{G}(z) + \eta, \quad \eta \in N(0, \Gamma),$$

where $y = (y_1^T, y_2^T, \ldots, y_B^T)^T$, $\Gamma = \text{diag}[V, V, \ldots, V]$ is a diagonal block matrix, each block V is of size $n \times n$, and $N = Bn$ for some positive constant B. To reformulate the problem in the dynamic system (1.1), we define an augmented state vector by an n-vector γ_t:

$$x_t = \begin{pmatrix} z \\ \gamma_t \end{pmatrix}, \quad \gamma_t = \mathcal{G}_t(z) + u_t, \quad u_t \sim N(0, \alpha V), \tag{3.7}$$

where $\mathcal{G}_t(\cdot)$ is the mean response function for a mini-batch of data drawn at stage t, and $0 < \alpha < 1$ is a pre-specified constant. In this dissertation, α is called the variance splitting proportion.

Let $\boldsymbol{\pi}(z)$ denote the prior density function of z, which is differentiable with respect to z. The conditional distribution of γ_t is $\gamma_t | z \sim N(\mathcal{G}_t(z), \alpha V)$, and then the joint density

function of x_t is $\boldsymbol{\pi}(x_t) = \boldsymbol{\pi}(z)\boldsymbol{\pi}(\gamma_t|z)$. Based on Langevin dynamics, a system identical to (3.2) in symbol can be constructed for the nonlinear inverse problem:

$$\begin{aligned} x_t &= x_{t-1} + \epsilon_t \frac{n}{2N} \nabla_x \log \boldsymbol{\pi}(x_{t-1}) + w_t \\ y_t &= H_t x_t + v_t, \end{aligned} \tag{3.8}$$

where $w_t \sim N(0, \frac{n}{N} Q_t)$, $Q_t = \epsilon_t I_p$, p is the dimension of x_t; $H_t = (0, I)$ such that $H_t x_t = \gamma_t$; $v_t \sim N(0, (1-\alpha)V)$, which is independent of w_t for all t; and y_t is a mini-batch sample randomly drawn from $\{y_1, y_2, \ldots, y_B\}$.

By the variance splitting state augmentation approach, we have successfully converted the nonlinear inverse problem to the dynamic system (1.1). In this approach, z, γ_t and y_t form a hierarchical model, and it is easy to derive that

$$\gamma_t | z, y_t \sim N(\alpha y_t + (1-\alpha)\mathcal{G}_t(z), \alpha(1-\alpha)V), \tag{3.9}$$

which will be used later in justifying efficiency of the LEnKF for nonlinear inverse problems. In particular, if α is close to 1, the conditional variance of γ_t given y_t and z can be much smaller than V.

With the above formulation, the following variant of the LEnKF, i.e., Algorithm 5, can be applied to simulate samples from the posterior distribution $\boldsymbol{\pi}(x|y)$. The posterior samples of $\boldsymbol{\pi}(z|y)$ can be obtained from those of $\boldsymbol{\pi}(x|y)$ via marginalization. Let $x_{t,k}^{a,\mathrm{i}}$ denote the i-th sample obtained at iteration k of stage t. In each stage of the algorithm, a mini-batch sample y_t is drawn at random and the augmented state x_t is updated for $\mathcal{K}$ iterations.

Compared to Algorithm 4, the Algorithm 5 includes a few more iterations at each stage. The added iterations help to drive γ_t towards its conditional equilibrium (3.9). Regarding the convergence of the algorithm, we have the following remark:

Remark 3.2. *In equation (3.10),* $\nabla_x \log \boldsymbol{\pi}(x_{t,k-1})$ *is calculated based on a mini-batch of data:*

$$\nabla_x \log \boldsymbol{\pi}(x_{t,k-1}) = \begin{pmatrix} \nabla_z \log \boldsymbol{\pi}(z_{t,k-1}) + \frac{1}{\alpha}\frac{N}{n} \nabla_z \mathcal{G}_t(z_{t,k-1}) V^{-1} \left(\gamma_{t,k-1} - \mathcal{G}_t(z_{t,k-1})\right) \\ -\frac{1}{\alpha} V^{-1} \left(\gamma_{t,k-1} - \mathcal{G}_t(z_{t,k-1})\right) \end{pmatrix}, \tag{3.12}$$

Algorithm 5: LEnKF for Nonlinear Inverse Problems

(i) Initialization: Initial an ensemble $x_{1,0}^{a,1}, x_{1,0}^{a,2}, \ldots, x_{1,0}^{a,m}$, where m denotes the ensemble size.

for *t=1,2,…,T* **do**

(ii) Subsampling: Draw without replacement a mini-batch sample, denoted by (y_t, H_t), of size n from the full dataset of size N.

for *k=1,2,…,K* **do**

Set $Q_{t,k} = \epsilon_{t,k} I_p$, $R_t = 2(1-\alpha)V$ and the Kalman gain matrix $K_{t,k} = Q_{t,k} H_t^T (H_t Q_{t,k} H_t^T + R_t)^{-1}$.

for *i=1,2,…,m* **do**

(iii) Forecast: Draw $w_{t,k}^{\mathrm{i}} \sim N_p(0, \frac{n}{N} Q_{t,k})$ and calculate

$$x_{t,k}^{f,\mathrm{i}} = x_{t,k-1}^{a,\mathrm{i}} + \epsilon_{t,k} \frac{n}{2N} \nabla \log \boldsymbol{\pi}(x_{t,k-1}^{a,\mathrm{i}}) + w_{t,k}^{\mathrm{i}}, \tag{3.10}$$

where, if $k = 1$, $x_{t,0}^{a,\mathrm{i}} = x_{t-1,\mathcal{K}}^{a,\mathrm{i}}$ for its z-component and $x_{t,0}^{a,\mathrm{i}} = y_t$ for its γ-component.

(iv) Analysis: Draw $v_{t,k}^{\mathrm{i}} \sim N_n(0, \frac{n}{N} R_t)$, and calculate

$$x_{t,k}^{a,\mathrm{i}} = x_{t,k}^{f,\mathrm{i}} + K_{t,k}(y_t - H_t x_{t,k}^{f,\mathrm{i}} - v_{t,k}^{\mathrm{i}}) \stackrel{\Delta}{=} x_{t,k}^{f,\mathrm{i}} + K_{t,k}(y_t - y_{t,k}^{f,\mathrm{i}}). \tag{3.11}$$

where the component $\nabla_z \log \boldsymbol{\pi}(z_{t,k-1}) + \frac{1}{\alpha}\frac{N}{n}\nabla_z \mathcal{G}_t(z_{t,k-1})V^{-1}\left(\gamma_{t,k-1} - \mathcal{G}_t(z_{t,k-1})\right)$ *provides an unbiased estimate of* $\nabla_z \log \boldsymbol{\pi}(z|y)$ *as implied by (3.9). It follows from the standard convergence theory of SGLD that the* z*-component of* $x_{t,k}$ *will converge to* $\boldsymbol{\pi}(z|y)$*, provided that* $\epsilon_{t,k}$ *satisfies the condition:* $\{\epsilon_{t,k}\}$ *is a positive sequence, decreasing in* t *and non-increasing in* k*, such that for any* $k \in \{1, 2, \ldots, \mathcal{K}\}$*,* $\epsilon_{t,k} = O(1/t^{\varsigma})$ *for some* $0 < \varsigma < 1$*.*

From the Kalman gain matrix $K_{t,k}$, it is easy to see that only the γ-component of $x_{t,k}$ is updated at the analysis step. Intuitively, $\gamma_{t,k}$ can converge very fast, as it is updated with the second-order gradient information. Therefore, $\mathcal{K}$ is not necessarily very large. In this dissertation, $\mathcal{K} = 5$ is set as the default. Further, by (3.9), $\nabla_z \log \boldsymbol{\pi}(z_{t,k-1}) + \frac{1}{\alpha}\frac{N}{n}\nabla_z \mathcal{G}_t(z_{t,k-1})V^{-1}(\gamma_{t,k-1} - \mathcal{G}_t(z_{t,k-1}))$ represents an improved gradient estimator compared to the estimator $\nabla_z \log \boldsymbol{\pi}(z_t) + \frac{N}{n}\nabla_z \mathcal{G}_t(z_t)\; V^{-1}\left(y_t - \mathcal{G}_t(z_t)\right)$ used by SGLD in simulating from $\boldsymbol{\pi}(z|y)$. It is easy to show that the two stochastic gradients have the same mean value, but the former has a smaller variance than the latter. More precisely,

$$\mathrm{Var}\left(\frac{1}{\alpha}\frac{N}{n}\nabla_z \mathcal{G}_t(z_t)V^{-1}\left(\gamma_t - \mathcal{G}_t(z_t)\right)\Big|z_t\right) = \frac{1-\alpha}{\alpha}\mathrm{Var}\left(\frac{N}{n}\nabla_z \mathcal{G}_t(z_t)V^{-1}\left(y_t - \mathcal{G}_t(z_t)\right)\Big|z_t\right), \tag{3.13}$$

which implies that for nonlinear inverse problems, the LEnKF represents a variance reduction version of SGLD, and it is potentially more efficient than SGLD if $0.5 < \alpha < 1$ is chosen. In this dissertation, we set $\alpha = 0.9$ as the default and initialized $\gamma_{t,0}$ by y_t at each stage, which enhances the convergence of the simulation.

3.2 Langevinized Ensemble Kalman filter for data assimilation problems

3.2.1 Data assimilation with linear measurement equation

Consider the dynamic system (1.1), for which we assume that at each stage t, y_t can be partitioned into $B_t = N_t/n_t$ blocks such that $y_{t,k} = H_{t,k}x_t + v_{t,k}$, $k = 1, 2, \ldots B_t$, where N_t is the total number of observations at stage t, $y_{t,k}$ is a block of n_t observations randomly drawn from $y_t = \{y_{t,1}, \ldots, y_{t,B_t}\}$, $v_{t,k} \sim N(0, V_t)$ for all k, and $v_{t,k}$'s are mutually independent.

To motivate the development of the algorithm, we first consider the Bayesian formula

$$\boldsymbol{\pi}(x_t|y_{1:t}) = \frac{f(y_t|x_t)\boldsymbol{\pi}(x_t|y_{1:t-1})}{\int f(y_t|x_t)\boldsymbol{\pi}(x_t|y_{1:t-1})dx_t}, \tag{3.14}$$

which suggests that to get the filtering distribution $\boldsymbol{\pi}(x_t|y_{1:t})$, the predictive distribution $\boldsymbol{\pi}(x_t|y_{1:t-1})$ should be used as the prior at stage t. To estimate the gradient $\nabla \log \boldsymbol{\pi}(x_t|y_{1:t-1})$, we employ the following identity established by Song, Sun, Ye, *et al.* [58]:

$$\nabla_\beta \log \boldsymbol{\pi}(\beta \mid D) = \int \nabla_\beta \log \boldsymbol{\pi}(\beta \mid \gamma, D)\boldsymbol{\pi}(\gamma \mid \beta, D)d\gamma, \tag{3.15}$$

where D denotes data, and β and γ denote two parameters of a posterior distribution $\boldsymbol{\pi}(\beta, \gamma|D)$. By the identity, we have

$$\begin{aligned}
\nabla_{x_t} \log \boldsymbol{\pi}(x_t|y_{1:t-1}) &= \int \nabla_{x_t} \log \boldsymbol{\pi}(x_t|x_{t-1}, y_{1:t-1})\boldsymbol{\pi}(x_{t-1}|x_t, y_{1:t-1})dx_{t-1} \\
&= \int \nabla_{x_t} \log \boldsymbol{\pi}(x_t|x_{t-1})\frac{\boldsymbol{\pi}(x_{t-1}|x_t, y_{1:t-1})}{\boldsymbol{\pi}(x_{t-1}|y_{1:t-1})}\boldsymbol{\pi}(x_{t-1}|y_{1:t-1})dx_{t-1} \\
&= \int \nabla_{x_t} \log \boldsymbol{\pi}(x_t|x_{t-1})\omega(x_{t-1}|x_t)\boldsymbol{\pi}(x_{t-1}|y_{1:t-1})dx_{t-1},
\end{aligned} \tag{3.16}$$

where $\omega(x_{t-1}|x_t) = \boldsymbol{\pi}(x_{t-1}|x_t, y_{1:t-1})/\boldsymbol{\pi}(x_{t-1}|y_{1:t-1}) = \boldsymbol{\pi}(x_t|x_{t-1})/\boldsymbol{\pi}(x_t|y_{1:t-1}) \propto \boldsymbol{\pi}(x_t|x_{t-1})$, as $\boldsymbol{\pi}(x_t|y_{1:t-1})$ is a constant for a given particle x_t and the data $\{y_1, y_2, \ldots, y_{t-1}\}$. Therefore, given a set of samples $\mathcal{X}_{t-1} = \{x_{t-1,1}, x_{t-1,2}, \ldots, x_{t-1,m}\}$ drawn from $\boldsymbol{\pi}(x_{t-1}|y_{1:t-1})$, an importance resampling procedure can be employed to draw samples from $\boldsymbol{\pi}(x_{t-1}|x_t, y_{1:t-1})$. The importance resampling procedure can be executed very fast, as calculation of the importance weight $\omega(x_{t-1}|x_t)$ does not involve any data.

With the above formulas and Langevin dynamics, we can construct a dynamic system at stage t as

$$\begin{aligned}
x_{t,k} &= x_{t,k-1} - \epsilon_t \frac{n_t}{2N_t} U_t^{-1}(x_{t,k-1} - g(\tilde{x}_{t-1,k-1})) + w_{t,k}, \\
y_{t,k} &= H_{t,k}x_{t,k} + v_{t,k},
\end{aligned} \tag{3.17}$$

for $k = 1, 2, \ldots$, where $x_{t,0} = g(x_{t-1}) + u_t$; $\tilde{x}_{t-1,k-1}$ denotes an approximate sample drawn from $\boldsymbol{\pi}(x_{t-1}|x_{t,k-1}, y_{1:t-1})$ at iteration k of stage t through the importance resampling procedure; $w_{t,k} \sim N(0, \frac{n_t}{N_t}\epsilon_{t,k} I_p)$, $Q_{t,k} = \epsilon_{t,k} I_p$, and p is the dimension of x_t. Note that $-U_t^{-1}(x_{t,k-1} - g(\tilde{x}_{t-1,k-1}))$ forms an unbiased estimator of $\nabla \log \boldsymbol{\pi}(x_{t,k-1}|y_{1:t-1})$ only in the ideal case that the samples in $\mathcal{X}_{t-1}$ follows from the distribution $\boldsymbol{\pi}(x_{t-1}|y_{1:t-1})$ and the sample size $|\mathcal{X}_{t-1}|$ is sufficiently large. Our theory for the convergence of the algorithm has taken care of the deviations from the ideal case as discussed in Remark 3.3. In practice, $\mathcal{X}_{t-1}$ can be collected at stage $t-1$ from iterations $k_0 + 1, \ldots, \mathcal{K}$, where k_0 denotes the burn-in period and $\mathcal{K}$ denotes the total number of iterations performed at each stage. Applying the LEnKF at stage t leads to the Algorithm 6.

Since $\mathcal{K}$ is usually not allowed to go to ∞ for such a dynamic system, $x_{t,\mathcal{K}}^{a,1}, \ldots, x_{t,\mathcal{K}}^{a,m} \sim \boldsymbol{\pi}(x_t|y_{1:t})$ only approximately holds for each stage t. However, as shown in Theorem 3.3.2, a smaller approximation error at one stage helps to reduce the approximation error at the next stage; and the approximation error becomes negligible as the number of stages increases, if the number of observations at each stage is reasonably large. To facilitate discussion, Theorem 3.3.2 is restated as a proposition in a less rigorous language in what follows.

Proposition 3.2. *Assume that for each stage t, the matrices H_t, U_t and V_t and the state propagator $g(x_t)$ satisfy mild regularity conditions (listed in the Section 3.3). Then there exists a sufficiently large iteration number $\mathcal{K}$ such that* $\limsup_{t\to\infty} W_2(\tilde{\boldsymbol{\pi}}(x_t|y_{1:t}), \boldsymbol{\pi}(x_t|y_{1:t})) = O\left(\frac{1}{\liminf_t N_t}\right)$.

Remark 3.3. *In Step 2, the probability $P(S_{t,k,\mathrm{i}} = s)$ is calculated in a self-normalized importance sampling estimator, which is known to be consistent but biased. The bias has been taken into account in our proof of Theorem 3.3.2 as implied by equation (3.39) in Section 3.3.*

Remark 3.4. *We argue that the convergence rate $O(1/\liminf_t N_t)$ is a reasonable precision. If the state x_t is observed, then it is not difficult to derive that $\boldsymbol{\pi}(x_{t+1}) \sim N(g(x_t), \Gamma_t)$ and $\boldsymbol{\pi}(x_{t+1}|y_{1:t+1}) = \boldsymbol{\pi}(x_{t+1}|y_{t+1}) \sim N(\mu_{t+1}, K)$ for some μ_{t+1} and $K^{-1} = H_t^T \Sigma_t^{-1} H_t + \Gamma_t^{-1}$. When the eigenvalues of Σ_t are bounded by constant, and H is row-wisely independent, then the eigenvalues of K have a lower bound of $O(1/N_t)$. In other words, even when the*

Algorithm 6: LEnKF for Data Assimilation

(i) Initialization: Start with an ensemble $x_{1,0}^{a,1}, x_{1,0}^{a,2}, \ldots, x_{1,0}^{a,m}$ drawn from the prior distribution $\boldsymbol{\pi}(x_1)$, where m denotes the ensemble size.

for *t=1,2,…,T* **do**

Set $\mathcal{X}_t = \emptyset$ and k_0 as the common burn-in period.

for *k = 1, …, K* **do**

(ii) Subsampling: Draw without replacement a mini-batch sample, denoted by $(y_{t,k}, H_{t,k})$, of size n_t from the full dataset of size N_t. Set $Q_{t,k} = \epsilon_{t,k} I_p$, $R_t = 2V_t$, and the Kalman gain matrix $K_{t,k} = Q_{t,k} H_{t,k}^T (H_{t,k} Q_{t,k} H_{t,k}^T + R_t)^{-1}$.

for *i= 1, …, m* **do**

(iii) Importance resampling: If $t > 1$, calculate importance weights $\omega_{t,k-1,\mathrm{j}}^{\mathrm{i}} = \boldsymbol{\pi}(x_{t,k-1}^{a,\mathrm{i}} | x_{t-1,\mathrm{j}}) = \phi(x_{t,k-1}^{a,\mathrm{i}} : g(x_{t-1,\mathrm{j}}), U_t)$ for $\mathrm{j} = 1, 2, \ldots, |\mathcal{X}_{t-1}|$, where $\phi(\cdot)$ denotes a Gaussian density, and $x_{t-1,\mathrm{j}} \in \mathcal{X}_{t-1}$ denotes the jth sample in $\mathcal{X}_{t-1}$; if $k = 1$, set $x_{t,0}^{a,\mathrm{i}} = g(x_{t-1,\mathcal{K}}^{a,\mathrm{i}}) + u_t^{a,\mathrm{i}}$ and $u_t^{a,\mathrm{i}} \sim N(0, U_t)$. Resample $s \in \{1, 2, \ldots, |\mathcal{X}_{t-1}|\}$ with a probability $\propto \omega_{t,k-1,s}^{\mathrm{i}}$, i.e., $P(S_{t,k,\mathrm{i}} = s) = \omega_{t,k-1,s}^{\mathrm{i}} / \sum_{\mathrm{j}=1}^{|\mathcal{X}_{t-1}|} \omega_{t,k-1,\mathrm{j}}^{\mathrm{i}}$, and denote the sample drawn from $\mathcal{X}_{t-1}$ by $\tilde{x}_{t-1,k-1}^{\mathrm{i}}$.

(iv) Forecast: Draw $w_{t,k}^{\mathrm{i}} \sim N_p(0, \frac{n}{N} Q_{t,k})$. If $t = 1$, set

$$x_{t,k}^{f,\mathrm{i}} = x_{t,k-1}^{a,\mathrm{i}} - \epsilon_{t,k} \frac{n_t}{2N_t} \nabla \log \boldsymbol{\pi}(x_{t,k-1}^{a,\mathrm{i}}) + w_{t,k}^{\mathrm{i}}, \tag{3.18}$$

where $\boldsymbol{\pi}(\cdot)$ denotes the prior distribution of x_1. If $t > 1$, set

$$x_{t,k}^{f,\mathrm{i}} = x_{t,k-1}^{a,\mathrm{i}} - \epsilon_{t,k} \frac{n_t}{2N_t} U_t^{-1} (x_{t,k-1}^{a,\mathrm{i}} - g(\tilde{x}_{t-1,k-1}^{\mathrm{i}})) + w_{t,k}^{\mathrm{i}}. \tag{3.19}$$

(v) Analysis: Draw $v_{t,k}^{\mathrm{i}} \sim N_n(0, \frac{n}{N} R_t)$ and set

$$x_{t,k}^{a,\mathrm{i}} = x_{t,k}^{f,\mathrm{i}} + K_{t,k}(y_{t,k} - H_{t,k} x_{t,k}^{f,\mathrm{i}} - v_{t,k}^{\mathrm{i}}) \stackrel{\Delta}{=} x_{t,k}^{f,\mathrm{i}} + K_{t,k}(y_{t,k} - y_{t,k}^{f,\mathrm{i}}). \tag{3.20}$$

(vi) Sample collection: If $k > k_0$, add the sample $x_{t,k}^{a,\mathrm{i}}$ into the set $\mathcal{X}_t$.

state x_t is known, the filtering distribution $\boldsymbol{\pi}(x_{t+1}|y_{1:t+1})$ has a variation scale of at least $O(1/\sqrt{N_t})$. Compared to this variation scale, the estimation inaccuracy of $W_2(\tilde{\boldsymbol{\pi}}_{t+1}, \boldsymbol{\pi}_{t+1})$ is really negligible.

It is known that weight degeneracy is an inherent default of sequential importance sampling (SIS) (also known as sequential Monte Carlo), especially when the dimension of the system is high. When it occurs, the importance weights concentrates on a few samples, the effective sample size is low, and the resulting importance sampling estimate is heavily biased. Fortunately, the LEnKF is essentially immune to this issue. In LEnKF, the importance resampling procedure is to draw from $\mathcal{X}_{t-1}$ a particle which matches a given particle x_t in state propagation such that the gradient $\nabla_{x_t} \log \boldsymbol{\pi}(x_t|y_{1:t-1})$ can be reasonably well estimated, which is then combined with the gradient of the likelihood function of the new data y_t to have x_t updated. By equation (3.14), $\boldsymbol{\pi}(x_t|y_{1:t-1})$ works as the prior distribution of x_t for the filtering distribution $\boldsymbol{\pi}(x_t|y_{1:t})$. Therefore, the effect of the importance resampling procedure on the performance of the algorithm is limited if the sample size of y_t is reasonably large at each stage t. In contrast, the importance resampling procedure in SIS is to draw a particle from $\mathcal{X}_{t-1}$ and treat the particle as from the filtering distribution $\boldsymbol{\pi}(x_t|y_{1:t})$. For high-dimensional problems, the overlap between the high density regions of the neighboring stage filtering distributions can be very small, which naturally causes the weight degeneracy issue.

In summary, the importance resampling step of the LEnKF aims to draw a sample for the prior $\boldsymbol{\pi}(x_t|y_{1:t-1})$ used at each stage t, while that of SIS aims to draw a sample for the posterior $\boldsymbol{\pi}(x_t|y_{1:t})$. In consequence, the LEnKF is less bothered by the weight degeneracy issue.

3.2.2 Data assimilation with nonlinear measurement equation

To apply the LEnKF to the case that the measurement equation is nonlinear, the variance splitting state augmentation approach proposed in Section 3.1.2 can be used. For simplicity,

we describe the algorithm under the full data scenario. Extension of the algorithm to the mini-batch scenario is straightforward. Consider the dynamic system

$$\begin{aligned} z_t &= g(z_{t-1}) + u_t, \quad u_t \sim N(0, U_t), \\ y_t &= h(z_t) + \eta_t, \quad \eta_t \sim N(0, \Gamma_t), \end{aligned} \tag{3.21}$$

where both $g(\cdot)$ and $h(\cdot)$ are nonlinear. As for nonlinear inverse problems, we can augment the state z_t by γ_t at each stage t, where

$$\gamma_t = h(z_t) + \xi_t, \quad \xi_t \sim N(0, \alpha_t \Gamma_t),$$

for some constant $0 < \alpha_t < 1$. Let $x_t^T = (z_t^T, \gamma_t^T)$ and thus $\boldsymbol{\pi}(x_t|z_{t-1}, y_{1:t-1}) = \boldsymbol{\pi}(z_t|z_{t-1}, y_{1:t-1})\boldsymbol{\pi}(\gamma_t|z_t)$. Similar to system (3.17), at stage t, we have the following dynamic system

$$\begin{aligned} x_{t,k} &= x_{t,k-1} + \frac{\epsilon_t}{2} \nabla_z \log \boldsymbol{\pi}(x_{t,k-1}|\tilde{x}_{t-1,k-1}, y_{1:t-1}) + w_{t,k} \\ y_{t,k} &= H_t x_{t,k} + v_{t,k}, \end{aligned} \tag{3.22}$$

where $\tilde{x}_{t-1,k-1}$ denotes a sample drawn from $\boldsymbol{\pi}(x_{t-1}|x_{t,k-1}, y_{1:t-1})$, $x_{t,k}$ denotes the estimate of x_t obtained at iteration k of stage t, $y_{t,k} = y_t$ for all $k = 1, 2, \ldots, \mathcal{K}$, $H_t = (0, I)$ such that $H_t x_t = \gamma_t$, $w_{t,k} \sim N(0, \epsilon_{t,k} I_p)$, p is the dimension of x_t, and $v_{t,k} \sim N(0, (1 - \alpha_t)\Gamma_t)$. Algorithm 6 can then be applied to simulate samples from $\boldsymbol{\pi}(x_t|y_{1:t})$ and thus the samples from $\boldsymbol{\pi}(z_t|y_{1:t})$ can be obtained by marginalization. In mapping the ensemble from stage $t-1$ to stage t, z_t can be set according to the state evolution equation given in system (3.21), while γ_t can be set to y_t. The convergence of the algorithm follows from Theorem 3.3.2 and Remark 3.2.

3.3 Convergence analysis

In this section, we prove the convergence of Algorithm 4 and Algorithm 6. We are interested in studying the convergence of the LEnKF in Wasserstein distance. The r-th order Wasserstein distance between two probability measures μ and ν is defined by

$$W_r(\mu,\nu) = \left(\inf_{\pi\in\Pi(\mu,\nu)}\int_{\mathbb{X}\times\mathbb{X}} d(x,y)^r d\boldsymbol{\pi}(x,y)\right)^{1/r} = \inf_{\pi\in\Pi(\mu,\nu)}\{E_\pi d(X,Y)^r\}^{1/r},$$

where $\Pi(\mu,\nu)$ denotes the collection of all probability measures on $\mathbb{X}\times\mathbb{X}$ with marginals μ and ν respectively.

3.3.1 Convergence of Algorithm 4

Theorem 3.3.1. *Let $\Sigma_t = \frac{n}{N}(I - K_t H_t)$. If $\lambda_l \le \inf_t \lambda_{\min}(\Sigma_t) \le \sup_t \lambda_{\max}(\Sigma_t) \le \lambda_u$ holds for some positive constants λ_l and λ_u, the log-prior density function $\log \boldsymbol{\pi}(x)$ is differentiable with respect to x, and the learning rate $\epsilon_t = O(t^{-\varpi})$ for some $0 < \varpi < 1$, then $\lim_{t\to\infty} W_2(\tilde{\boldsymbol{\pi}}_t, \boldsymbol{\pi}_*) = 0$, where $\tilde{\boldsymbol{\pi}}_t$ denotes the empirical distribution of x_t^a, $\boldsymbol{\pi}_* = \boldsymbol{\pi}(x|y)$ denotes the target posterior distribution, and $W_2(\cdot,\cdot)$ denotes the second-order Wasserstein distance.*

Proof. First, we consider the Kalman gain matrix $K_t = Q_t H_t^T (R_t + H_t Q H_t^T)^{-1}$, which, with some algebra, can be shown

$$K_t = (I - K_t H_t) Q_t H_t^T R_t^{-1} = (H_t^T R_t^{-1} H_t + Q_t^{-1})^{-1} H_t^T R_t^{-1}. \tag{3.23}$$

Let $\mu_t = E(x_t^f | x_{t-1}^a) = x_{t-1}^a + \delta_t$, where $\delta_t = \epsilon_t \frac{n}{2N} \nabla \log \boldsymbol{\pi}(x_{t-1}^a)$. Therefore, $x_t^f = \mu_t + w_t$.

Taking conditional expectation on both sides of equation (3.4), we have

$$E(x_t^a | x_{t-1}^a) = \mu_t + K_t(y_t - H_t\mu_t) = x_t^f + K_t(y_t - H_t x_t^f) - (I - K_t H_t) w_t. \tag{3.24}$$

With the identity (3.23), (3.24) can be further written as

$$\begin{aligned} E(x_t^a|x_{t-1}^a) &= x_{t-1}^a + \delta_t + K_t(y_t - H_t x_{t-1}^a - H_t\delta_t) = x_{t-1}^a + K_t(y_t - H_t x_{t-1}^a) + (I - K_tH_t)\delta_t \\ &= x_{t-1}^a + (I - K_tH_t)Q_tH_t^TR_t^{-1}(y_t - H_t x_{t-1}^a) + (I - K_tH_t)\delta_t \\ &= x_{t-1}^a + (I - K_tH_t)Q_t\left[H_t^TR_t^{-1}(y_t - H_t x_{t-1}^a) + Q_t^{-1}\delta_t\right] \\ &= x_{t-1}^a + \frac{n}{2N}(I - K_tH_t)Q_t\left[\frac{N}{n}H_t^TV_t^{-1}(y_t - H_t x_{t-1}^a) + \nabla \log \boldsymbol{\pi}(x_{t-1}^a)\right], \\ &= x_{t-1}^a + \frac{\epsilon_t}{2}\Sigma_t\left[\frac{N}{n}H_t^TV_t^{-1}(y_t - H_t x_{t-1}^a) + \nabla \log \boldsymbol{\pi}(x_{t-1}^a)\right], \end{aligned} \tag{3.25}$$

by defining $\Sigma_t = \frac{n}{N}(I - K_tH_t)$ and by noting $Q_t = \epsilon_t I_p$ and $R_t = 2V_t$.

For the LEnKF, the difference between equation (3.4) and equation (3.24) is

$$\mathrm{e}_t = (I - K_tH_t)w_t - K_tv_t = w_t - K_t(H_tw_t + v_t),$$

for which the mean $E(\mathrm{e}_t) = 0$ and the covariance

$$\begin{aligned} \mathrm{Var}(\mathrm{e}_t) &= \frac{n}{N}Q_t + K_t(\frac{n}{N}H_tQ_tH_t^T + \frac{n}{N}R_t)K_t^T - 2\frac{n}{N}K_tH_tQ_t = \frac{n}{N}\left[Q_t + K_tH_tQ_t - 2K_tH_tQ_t\right] \\ &= \frac{n}{N}(I - K_tH_t)Q_t = \epsilon_t\Sigma_t, \end{aligned}$$

where the second equality holds due to the symmetry of Q_t and R_t and the identity $K_t(H_tQ_tH_t^T + R_t)K_t^T = K_t(H_tQ_tH_t^T + R_t)(H_tQ_t^TH_t^T + R_t^T)^{-1}H_tQ_t^T = K_tH_tQ_t$. Then, by (3.25), we have

$$\begin{aligned} x_t^a &= x_t^f + K_t\left[y_n - H_tx_t^f - v_t\right] \\ &= x_{t-1}^a + \frac{\epsilon_t}{2}\Sigma_t\left[\frac{N}{n}H_t^TV_t^{-1}(y_t - H_t x_{t-1}^a) + \nabla \log \boldsymbol{\pi}(x_{t-1}^a)\right] + \mathrm{e}_t, \end{aligned} \tag{3.26}$$

where $\frac{N}{n}H_t^TV^{-1}(y_t - H_t x_{t-1}^a)$ represents an unbiased estimator for the gradient of the log-likelihood function, and $\nabla \log \boldsymbol{\pi}(x_{t-1}^a)$ represents the gradient of the log-prior density function. By Corollary 1 (with $\eta = 0$), we have $\lim_{t\to\infty} W_2(\tilde{\boldsymbol{\pi}}_t, \boldsymbol{\pi}_*) = 0$. For this algorithm, it is easy to see that (3.30) is satisfied, for which the bias factor $\eta = 0$ as $\frac{N}{n}H_t^TV_t^{-1}(y_t - H_t x_{t-1}^a) +$

$\nabla \log \boldsymbol{\pi}(x_{t-1}^a)$ forms an unbiased estimator of $\nabla \log \boldsymbol{\pi}(x|y)$ at x_{t-1}^a, and the variance of the estimation error is upper bounded by a quadratic function of $\|x_{t-1}^a\|$. □

3.3.2 Convergence of Algorithm 6

Let $\boldsymbol{\pi}_t = \boldsymbol{\pi}(x_t|y_{1:t})$ denote the filtering distribution at stage t, and let $\tilde{\boldsymbol{\pi}}_t$ denote the marginal distribution of $x_{t,\mathcal{K}}^{a,\mathrm{i}}$ generated by Algorithm 6 at iteration $\mathcal{K}$ of stage t. The following conditions are assumed for the dynamic system (1.1) at each stage t:

Assumption 3.1. *$\boldsymbol{\pi}_t$ is s_t-strongly log-concave:*

$$f(x_t) - f(x_t^*) - \nabla f(x_t^*)^T (x_t - x_t^*) \geq \frac{s_t}{2} \|x_t - x_t^*\|_2^2, \quad \forall x_t, x_t^* \in \mathbb{R}_t^p, \tag{3.27}$$

where $f(x_t) = -\log \boldsymbol{\pi}(x_t|y_{1:t}) = -\log \boldsymbol{\pi}_t$, and s_t is a positive number satisfying $s_t \geq cN_t$ for some constant $c > 0$.

Assumption 3.2. *$\log(\boldsymbol{\pi}_t)$ is S_t-gradient Lipschitz continuous:*

$$\|\nabla f(x_t) - \nabla f(x_t^*)\|_2 \leq S_t \|x_t - x_t\|_2, \quad \forall x_t, x_t^* \in \mathbb{R}_t^p. \tag{3.28}$$

where S_t is a positive number satisfying $S_t \leq CN_t$ for some constant $C > 0$. Note that we must have $s_t \leq S_t$.

Assumption 3.3. *Let $\Sigma_{t,k} = \frac{n}{N}(I - K_{t,k}H_{t,k})$, and assume that $\lambda_{t,l} \leq \inf_k \lambda_{\min}(\Sigma_{t,k}) \leq \sup_k \lambda_{\max}(\Sigma_{t,k}) \leq \lambda_{t,u}$ for some $\lambda_{t,l}$ and $\lambda_{t,u}$, where $\lambda_{\max}(\cdot)$ and $\lambda_{\min}(\cdot)$ denote the largest and smallest eigenvalues, respectively.*

Assumption 3.4. *The stochastic error induced by the sub-sampling procedure has a bounded variance, i.e., $\forall x \in \mathbb{R}_t^p$,*

$$E[\|(N/n)H_{t,k}^T V_t^{-1}(y_{t,k} - H_{t,k}x) - H_t^T \Gamma_t^{-1}(y_t - H_t x)\|^2] \leq \sigma_{t,s}^2 (p + \|x\|^2),$$

for some constant $\sigma_{t,s}^2 > 0$, where the expectation is with respect to random sub-sampling.

Assumption 3.5. *The state propagator $g(x_t)$ is l-Lipschitz and bounded by M_g (i.e., $\sup_x \|g(x)\| \leq M_g$), and $\lambda_{\min}(U_t) \geq \lambda_{t,s} > 0$ for some positive constant $\lambda_{t,s}$.*

Assumption 3.6. *There exist some constant M such that $W_2(\nu_{t+1}, \boldsymbol{\pi}_{t+1}) \leq M$ for all $t \geq 0$, where $\nu_{t+1}(x_{t+1}) = \int \boldsymbol{\pi}(x_{t+1}|x_t)\boldsymbol{\pi}_t(x_t)dx_t$ is the ideal stage initial distribution of $x^{a,\mathrm{i}}_{t+1,0}$ for $t \geq 1$, and ν_1 is the initial distribution used at stage 1. Similarly, we define $\tilde{\nu}_{t+1}(x_{t+1}) = \int \boldsymbol{\pi}(x_{t+1}|x_t)\tilde{\boldsymbol{\pi}}_t(x_t)dx_t$ to be the practical stage initial distribution of $x^{a,\mathrm{i}}_{t+1,0}$ for $t \geq 1$.*

Remark 3.5. *Log-concavity and strong log-concavity are preserved by products and marginalization [61]. If the prior density $\boldsymbol{\pi}(x_1)$ is log-concave, the state transition density $\boldsymbol{\pi}(x_t|x_{t-1})$ is log-concave with respect to both x_t and x_{t-1} for each stage t, and the emission density $\boldsymbol{\pi}(y_t|x_t)$ is λ_t-strongly-log-concave for each stage t where λ_t is the smallest eigenvalue of $H_t^T\Gamma_t^{-1}H_t$, then Assumption 3.1 holds with $s_t = \lambda_t$. Furthermore, by Brascamp-Lieb inequality [62], we must have that $\boldsymbol{\pi}_t$ has finite variance, that is*

$$p\sigma^2_{t,v} := E_{\boldsymbol{\pi}_t}\|X - E(X)\|^2 \leq p/s_t. \tag{3.29}$$

Strongly log-concave conditions are commonly used in the theoretical study of Langevin Monte Carlo (see e.g., [63] and [64]). These conditions potentially can be relaxed following the work of Durmus and Moulines [65].

Remark 3.6. *Assumption 3.6 says the ideal initialization distribution in each stage is not too bad, and essentially, it actually requires that the data are coherent to the state space model (1.1) given in the main text, in the sense that the predictive distribution based on $y_{1:t}$ ($\boldsymbol{\pi}(x_{t+1}|y_{1:t})$) and estimated parameter based on y_{t+1} ($\hat{x}_{t+1} = (H_{t+1}^TH_{t+1})^{-1}H_{t+1}^Ty_{t+1}$) are close.*

Lemma 3.1. *Let μ and ν be two distribution laws on $\mathbb{R}^p$, and let f be an L-Lipschitz continuous function, then*

$$\left\| \int f(x)d\mu(x) - \int f(x)d\nu(x) \right\| \leq LW_2(\mu, \nu).$$

Proof. By definition of Wasserstein distance, there exist random variables X_1 and X_2, whose marginal distributions follow μ and ν respectively, such that $\|X_1 - X_2\|_{L_2} = (E\|X_1 - X_2\|_2^2)^{1/2} = W_2(\mu, \nu)$.

$$\begin{aligned}&\left\| \int f(x)d\mu(x) - \int f(x)d\nu(x) \right\| = \|Ef(X_1) - Ef(X_2)\| \leq E\|f(X_1) - f(X_2)\| \\ \leq & EL\|X_1 - X_2\|_2 = LE\sqrt{\|X_1 - X_2\|_2^2} \leq L\sqrt{E\|X_1 - X_2\|_2^2} = LW_2(\mu, \nu).\end{aligned}$$

□

Lemma 3.2. *Under the Assumption 3.5, we have*

$$W_2(\tilde{\nu}_{t+1}, \nu_{t+1}) \leq lW_2(\boldsymbol{\pi}_t, \tilde{\boldsymbol{\pi}}_t)$$

Proof. By definition of Wasserstein distance, there exist random variables X_1 and X_2, whose marginal distributions are $\boldsymbol{\pi}_t$ and $\tilde{\boldsymbol{\pi}}_t$ respectively, and $E(\|X_1 - X_2\|_2^2) = W_2^2(\tilde{\boldsymbol{\pi}}_t, \boldsymbol{\pi}_t)$. Define $Y_1 = g(X_1)+u$, and $Y_2 = g(x_2)+u$, where $u \sim N(0, U_{t+1})$ such that the marginal distributions of Y_1 and Y_2 are ν_{t+1} and $\tilde{\nu}_{t+1}$ respectively. Then,

$$W_2^2(\tilde{\nu}_{t+1}, \nu_{t+1}) \leq E\|Y_1 - Y_2\|_2^2 = E\|g(X_1) - g(X_2)\|_2^2 \leq El^2\|X_1 - X_2\|_2^2 = l^2W_2^2(\boldsymbol{\pi}_t, \tilde{\boldsymbol{\pi}}_t).$$

□

Lemma 3.3. *If f is an L-Lipschitz continuous function, then $E\|f(X) - E(f(X))\|_2^2 \leq L^2E\|X - E(X)\|_2^2$.*

Proof. Let X_1 and X_2 be two independent copies of X. Then

$$\begin{aligned}&E\|f(X) - E(f(X))\|_2^2 = (1/2)E\|f(X_1) - f(X_2)\|_2^2 \leq (1/2)E(L\|X_1 - X_2\|_2)^2 \\ \leq & L^2(1/2)E(\|X_1 - X_2\|_2)^2 = L^2E\|X - E(X)\|_2^2.\end{aligned}$$

□

Lemma 3.4. *Let $X \sim \mu$ and $Y \sim \nu$, then*

$$E\|Y - E(Y)\|_2^2 \leq E\|X - E(X)\|_2^2 + W_2^2(\mu,\nu) + 2W_2(\mu,\nu)\sqrt{E\|X - E(X)\|_2^2}.$$

Proof. By definition of Wasserstein metric, we can assume that X and Y satisfy $\|X-Y\|_{L_2} = (E\|X-Y\|_2^2)^{1/2} = W_2(\mu,\nu)$. Without loss of generality, we also assume that EX, the mean of measure μ, is 0. Then

$$\begin{aligned}
&[E\|Y - E(Y)\|_2^2 - E\|X\|_2^2] - W_2^2(\mu,\nu) \\
=&EY^TY - (EY)^T(EY) - EX^TX - EX^TX - EY^TY + 2EX^TY \\
=&2EX^TY - 2EX^TX - (EY)^T(EY) \leq 2EX^T(Y-X) \leq 2E\|X\|_2\|Y-X\|_2 \\
\leq&2\sqrt{E\|X\|_2^2 E\|Y-X\|_2^2} = 2W_2(\mu,\nu)\sqrt{E\|X\|_2^2}.
\end{aligned}$$

□

Lemma 3.5. *Let $X \sim \mu$ and $Y \sim \nu$, then*

$$E\|Y\|^2 \leq E\|X\|^2 + W_2^2(\mu,\nu) + 2W_2(\mu,\nu)\sqrt{E\|X\|^2},$$

where $W_2(\cdot,\cdot)$ denotes the second order Wasserstein distance.

Proof. By definition of Wasserstein metric, W.O.L.G, we can assume that X and Y satisfy $E\|X-Y\|^2 = W_2^2(\mu,\nu)$. Then

$$\begin{aligned}
&[E\|Y\|^2 - E\|X\|^2] - W_2^2(\mu,\nu) \\
=&EY^TY - EX^TX - EX^TX - EY^TY + 2EX^TY \\
=&2EX^TY - 2EX^TX = 2EX^T(Y-X) \leq 2E\|X\|\|Y-X\| \\
\leq&2\sqrt{E\|X\|^2 E\|Y-X\|^2} = 2W_2(\mu,\nu)\sqrt{E\|X\|^2}.
\end{aligned}$$

□

Lemma 3.6 is a generalization of Theorem 4 of Dalalyan and Karagulyan [63], as well as a generalization of Lemma S2 of Song, Sun, Ye, *et al.* [58].

Lemma 3.6. *Let x_k and x_{k+1} be two random vectors in $\mathbb{R}^p$ satisfying*

$$x_{k+1} = x_k - \epsilon\Sigma[\nabla f(x_k) + \zeta_k] + \sqrt{2\epsilon}\mathrm{e}_{k+1},$$

where $\mathrm{e}_{k+1} \sim N(0, \Sigma)$, and ζ_k denotes the random error of the gradient estimate which can depend on x_k. Let $\boldsymbol{\pi}_k$ be the distribution of x_k, and let $\boldsymbol{\pi}_* \propto \exp\{-f\}$ be the target distribution. Suppose that ζ_k satisfies

$$\|E(\zeta_k|x_k)\|^2 \le \eta^2 p, \quad E[\|\zeta_k - E(\zeta_k|x_k)\|^2] \le \sigma_1^2 p + \sigma_2^2\|x_k\|^2, \tag{3.30}$$

for some constants η and σ, and ζ_k's are independent of e_{k+1}'s. If the function f is s-strongly convex and S-gradient-Lipschitz, $\lambda_{\min}(\Sigma) = \lambda_l$, $\lambda_{\max}(\Sigma) = \lambda_u$, and the learning rate $\epsilon \le 2/(s\lambda_l + S\lambda_u)$, then

$$\begin{aligned} W_2^2(\boldsymbol{\pi}_{k+1}, \boldsymbol{\pi}_*) \le \Big[(1 - \lambda_l s\epsilon + \sqrt{2}\sigma_2\lambda_u\epsilon)W_2(\boldsymbol{\pi}_k, \boldsymbol{\pi}_*) + 1.65S(\lambda_u^3\epsilon^3 p)^{1/2} + \epsilon\eta\lambda_u\sqrt{p}\Big]^2 \\ + \epsilon^2\sigma_1^2\lambda_u^2 p + 2\epsilon^2\sigma_2^2\lambda_u^2 pV. \end{aligned} \tag{3.31}$$

where $V = \int \|x\|^2\boldsymbol{\pi}_*(x)dx$.

Proof. First of all, the updating iteration can be rewritten as:

$$\tilde{x}_{k+1} = \tilde{x}_k - \epsilon[\nabla\tilde{f}(\tilde{x}_k) + \tilde{\zeta}_k] + \sqrt{2\epsilon}\tilde{\mathrm{e}}_{k+1},$$

where $\tilde{f}(x) = f(\Sigma^{1/2}x)$, $\tilde{x}_k = \Sigma^{-1/2}x_k$, $\tilde{\zeta}_k = \Sigma^{1/2}\zeta_k$ and $\tilde{\mathrm{e}}_{k+1} \sim N(0, I)$.

Let $\tilde{\boldsymbol{\pi}}_*$ denote the distribution $\tilde{\boldsymbol{\pi}}_* \propto \exp\{-\tilde{f}\}$. It is easy to see that the distribution $\tilde{\boldsymbol{\pi}}_*$ is $s\lambda_l$-strongly log-concave and $S\lambda_u$-gradient-Lipschitz. In addition, $\tilde{\zeta}_k$ satisfies

$$\begin{aligned} \|E(\tilde{\zeta}_k|\tilde{x}_k)\|^2 = \|\Sigma^{1/2}E(\zeta_k|x_k)\|^2 \le \lambda_u\eta^2 p \\ E[\|\tilde{\zeta}_k - E(\tilde{\zeta}_k|\tilde{x}_k)\|^2] = E[\|\Sigma^{1/2}\zeta_k - E(\Sigma^{1/2}\zeta_k|x_k)\|^2] \le \lambda_u\sigma_1^2 p + \lambda_u\sigma_2^2\|\Sigma^{1/2}\tilde{x}_k\|^2, \end{aligned} \tag{3.32}$$

Let L_t be the stochastic process defined by $dL_t = -(\Sigma^{1/2}\nabla f(\Sigma^{1/2}L_t))dt + \sqrt{2}dW_t$ with initialization $L_0 \sim \tilde{\boldsymbol{\pi}}_*$ (hence $L_t \sim \tilde{\boldsymbol{\pi}}_*$). Define $\Delta_2 = L_\epsilon - \tilde{x}_{t+1}$ and $\Delta_1 = L_0 - \tilde{x}_t$. Then, by

the same arguments used in the proof of Proposition 2 in Dalalyan and Karagulyan [63]. we have

$$\begin{aligned}&\|\Sigma^{1/2}\Delta_2\|_{L_2}^2\\ \leq&\{\|\Sigma^{1/2}\Delta_1-\epsilon\Sigma^{1/2}U\|_{L_2}+\|\Sigma^{1/2}W\|_{L_2}+\epsilon\|\Sigma^{1/2}E(\tilde{\zeta}_k|\tilde{x}_k)\|_{L_2}\}^2+\epsilon^2\|\Sigma^{1/2}(\tilde{\zeta}_k-E(\tilde{\zeta}_k|\tilde{x}_k))\|_{L_2}^2,\end{aligned} \tag{3.33}$$

where $W=\int_0^\epsilon(\nabla\tilde{f}(L_t)-\nabla\tilde{f}(L_0))dt$ and $U=\nabla\tilde{f}(\tilde{x}_k+\Delta_1)-\nabla\tilde{f}(\tilde{x}_k)$.

By Lemma 4 of Dalalyan and Karagulyan [63], $\|W\|_{L_2}\leq 0.5\sqrt{\epsilon^4S^3\lambda_u^3p}+(2/3)\sqrt{2\epsilon^3p}S\lambda_u\leq 1.65S\lambda_u(\epsilon^3p)^{1/2}$. By similar argument of Lemma 2 of Dalalyan and Karagulyan [63], we can show that $\|\Sigma^{1/2}\Delta_1-\epsilon\Sigma^{1/2}U\|_2\leq\rho\|\Sigma^{1/2}\Delta_1\|_2$, where $\rho=\max(1-s\lambda_l\epsilon,S\lambda_u\epsilon-1)=1-s\lambda_l\epsilon$. Combining with (3.32) and (3.33), we have that

$$\|\Sigma^{1/2}\Delta_2\|_{L_2}^2\leq\{\rho\|\Sigma^{1/2}\Delta_1\|_{L_2}+1.65S(\lambda_u^3\epsilon^3p)^{1/2}+\epsilon\lambda_u\eta\sqrt{p}\}^2+\epsilon^2\lambda_u^2\sigma_1^2p+\epsilon^2\lambda_u^2\sigma_2^2E\|x_k\|^2,$$

which further implies

$$W_2^2(\boldsymbol{\pi}_{k+1},\boldsymbol{\pi}_*)\leq\{(1-s\lambda_l\epsilon)W_2^2(\boldsymbol{\pi}_{k+1},\boldsymbol{\pi}_*)+1.65S(\lambda_u^3\epsilon^3p)^{1/2}+\epsilon\lambda_u\eta\sqrt{p}\}^2+\epsilon^2\lambda_u^2\sigma_1^2p+\epsilon^2\lambda_u^2\sigma_2^2E\|x_k\|^2.$$

By Lemma 3.5, $E\|x_k\|^2\leq(W_2(\boldsymbol{\pi}_k,\boldsymbol{\pi}_*)+\sqrt{V})^2$, we can derive that

$$\begin{aligned}W_2^2(\boldsymbol{\pi}_{k+1},\boldsymbol{\pi}_*)\leq&\Big[(1-s\lambda_l\epsilon)W_2(\boldsymbol{\pi}_k,\boldsymbol{\pi}_*)+1.65S(\lambda_u^3\epsilon^3p)^{1/2}+\epsilon\eta\lambda_u\sqrt{p}\Big]^2\\&+\epsilon^2\sigma_1^2\lambda_u^2p+\epsilon^2\sigma_2^2\lambda_u^2(W_2(\boldsymbol{\pi}_k,\boldsymbol{\pi}_*)+\sqrt{V})^2\\\leq&\Big[(1-s\lambda_l\epsilon)W_2(\boldsymbol{\pi}_k,\boldsymbol{\pi}_*)+1.65S(\lambda_u^3\epsilon^3p)^{1/2}+\epsilon\eta\lambda_u\sqrt{p}\Big]^2\\&+\epsilon^2\sigma_1^2\lambda_u^2p+2\epsilon^2\sigma_2^2\lambda_u^2pV+2\epsilon^2\sigma_2^2\lambda_u^2W_2^2(\boldsymbol{\pi}_k,\boldsymbol{\pi}_*)\\\leq&\Big[(1-s\lambda_l\epsilon+\sqrt{2}\sigma_2\epsilon\lambda_u)W_2(\boldsymbol{\pi}_k,\boldsymbol{\pi}_*)+1.65S(\lambda_u^3\epsilon^3p)^{1/2}+\epsilon\eta\lambda_u\sqrt{p}\Big]^2\\&+\epsilon^2\sigma_1^2\lambda_u^2p+2\epsilon^2\sigma_2^2\lambda_u^2pV,\end{aligned}$$

which concludes the proof. □

Remark 3.7. *Condition (3.30) in Lemma 3.6 can be further relaxed. For example if we assume bias of gradient estimation is not uniformly bound, increasing with $\|x_k\|$ in the form $\|E(\zeta_k|x_k)\|^2 \leq \eta^2(\sqrt{p}+\|x_k\|)^2$, then by the same technique we can prove that*

$$W_2^2(\boldsymbol{\pi}_{k+1},\boldsymbol{\pi}_*) \leq \Big[(1-s\lambda_l\epsilon+\eta\lambda_u\epsilon+\sqrt{2}\sigma_2\epsilon\lambda_u)W_2(\boldsymbol{\pi}_k,\boldsymbol{\pi}_*)+1.65S(\lambda_u^3\epsilon^3 p)^{1/2}+\epsilon\eta\lambda_u(\sqrt{p}+\sqrt{V})\Big]^2 + \epsilon^2\sigma_1^2\lambda_u^2 p+2\epsilon^2\sigma_2^2\lambda_u^2 pV.$$

In the proof of Lemma 3.6, p is treated as a constant, so it is trivial to extend the proof to the case that $\|E(\zeta_k|x_k)\|^2$ and $E[\|\zeta_k-E(\zeta_k|x_k)\|^2]$ increase in a polynomial of p. In that case, the statements in Remark 3.7, Corollary 1, and Theorem 3.3.2 can be updated accordingly.

Remark 3.8. *Consider a Langevin Monte Carlo algorithm with inaccurate gradients, varying conditioning matrices and a constant learning rate ϵ, i.e.,*

$$x_{k+1}=x_k-\epsilon\Sigma_k[\nabla f(x_k)+\zeta_k]+\sqrt{2\epsilon}\xi_{k+1};\quad \xi_{k+1}\sim N(0,\Sigma_k).$$

If Σ_k is positive definite, $\lambda_l \leq \inf_k \lambda_{\min}(\Sigma_k) \leq \sup_k \lambda_{\max}(\Sigma_k) \leq \lambda_u$ and $\epsilon \leq 2/(s\lambda_l+S\lambda_u)$, then (3.31) holds for all iterations. Combining it with Lemma 1 of Dalalyan et al.*[63], it is easy to obtain that*

$$\begin{aligned} W_2(\boldsymbol{\pi}_k,\boldsymbol{\pi}_*) \leq & (1-s\lambda_l\epsilon+\sqrt{2}\sigma_2\epsilon\lambda_u)^k W_2(\boldsymbol{\pi}_0,\boldsymbol{\pi}_*) \\ & + \frac{\eta\lambda_u\sqrt{p}}{s\lambda_l-\sqrt{2}\sigma_2\lambda_u}+\frac{1.65S(\lambda_u^3\epsilon p)^{1/2}}{s\lambda_l-\sqrt{2}\sigma_2\lambda_u}+\frac{\sqrt{\epsilon}\lambda_u(\sigma_1^2 p+2\sigma_2^2 V)}{1.65S(\lambda_u p)^{1/2}}, \end{aligned} \tag{3.34}$$

where the first term in the RHS of (3.34) converges to 0 if $s\lambda_l > \sqrt{2}\sigma_2\lambda_u$.

The next corollary provides a decaying-learning-rate version of the convergence result (3.34).

Corollary 1. *Consider a Langevin Monte Carlo algorithm*

$$x_{k+1}=x_k-\epsilon_{k+1}\Sigma_k[\nabla f(x_k)+\zeta_k]+\sqrt{2\epsilon_{k+1}}\xi_k;\quad \xi_k\sim N(0,\Sigma_k),$$

where ζ_k satisfies (3.30), Σ_k is positive definite, $\lambda_l \le \inf_k \lambda_{\min}(\Sigma_k) \le \sup_k \lambda_{\max}(\Sigma_k) \le \lambda_u$ and the learning rate $\epsilon_k = 2/[(s\lambda_l + S\lambda_u)k^{\varpi}]$ for some $\varpi \in (0,1)$. If $s\lambda_l > \sqrt{2}\sigma_2\lambda_u$, then we have

$$\limsup_{k\to\infty} W_2(\boldsymbol{\pi}_k, \boldsymbol{\pi}_*) \le \frac{\varphi}{1-\varphi}\frac{\eta\lambda_u\sqrt{p}}{s\lambda_l - \sqrt{2}\sigma_2\lambda_u}, \text{ for some constant } \varphi \in (0,1). \tag{3.35}$$

Proof. The proof of this corollary closely follows the proof of Theorem 2(i) in Song, Sun, Ye, *et al.* [58]. Let $K_0 = 0$, and K_{i} ($\mathrm{i} > 0$) be the smallest integer such that $K_{\mathrm{i}}^{-\varpi} \le (1+\mathrm{i})^{-\chi}$, where $\chi = \varpi/(1-\varpi)$. Thus, asymptotically, we have $K_{\mathrm{i}+1} - K_{\mathrm{i}} \approx (\chi/\varpi)K_{\mathrm{i}+1}^{\varpi}$.

In the spirit of (3.34), we have

$$\begin{aligned} W_2(\boldsymbol{\pi}_{K_{\mathrm{i}+1}}, \boldsymbol{\pi}_*) \le & (1 - (s\lambda_l - \sqrt{2}\sigma_2\lambda_u)\epsilon_{K_{\mathrm{i}+1}})^{K_{\mathrm{i}+1}-K_{\mathrm{i}}} W_2(\boldsymbol{\pi}_{K_{\mathrm{i}}}, \boldsymbol{\pi}_*) \\ & + \frac{\eta\lambda_u\sqrt{p}}{s\lambda_l - \sqrt{2}\sigma_2\lambda_u} + \left[\frac{1.65S(\lambda_u^3 p)^{1/2}}{s\lambda_l - \sqrt{2}\sigma_2\lambda_u} + \frac{\lambda_u(\sigma_1^2 p + 2\sigma_2^2 V)}{1.65S(\lambda_u p)^{1/2}}\right]\sqrt{\epsilon_{K_{\mathrm{i}}}}. \end{aligned}$$

Note that due to the fact that $K_{\mathrm{i}+1} - K_{\mathrm{i}} \approx (\chi/\varpi)\epsilon_{K_{\mathrm{i}+1}}^{-1}$, we have

$$\lim_{\mathrm{i}\to\infty} [1 - (s\lambda_l - \sqrt{2}\sigma_2\lambda_u)\epsilon_{K_{\mathrm{i}+1}}]^{K_{\mathrm{i}+1}-K_{\mathrm{i}}} = \exp\left\{-\frac{2(s\lambda_l - \sqrt{2}\sigma_2\lambda_u)\chi}{(s\lambda_l + S\lambda_u)\varpi}\right\} < 1.$$

Therefore, there exists some positive constant $\varphi \in (\exp(-\frac{2(s\lambda_l - \sqrt{2}\sigma_2\lambda_u)\chi}{(s\lambda_l + S\lambda_u)\varpi}), 1)$, such that for all $\mathrm{i} \ge 1$, $(1 - (s\lambda_l - \sqrt{2}\sigma_2\lambda_u)\epsilon_{K_{\mathrm{i}+1}})^{K_{\mathrm{i}+1}-K_{\mathrm{i}}} \le \varphi$, i.e.,

$$W_2(\boldsymbol{\pi}_{K_{\mathrm{i}+1}}, \boldsymbol{\pi}_*) \le \varphi W_2(\boldsymbol{\pi}_{K_{\mathrm{i}}}, \boldsymbol{\pi}_*) + \frac{\eta\lambda_u\sqrt{p}}{s\lambda_l - \sqrt{2}\sigma_2\lambda_u} + \left[\frac{1.65S(\lambda_u^3 p)^{1/2}}{s\lambda_l - \sqrt{2}\sigma_2\lambda_u} + \frac{\lambda_u(\sigma_1^2 p + 2\sigma_2^2 V)}{1.65S(\lambda_u p)^{1/2}}\right]\sqrt{\epsilon_{K_{\mathrm{i}}}}.$$

The above recursive inequality implies that

$$\begin{aligned} W_2(\boldsymbol{\pi}_{K_T}, \boldsymbol{\pi}_*) \le & \varphi^T W_2(\boldsymbol{\pi}_{K_0} = \boldsymbol{\pi}_0, \boldsymbol{\pi}_*) + (\sum_{t=1}^{T} \varphi^{t-1}) \frac{\eta\lambda_u\sqrt{p}}{s\lambda_l - \sqrt{2}\sigma_2\lambda_u} \\ & + (\sum_{t=1}^{T} \varphi^{t-1} K_{T-t}^{-\varpi/2}) \sqrt{\frac{2}{s\lambda_l + S\lambda_u}} \left[\frac{1.65S(\lambda_u^3 p)^{1/2}}{s\lambda_l - \sqrt{2}\sigma_2\lambda_u} + \frac{\lambda_u(\sigma_1^2 p + 2\sigma_2^2 V)}{1.65S(\lambda_u p)^{1/2}}\right]. \end{aligned} \tag{3.36}$$

As $T \to \infty$, $\sum_{t=1}^{T} \varphi^{t-1} K_{T-t}^{-\varpi/2} \to 0$, hence we have that $W_2(\boldsymbol{\pi}_{K_{T+1}}, \boldsymbol{\pi}_*) \to \frac{\varphi}{1-\varphi} \frac{\eta \lambda_u \sqrt{p}}{s\lambda_l - \sqrt{2}\sigma_2 \lambda_u}$.

□

Remark 3.9. *For technical simplicity, we require $\varpi < 1$ for the decay of the learning rate ($\epsilon_t \propto t^{-\varpi}$) in the above corollary. We conjecture that the corollary still holds under the choice $\epsilon_t \propto t^{-1}$, i.e., $\varpi = 1$. However, more subtle technical tools are necessary to rigorously characterize the convergence rate under $\epsilon_t \propto t^{-1}$, see Teh, Thiery, and Vollmer [60].*

Theorem 3.3.2. *Given Assumption 3.1-3.6 hold, the ensemble size is sufficiently large, $c_1 N_t \leq s_t \leq S_t \leq c_2 N_t$, $c_3(n_t/N_t) \leq \lambda_{t,l} \leq \lambda_{t,u} \leq c_4(n_t/N_t)$, and $\sigma_{t,s}^2 \leq c_5(N_t/n_t)$. Let $\sup_t 1/(N_t n_t) \leq c_6$ for some sufficiently small constant c_6 such that*

$$\varphi_0 = 2\frac{c_1 c_3 - 2\sqrt{c_5 c_6} c_4}{c_1 c_3 + c_2 c_4} \in (0, 1).$$

Denote $V_t = \int \|x\|^2 d\boldsymbol{\pi}_t$ and it is assumed to satisfy the constraint $V_t \leq c_7 p$ for some constant $c_7 > 0$. We choose the learning rate $\epsilon_{t,k} = 2/[(s_t\lambda_{t,l} + S_t\lambda_{t,u})k^{\varpi}]$ ($k = 1, \ldots, \mathcal{K}$, $t = 1, \ldots, \infty$) for some $\varpi \in (0, 1)$. If all N_t's are bounded away from 0, then $\limsup_{t\to\infty} W_2(\tilde{\boldsymbol{\pi}}_{t+1}, \boldsymbol{\pi}_{t+1}) = O(1/\liminf_t N_t)$, as long as $\mathcal{K}$ is sufficiently large (refer to (3.44) for formal conditions).

Proof. Define K_{i} as in the proof of Corollary 1, and we let $\mathcal{K} = K_{\varkappa} (\asymp \varkappa^{\chi/\varpi})$ for some $\varkappa$ to be specified later, where $\chi = \varpi/(1-\varpi)$.

At stage $t = 1$, Algorithm 6 performs exactly as Algorithm 4; that is, it is a Langevin Monte Carlo algorithm with a varying conditioning matrix. By Corollary 1 with no bias $\eta = 0$, we obtain that there exists some $\varphi \in (\exp\{-\varphi_0 \chi/\varpi\}, 1)$, such that

$$\begin{aligned} W_2(\tilde{\boldsymbol{\pi}}_1, \boldsymbol{\pi}_1) \leq & \varphi^{\varkappa} W_2(\nu_1, \boldsymbol{\pi}_1) + (\sum_{\mathrm{j}=1}^{\varkappa} \varphi^{\mathrm{j}-1} K_{\varkappa-\mathrm{j}}^{-\frac{\varpi}{2}}) \sqrt{\frac{2}{s_1\lambda_{1,l} + S_1\lambda_{1,u}}} \\ & \times \left[\frac{1.65 S_1 \lambda_{1,u} \sqrt{p\lambda_{1,u}}}{s_1\lambda_{1,l} - \sqrt{2}\sigma_{1,s}\lambda_{1,u}} + \frac{\sigma_{1,s}^2 \lambda_{1,u}(p + 2V_1)}{1.65 S_1 \sqrt{\lambda_{1,u} p}} \right] \\ \leq & \varphi^{\varkappa} M + C_0 (\sum_{\mathrm{j}=1}^{\varkappa} \varphi^{\mathrm{j}-1} K_{\varkappa-\mathrm{j}}^{-\frac{\varpi}{2}}) \sqrt{\frac{p}{N_1}}, \end{aligned} \tag{3.37}$$

for some constant C_0 (which only depends on the c_{i}'s values in the statement of this theorem), where ν_1 denotes the prior distribution of x_1.

Now, we study the relationship between $W_2(\tilde{\pi}_{t+1}, \pi_{t+1})$ and $W_2(\tilde{\pi}_t, \pi_t)$ for $t \geq 2$. At stage $t+1$, the algorithm can be rewritten as

$$\begin{aligned} x^{a,\mathrm{i}}_{t+1,k+1} &= x^{a,\mathrm{i}}_{t+1,k} + \epsilon_{t+1,k+1}\Sigma_{t+1,k+1}\left[\frac{N}{n}H^T_{t+1,k}V^{-1}_{t+1}(y_{t+1,k} - H_{t+1,k}x^{a,\mathrm{i}}_{t+1,k}) + \nabla \log \pi(x^{a,\mathrm{i}}_{t+1,k}|\tilde{x}^{\mathrm{i}}_{t,k})\right] + \mathrm{e}_{t+1} \\ &\triangleq x^{a,\mathrm{i}}_{t+1,k} + \epsilon_{t+1,k+1}\Sigma_{t+1,k+1}\left[(I) + (II)\right] + \mathrm{e}_{t+1}, \end{aligned} \tag{3.38}$$

where $\mathrm{e}_{t+1} \sim N(0, 2\epsilon_{t+1,k+1}\Sigma_{t+1,k+1})$, and $\tilde{x}^{\mathrm{i}}_{t,k}$ denotes a sample drawn from the set $\mathcal{X}_t$ according to an importance weight proportional to $\pi(x^{a,\mathrm{i}}_{t+1,k}|\tilde{x}^{\mathrm{i}}_{t,k})$.

To apply (3.36), it is necessary to study the bias and variance of the gradient estimate used in (3.38). Note that the term (I) is unbiased due to the property of simple random sampling. To study the bias of term (II), we define $\pi(z|x^{a,\mathrm{i}}_{t+1,k}, y_{1:t}) \propto \pi(x^{a,\mathrm{i}}_{t+1,k}|z)\pi_t(z|y_{1:t})$; that is, $\pi(z|x^{a,\mathrm{i}}_{t+1}, y_{1:t})$ can be viewed as a posterior density obtained with the prior density $\pi_t(z|y_{1:t})$ and the likelihood $\pi(x^{a,\mathrm{i}}_{t+1}|z)$. Similarly, we define $\tilde{\pi}(z|x^{a,\mathrm{i}}_{t+1,k}, y_{1:t}) \propto \pi(x^{a,\mathrm{i}}_{t+1,k}|z)\tilde{\pi}_t(z|y_{1:t})$. Then, by equation (3.15) of the main text, the bias of term (II) can be bounded by

$$\begin{aligned} &\left\| \int \nabla \log \pi(x^{a,\mathrm{i}}_{t+1,k}|z)[d\tilde{\pi}(z|x^{a,\mathrm{i}}_{t+1,k}, y_{1:t}) - d\pi(z|x^{a,\mathrm{i}}_{t+1,k}, y_{1:t})] \right\| \\ =&\left\| \int U_t^{-1}[x^{a,\mathrm{i}}_{t+1,k} - g(z)][d\tilde{\pi}(z|x^{a,\mathrm{i}}_{t+1,k}, y_{1:t}) - d\pi(z|x^{a,\mathrm{i}}_{t+1,k}, y_{1:t})] \right\| \\ =&\left\| -\int U_t^{-1} g(z)[d\tilde{\pi}(z|x^{a,\mathrm{i}}_{t+1,k}, y_{1:t}) - d\pi(z|x^{a,\mathrm{i}}_{t+1,k}, y_{1:t})] \right\| \leq 2M_g/\lambda_{t,s}, \end{aligned} \tag{3.39}$$

which holds for any $\tilde{\pi}(\cdot|\cdot)$.

By Assumption 3.4, the variance of term (I) is bounded by $\sigma^2_{t+1,s}(p + \|x\|^2)$. The variance of term (II) is upper bounded by

$$\begin{aligned} &E\left\| \nabla \log \pi(x^{a,\mathrm{i}}_{t+1,k}|\tilde{x}^{\mathrm{i}}_{t,k}) - E\left(\nabla \log \pi(x^{a,\mathrm{i}}_{t+1,k}|\tilde{x}^{\mathrm{i}}_{t,k})\right) \right\|^2 \leq (l/\lambda_{t,s})^2 E\|\tilde{x}^{\mathrm{i}}_{t,k} - E(\tilde{x}^{\mathrm{i}}_{t,k})\|^2 \quad \text{(by Lemma 3.3)} \\ \leq &(l/\lambda_{t,s})^2 \left[W_2(\pi_t, \tilde{\pi}_t)^2 + p\sigma^2_{t,v} + 2W_2(\pi_t, \tilde{\pi}_t)\sqrt{p\sigma^2_{t,v}}\right] \quad \text{(by Lemma 3.4 and (3.29))}. \end{aligned} \tag{3.40}$$

Combining the above results together, the variance of the estimated gradient is upper bounded by

$$2\sigma_{t+1,s}^2 p + 2(l/\lambda_{t,s})^2 (W_2(\boldsymbol{\pi}_t, \tilde{\boldsymbol{\pi}}_t) + \sqrt{p\sigma_{t,v}^2})^2 + 2\sigma_{t+1,s}^2 \|x_{t+1,k}^{a,\mathrm{i}}\|^2 := \sigma_p^2 + 2\sigma_{t+1,s}^2 \|x_{t+1,k}^{a,\mathrm{i}}\|^2, \quad (3.41)$$

which implies that a smaller value of $W_2(\boldsymbol{\pi}_t, \tilde{\boldsymbol{\pi}}_t)$ will help to reduce the variance of the stochastic gradient at iteration $t+1$ as well as $W_2(\tilde{\boldsymbol{\pi}}_{t+1}, \boldsymbol{\pi}_{t+1})$ as implied by (3.42).

Recall that $\tilde{\nu}_{t+1}$ denotes the practical state initial distribution of $x_{t+1,0}^{a,\mathrm{i}}$. Applying equation (3.36) with $\sigma_1^2 = \sigma_p^2/p$, $\sigma_2^2 = 2\sigma_{t+1,s}^2$ and $\eta = 2M_g/(\sqrt{p}\lambda_{t,s})$, we obtain that there exists some $\varphi \in (\exp\{-\varphi_0\chi/\varpi\}, 1)$ such that

$$\begin{aligned} W_2(\tilde{\boldsymbol{\pi}}_{t+1}, \boldsymbol{\pi}_{t+1}) \leq & \varphi^{\varkappa} W_2(\tilde{\nu}_{t+1}, \boldsymbol{\pi}_{t+1}) + \frac{\varphi}{1-\varphi} \frac{2M_g \lambda_{t+1,u}}{\lambda_{t,s}(s_{t+1}\lambda_{t+1,l} - 2\sigma_{t+1,s}\lambda_{t+1,u})} \\ & + (\sum_{\mathrm{j}=1}^{\varkappa} \varphi^{\mathrm{j}-1} K_{\varkappa-\mathrm{j}}^{-\frac{\varpi}{2}}) \sqrt{\frac{2}{s_{t+1}\lambda_{t+1,l} + S_{t+1}\lambda_{t+1,u}}} \\ & \times \left[\frac{1.65 S_{t+1}\sqrt{p\lambda_{t+1,u}^3}}{s_{t+1}\lambda_{t+1,l} - 2\sigma_{t+1,s}\lambda_{t+1,u}} + \frac{\sqrt{\lambda_{t+1,u}}(\sigma_p^2 + 4\sigma_{t+1,s}^2 V_t)}{1.65 S_{t+1}\sqrt{p}} \right]. \end{aligned} \quad (3.42)$$

Note that φ exists because $\varphi_0 \leq \min_t 2\frac{s_{t+1}\lambda_{t+1,l} - 2\sigma_{t+1,s}\lambda_{t+1,u}}{s_{t+1}\lambda_{t+1,l} + S_{t+1}\lambda_{t+1,u}}$, and w.o.l.g., we let the φ's in (3.42) and (3.37) match. By Assumption 3.6 and Lemma 3.2, $W_2(\tilde{\nu}_{t+1}, \boldsymbol{\pi}_{t+1}) \leq M + lW_2(\tilde{\boldsymbol{\pi}}_t, \boldsymbol{\pi}_t)$. Further, combining with other conditions stated in the theorem, we have

$$\begin{aligned} W_2(\tilde{\boldsymbol{\pi}}_{t+1}, \boldsymbol{\pi}_{t+1}) \leq & \varphi^{\varkappa}(M + lW_2(\tilde{\boldsymbol{\pi}}_t, \boldsymbol{\pi}_t)) + C_1 \frac{\varphi}{1-\varphi} \frac{2M_g}{N_{t+1}} \\ & + C_2 (\sum_{\mathrm{j}=1}^{\varkappa} \varphi^{\mathrm{j}-1} K_{\varkappa-\mathrm{j}}^{-\frac{\varpi}{2}}) \sqrt{\frac{p}{N_{t+1}}} + C_3 (\sum_{\mathrm{j}=1}^{\varkappa} \varphi^{\mathrm{j}-1} K_{\varkappa-\mathrm{j}}^{-\frac{\varpi}{2}}) \sqrt{\frac{1}{N_{t+1}^3 p}} W_2^2(\boldsymbol{\pi}_t, \tilde{\boldsymbol{\pi}}_t), \end{aligned} \quad (3.43)$$

for some positive constants C_1, C_2 and C_3 (which only depend on the c_{i}'s values in the statement of this theorem), where the last term follows from the definition of σ_p^2 given in (3.41).

Based on the recursive inequalities (3.37) and (3.43), one can conclude that $W_2(\tilde{\boldsymbol{\pi}}_{t+1}, \boldsymbol{\pi}_{t+1})$ can converge to any arbitrarily small quantity, as long as that $\varkappa$ and N_t are sufficiently large.

Let $\tilde{\varepsilon}$ be some small positive quantity $\tilde{\varepsilon} \in (0,1)$, and we assume that sample size N_t is non-decreasing with respect to t, N_t and the iteration number $\mathcal{K} = K_{\varkappa}(\asymp \varkappa^{\chi/\varpi})$ satisfy

$$\begin{aligned} &\lim N_t = N_\infty; \\ &\varphi^{\varkappa} < \min\{\tilde{\varepsilon}/(3M), 1/(3l)\} \text{ and } \sum_{\mathrm{j}=1}^{\varkappa} \varphi^{\mathrm{j}-1} K_{\varkappa-\mathrm{j}}^{-\frac{\varpi}{2}} \leq \min\left\{\frac{\sqrt{N_1^3 p}}{3C_3\overline{W}}, 1, \frac{\tilde{\varepsilon}}{C_2\sqrt{p/N_1}}\right\}, \end{aligned} \tag{3.44}$$

Let's define $\overline{W} = \max\{1 + 2C_1\frac{\varphi}{1-\varphi}\frac{2M_g}{N_1} + 2C_2\sqrt{\frac{p}{N_1}}, 1 + C_0\sqrt{p}\}$, then it is not difficult to verify that (i) $W_2(\tilde{\boldsymbol{\pi}}_t, \boldsymbol{\pi}_t) \leq \overline{W}$ for all $t \geq 1$ and (ii) $W_2(\tilde{\boldsymbol{\pi}}_{t+1}, \boldsymbol{\pi}_{t+1}) \leq \tilde{\varepsilon}/3 + W_2(\tilde{\boldsymbol{\pi}}_t, \boldsymbol{\pi}_t)/3 + W_2^2(\tilde{\boldsymbol{\pi}}_t, \boldsymbol{\pi}_t)/(3\overline{W}) + C_1\frac{\varphi}{1-\varphi}\frac{2M_g}{N_{t+1}} + \tilde{\varepsilon}$. Combine them with the fact that $\lim_T \sum_{t=1}^{T}(2/3)^{T-t}N_t^{-1} = 2\lim_t(1/N_t)$, this further implies

$$\limsup_{t\to\infty} W_2(\tilde{\boldsymbol{\pi}}_{t+1}, \boldsymbol{\pi}_{t+1}) \leq 4\tilde{\varepsilon} + 2C_1\frac{\varphi}{1-\varphi}M_g N_\infty^{-1}.$$

Equivalently, we claim that given a sufficiently large number of iterations in each stage, then

$$\limsup_{t\to\infty} W_2(\tilde{\boldsymbol{\pi}}_{t+1}, \boldsymbol{\pi}_{t+1}) = O\left(\frac{1}{\liminf_t N_t}\right).$$

□

4. EMPIRICAL RESULTS OF THE LENKF ALGORITHM

In this chapter, we present multiple examples, including Bayesian variable selection for linear and nonlinear systems, the Lorenz-96 model, and Bayesian learning for deep neural networks and LSTMs. The numerical results indicate that it is not only able to produce an accurate point estimate as the existing methods do, but also able to quantify uncertainty of the resulting estimate.

4.1 Numerical studies for static learning problems

This section illustrates the performance of the LEnKF as a general Monte Carlo algorithm for complex inverse problems. Two big data examples are considered, one is Bayesian variable selection for linear regression, and the other is Bayesian variable selection for general nonlinear systems modeled by deep neural networks (DNNs).

4.1.1 Bayesian variable selection for large-scale linear regression

Consider the linear regression

$$\boldsymbol{Y} = \boldsymbol{Z}\boldsymbol{\beta} + \varepsilon, \tag{4.1}$$

where $\boldsymbol{Y} \in \mathbb{R}^N$, $\boldsymbol{Z} = (Z_1, Z_2, \ldots, Z_p) \in \mathbb{R}^{N\times p}$, $\boldsymbol{\beta} \in \mathbb{R}^p$, and $\varepsilon \sim N(0, I_N)$. An intercept term has been implicitly included in the model. We generate ten datasets from this model with $N = 50,000$, $p = 2,000$, and $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_p) = (1, 1, 1, 1, 1, -1, -1, -1, 0, \ldots, 0)$. That is, the first 8 variables are true and the others are false. Each variable in Z has a marginal distribution of $N(0, I_N)$, but they are mutually correlated with a correlation coefficient of 0.5.

To conduct Bayesian analysis, we consider the following hierarchical mixture Gaussian prior distribution which, with the latent variable $\xi_{\mathrm{i}} \in \{0, 1\}$, can be expressed as

$$\begin{aligned} &\beta_{\mathrm{i}}|\xi_{\mathrm{i}} \sim (1-\xi_{\mathrm{i}})N(0, \tau_1^2) + \xi_{\mathrm{i}}N(0, \tau_2^2), \\ &P(\xi_{\mathrm{i}} = 1) = 1 - P(\xi_{\mathrm{i}} = 0) = p_0, \end{aligned} \tag{4.2}$$

for $\mathrm{i} = 1, 2, \ldots, p$. Such a prior distribution has been widely used in the literature of Bayesian variable selection (see, e.g., [66]). To apply the LEnKF to this problem, we first integrate out ξ_i from the prior distribution (4.2), which leads to the marginal distribution

$$\beta_\mathrm{i} \sim (1 - p_0)N(0, \tau_1^2) + p_0 N(0, \tau_2^2), \quad \mathrm{i} = 1, 2, \ldots, p, \tag{4.3}$$

for which the log-prior density is differentiable. Algorithm 4 can then be applied to simulate from the posterior distribution $\boldsymbol{\pi}(\boldsymbol{\beta}|\boldsymbol{Y}, \boldsymbol{Z})$. In the simulation, we set $p_0 = 1/p = 0.0005$, $\tau_1^2 = 0.01$ and $\tau_2^2 = 1$ for the prior distribution, and set the ensemble size $m = 100$, the mini-batch size $n = 100$, and the learning rate $\epsilon_t = 0.2/\max\{t_0, t\}^{0.6}$, where $t_0 = 100$. The algorithm was run for 10,000 iterations, which cost 375 CPU seconds on a personal computer with 2.9GHz Intel Core i7 CPU and 16GB RAM. All computation reported in this dissertation were done on the same computer.

To accomplish the goal of variable selection, we consider the factorization of the posterior distribution $\boldsymbol{\pi}(\boldsymbol{\beta}, \boldsymbol{\xi}|\boldsymbol{Y}) \propto \boldsymbol{\pi}(\boldsymbol{Y}|\boldsymbol{\beta})\boldsymbol{\pi}(\boldsymbol{\beta}|\boldsymbol{\xi})\boldsymbol{\pi}(\boldsymbol{\xi})$, where $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_p)$. Under *a priori* independent assumptions for β_i's and ξ_i's, we are able to draw posterior samples of $\boldsymbol{\xi}$ from the following distribution:

$$\boldsymbol{\pi}(\xi_{t\mathrm{i}} = 1|\beta_{t\mathrm{i}}, \boldsymbol{Y}) = \frac{a_{t\mathrm{i}}}{a_{t\mathrm{i}} + b_{t\mathrm{i}}}, \quad \mathrm{i} = 1, 2, \ldots, p, \tag{4.4}$$

where $a_{t\mathrm{i}} = \frac{p_0}{\tau_2}\exp(-\beta_{t\mathrm{i}}^2/2\tau_2^2)$, $b_{t\mathrm{i}} = \frac{1-p_0}{\tau_1}\exp(-\beta_{t\mathrm{i}}^2/2\tau_1^2)$, and $\beta_{t\mathrm{i}}$ denotes the posterior sample of β_i drawn by Algorithm 4 at stage t. Here we denote by $\boldsymbol{\beta}_t = (\beta_{t,1}, \beta_{t,2}, \ldots, \beta_{t,p})$ a posterior sample of $\boldsymbol{\beta}$ drawn by Algorithm 4 at the analysis step of stage t.

Figure 4.1 summarizes the variable selection results for one data set. The results for other data sets are similar. Figure 4.1(a) shows the sample trajectories of $\beta_1, \beta_2, \ldots, \beta_9$, which are averaged over the ensemble along with iterations. All the samples converge to their true values within 100 iterations, taking about 3.7 CPU seconds. This is extremely fast! Figure 4.1(b) shows the marginal inclusion probabilities of the variables $Z_1, Z_2, \ldots, Z_p$. From this graph, we can see that each of the 8 true variables (indexed 1-8) has a marginal inclusion probability close to 1, while each false variable has a marginal inclusion probability close

to 0. Figure 4.1(c) shows the scatter plot of the response variable and its fitted value for the training data, and Figure 4.1(d) shows the scatter plot of the response variable and its predicted value for 200 test samples generated from the model (4.1). In summary, Figure 4.1 shows that the LEnKF is able to identify true variables for large-scale linear regression and, moreover, it is extremely efficient.

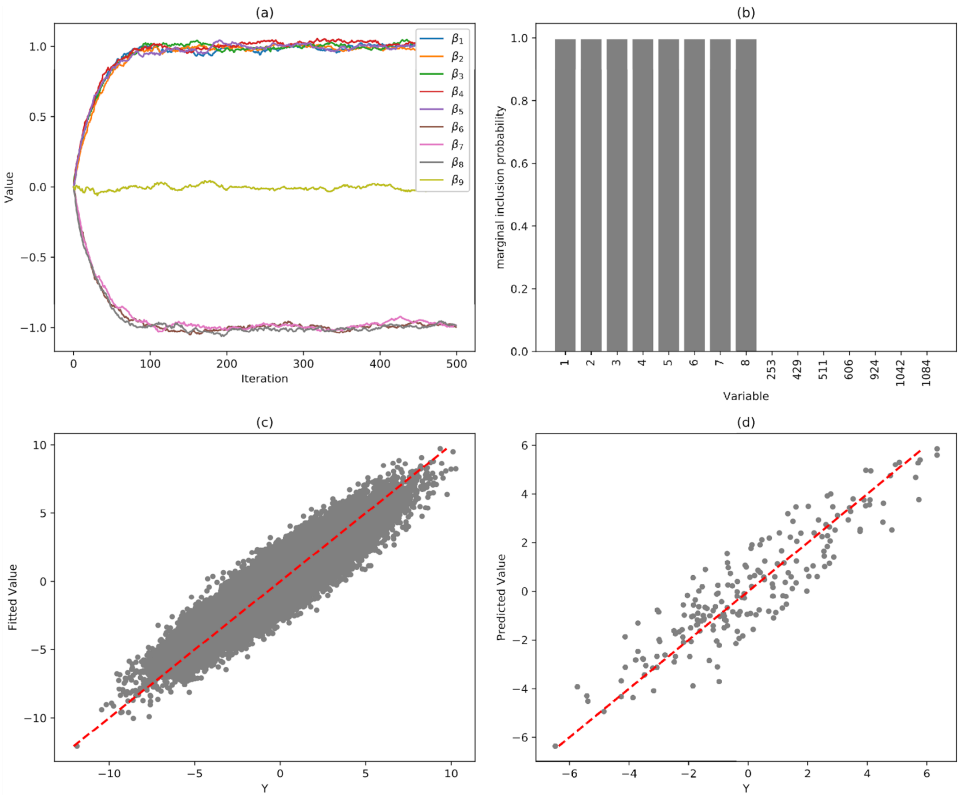


Figure 4.1. LEnKF for large-scale linear regression with 500 iterations: (a) Trajectories of $\beta_1, \ldots, \beta_9$, where $\beta_1, \ldots, \beta_5$ have a true value of 1, $\beta_6, \ldots, \beta_8$ have a true value of -1, and β_9 has a true value of 0. (b) marginal inclusion probabilities of all covariates $X_1, \ldots, X_p$, where the covariates are shown in the rank of marginal inclusion probabilities; (c) scatter plot of the response Y and the fitted value for training samples; and (d) scatter plot of the response Y and the predicted value for test samples.

For comparison, SGLD [1], preconditioned SGLD [6], and stochastic gradient Nosé-Hoover thermostat [2] were applied to this example. For these algorithms, the learning rates have been tuned to their maximum values such that the simulation converges fast while not exploding. For SGLD, we set $\epsilon_t = 4 \times 10^{-6} / \max\{t_0, t\}^{0.6}$ with $t_0 = 1000$; for pSGLD, we set

$\epsilon_t = 5 \times 10^{-6}/\max\{t_0, t\}^{0.6}$ with $t_0 = 1000$; and for SGNHT, we set $\epsilon = 0.0001$. Other than the learning rate, pSGLD contains two more tuning parameters, which control the extremes of the curvatures and the balance of the weights of the historical and current gradients, respectively. They both were set to the default values as suggested by Li, Chen, Carlson, *et al.* [6]. SGNHT also contains an extra parameter, the so-called diffusion parameter, for which different values, including 1, 5, 10, and 20, have been tried. The algorithm performed very similarly with each of the choices. Figures 4.2 summarizes the results of the algorithm with the diffusion parameter being set to 10.

For a fair comparison, we ran SGLD for 20,000 iterations, pSGLD for 10,000 iterations, and SGNHT for 15,000 iterations, which took about 387 CPU seconds, 410 CPU seconds and 380 CPU seconds, respectively. All these algorithms cost slightly longer CPU time than the LEnKF. Figure 4.2 compares the trajectories of $(\beta_1, \beta_2, \ldots, \beta_9)$ produced by the three algorithms. It indicates the LEnKF can converge significantly faster than SGLD, pSGLD, SGNHT for this example, which can be explained as the advantage of using the second-order gradient information in the LEnKF.

4.1.2 Bayesian nonlinear variable selection with deep neural networks

The goal of this example is to show that the LEnKF can be used to train Bayesian DNNs and as one of its important application, the Bayesian DNN can be used in variable selection for nonlinear systems. We generated 10 data sets from the nonlinear regression model

$$y = \frac{10x_1^2}{1 + x_2^2} + 5\sin(x_3 x_4) + x_5 + \epsilon, \tag{4.5}$$

where $\epsilon \sim N(0, I_N)$. The variables $x_1, \cdots, x_5$ together with additional 94 variables were generated as in Section 4.1.1: all the variables are mutually correlated with a correlation coefficient of 0.5, and each has a marginal distribution of $N(0, I_N)$. Each data set consists of $N = 2,000,000$ samples for training and 200 samples for testing.

We modeled the data using a 3-hidden-layer neural network, with $p = 100$ input units including a bias unit (for the intercept term), 5 units on each hidden layer and one unit on the

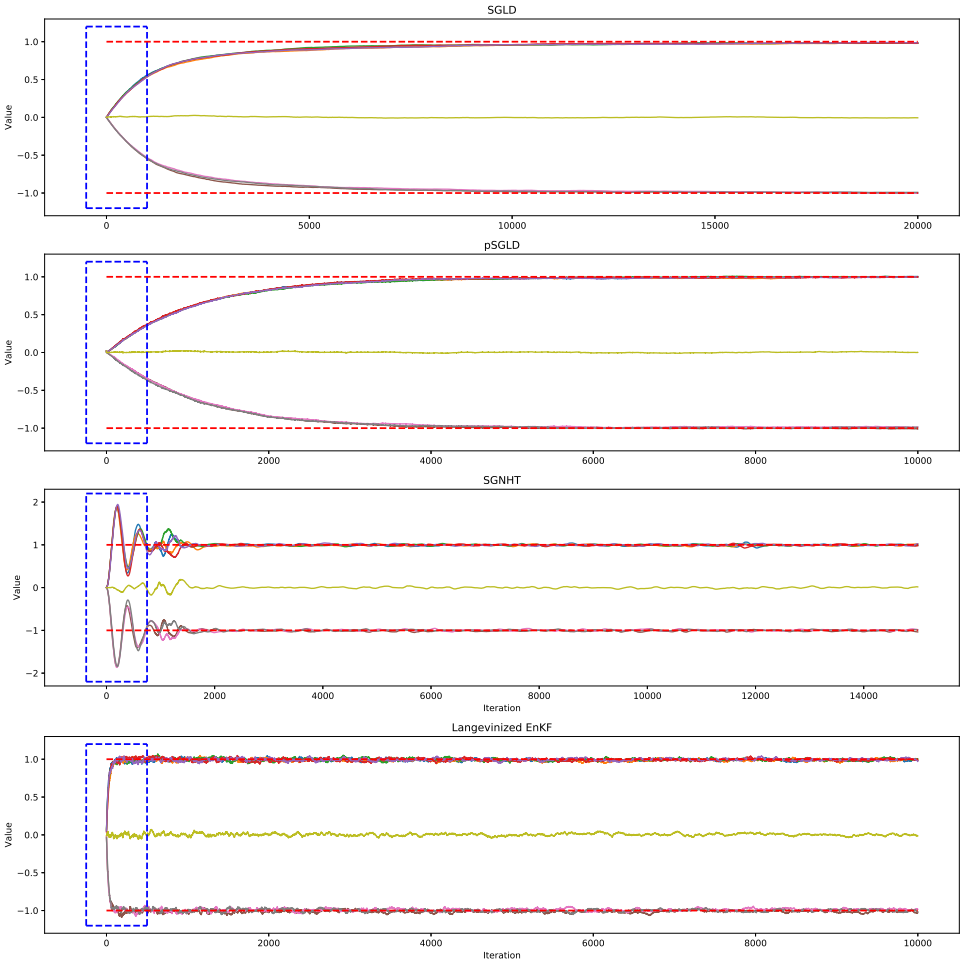


Figure 4.2. Convergence trajectories of SGLD, pSGLD, SGNHT and LEnKF for a large-scale linear regression example: Trajectories of $(\beta_1, \beta_2, \ldots, \beta_9)$ produced by SGLD (upper), pSGLD (upper middle), SGNHT (lower middle), and LEnKF (lower) in their whole runs, where the blue rectangle highlights the first 5% iterations of the runs.

output layer. The activation function is LeakyRelu given by $LeakyRelu(x) = 1_{(x<0)}(0.1x) + 1_{(x>=0)}(x)$. That is, we modeled the data by the neural network function

$$y = LeakyRelu(LeakyRelu(LeakyRelu(X \cdot W_1) \cdot W_2 + b_1) \cdot W_3 + b_2) \cdot W4 + b_3 + \epsilon,$$

where $X = (x_0, x_1, x_2, \ldots, x_{99})$, $W_1 \in \mathbb{R}^{p\times 5}$, $W_2 \in \mathbb{R}^{5\times 5}$, $W_3 \in \mathbb{R}^{5\times 5}$ and $W_4 \in \mathbb{R}^{5\times 1}$ represent the weight matrices at different layers of the neural network, and $b_1 \in \mathbb{R}^{5\times 1}$, $b_2 \in \mathbb{R}^{5\times 1}$, $b_3 \in \mathbb{R}^{5\times 1}$ represent bias vectors at different hidden layers. For each element of

W_i's and b_i's, we assume a hierarchical mixture Gaussian prior as given in (4.2). For the prior hyperparameters, we set $p_0 = 0.01$, $\tau_1^2 = 0.05$, $\tau_2^2 = 1$. Conditioned on each posterior sample of $(W_1, W_2, W_3, W_4, b_1, b_2, b_3)$, a sparse neural network can be drawn by simulating an indicator variable for each potential connection of the neural network according to the Bernoulli distribution given in (4.4).

To identify important input variables, for each sparse neural network drawn above, we define four $\boldsymbol{\xi}$-matrices G_1, G_2, G_3 and G_4, which correspond to W_1, W_2, W_3 and W_4, respectively. Each element of the $\boldsymbol{\xi}$-matrix indicates the status, existing or not, of the corresponding connection. Define $G = G_1 \cdot G_2 \cdot G_3 \cdot G_4$ as a p-vector and further truncates its each element to 1 if greater than 1. That is, each element of G indicates the effectiveness of the corresponding input variable. Averaging G over the sparse network samples produces the marginal inclusion probabilities of the input variables.

Algorithm 5 was first applied to this example with the variance split proportion $\alpha = 0.9$, the ensemble size $m = 20$, the mini-batch size $n = 100$, the iteration number $\mathcal{K} = 5$ for each stage, and the total number of stages $T = 20,000$. The learning rate was set to $\epsilon_{t,k} = 4 \times 10^{-4} / \max\{k_0, k\}^{0.9}$ with $k_0 = 1$ for $k = 1, \cdots, \mathcal{K}$. Each run took about 4000 CPU seconds. For stochastic gradient MCMC algorithms, to avoid simulations to explode and meanwhile to achieve fast convergence, we often need to start with a very small learning rate and then decrease it very slowly or even keep it as a constant. This note also applies to the remaining examples of this dissertation.

Figure 4.3 summarizes the results for one dataset. The results for the other datasets are similar. Figure 4.3(a) shows the marginal inclusion probabilities of all input variables $x_1, x_2, \ldots, x_p$ with a cutoff value of 0.5 (red dash line). The results are very encouraging: Each of the true variables (indexed 1-5) has a marginal inclusion probability close to 1, while each of the false variables has a marginal inclusion probability close to 0. Figure 4.3(b) shows the scatter plot of the response variable and its fitted value for 2,000 randomly chosen training samples. Figure 4.3(c) shows the scatter plot of the response variable and its predicted value for 200 test samples. In summary, Figure 4.3 shows that the LEnKF provides a feasible algorithm for training Bayesian deep neural networks, through which important variables can be identified for nonlinear systems.

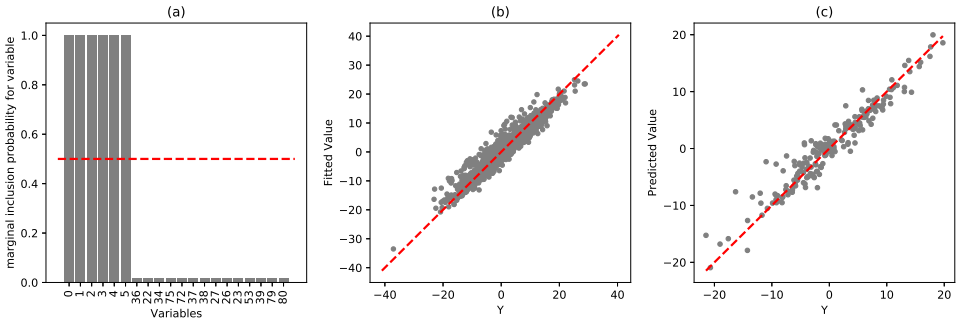


Figure 4.3. LEnKF for the nonlinear variable selection example: (a) marginal inclusion probabilities of the variables, where the variables are shown in the rank of marginal inclusion probabilities; (b) scatter plot of the response Y and the fitted value for 2,000 randomly selected training samples; and (c) scatter plot of the response Y and the predicted value for 200 test samples.

For comparison, SGLD and pSGLD were applied to this example. Each algorithm was run in parallel with 20 chains and each chain consisted of 100,000 iterations. For both algorithms, we set the learning rate as $\epsilon_t = 1 \times 10^{-4}/\max\{t_0, t\}$ with $t_0 = 10,000$, which has been tuned to its maximum value such that the simulations converge very fast but won't explode. Each run of SGLD cost about 4000 CPU seconds, and each run of pSGLD cost about 5600 CPU seconds. For the LEnKF, we measured the fitting and prediction errors, in mean squared errors (MSEs), at the last iteration of each stage. For SGLD and pSGLD, we measured the fitting and prediction errors at every 5th iteration. Figure 4.4 (a) shows the paths of the best fitting and prediction MSEs produced by the time by each chain of pSGLD, SGLD, and LEnKF. Figure 4.4 (b) shows the path of the best fitting and prediction MSEs by the time produced by respective algorithms. Both plots indicates the superiority of the LEnKF over SGLD and pSGLD: the LEnKF tends to produce smaller fitting and prediction errors than SGLD and pSGLD for this example.

Table 4.1 summarizes the results of LEnKF for 10 datasets under different settings of $\mathcal{K}$ and α, along with comparisons with SGLD and pSGLD. For each dataset, we calculated "MeanMSFE" by averaging the fitting MSEs over the stages $t = 15,001, \ldots, 20,000$, and "MeanMSPE" by averaging the prediction MSEs over the stages $t = 15,001, \ldots, 20,000$. Then their values were averaged over 10 datasets and denoted by "Ave-MeanMSFE" and "Ave-MeanMSPE", respectively. The comparisons indicate that for this example, the LEnKF

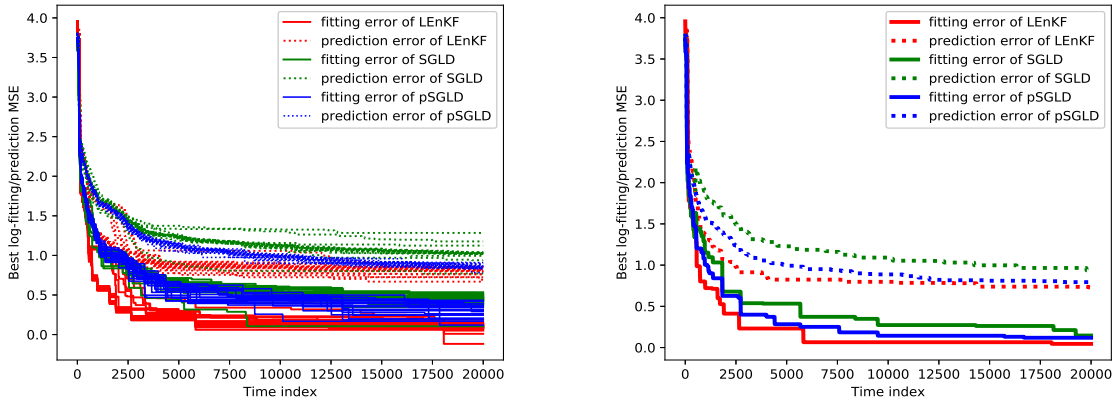


Figure 4.4. Comparison of the best fitting and prediction MSEs by the time (plotted in logarithm): (a) by each chain of SGLD, pSGLD and LEnKF; (b) by ensemble averaging of SGLD, pSGLD and averaging LEnKF.

outperforms SGLD and pSGLD when $\mathcal{K}$ and α are chosen appropriately, and it prefers to run with slightly large values of $\mathcal{K}$ and α. For this example, the LEnKF performed similarly with $\mathcal{K} = 5$ and $\mathcal{K} = 10$, while much better than with $\mathcal{K} = 1$ and 2; and the choice of α affects not much on its training error, but its prediction error: A larger value of α tends to lead to a smaller prediction error. As implied by (3.13), running the LEnKF with a mini-batch size of n is equivalent to running SGLD with a mini-batch size of $n\alpha/(1-\alpha)$. Indeed, as shown in Table 4.1, SGLD with $n = 900$ produced similar training and test errors as LEnKF with $n = 100$, $\alpha = 0.9$ and $\mathcal{K} = 5$. How the batch size affects on the prediction error is an interesting problem in machine learning, which can be partially explained by Theorem 1 of Liang, Cheng, Song, *et al.* [67], which accounts for the effect of batch size on the convergence of SGD, but deserves a further study.

In summary, this example shows that the LEnKF provides a more efficient algorithm than SGLD and pSGLD for training a Bayesian DNN, and that Bayesian DNN can be used in variable selection for nonlinear systems by imposing a mixture Gaussian prior on each weight of the DNN. For simplicity, we consider only the case that the sample size is larger than the DNN size, i.e., total number of connections of the fully connected DNN. As shown in Sun, Song, and Liang [68], such a prior also works for the case that the sample size is

Table 4.1. Comparison of LEnKF, parallel SGLD and parallel pSGLD for the nonlinear regression example, where the numbers in the parentheses denote the standard deviations of the averaged MeanMSFE and MeanMSPE values over 10 datasets.

	n	$\mathcal{K}$	α	Ave-MeanMSFE	Ave-MeanMSPE	CPU(s)
LEnKF	100	5	0.1	2.4749(0.11)	3.4482(0.16)	4050.41
	100	5	0.3	2.4171(0.09)	3.3056(0.13)	4171.32
	100	5	0.5	2.4472(0.10)	3.1841(0.13)	4082.99
	100	5	0.8	2.5852(0.09)	3.1449(0.12)	4040.84
	100	5	0.9	2.4319(0.10)	3.1437(0.14)	4041.91
	100	10	0.9	2.3422(0.07)	3.1469(0.17)	7766.84
	100	2	0.9	2.8509(0.17)	4.4075(0.27)	1943.71
	100	1	0.9	3.6848(0.25)	4.8561(0.22)	1093.09
SGLD	100			2.4747(0.12)	3.3877(0.12)	4111.09
	900			2.4255(0.17)	3.1997(0.13)	5772.94
pSGLD	100			2.6317(0.11)	3.3988(0.10)	5624.10

much smaller than the DNN size. In this case, posterior consistency and variable selection selection still hold.

4.2 Numerical studies for dynamic learning problems

This section illustrates the performance of the LEnKF as a particle filtering algorithm for dynamic problems. Two examples are considered under the Bayesian framework. One is uncertainty quantification for the Lorenz-96 model which has been considered as the benchmark example for weather forecasting. The other is on-line learning with Long short-term memory (LSTM) networks.

4.2.1 Uncertainty quantification for the Lorenz-96 model

The Lorenz-96 model was developed by Edward Lorenz in 1996 to study difficult questions regarding predictability in weather forecasting [33]. The model is given by

$$\frac{dx^{\mathrm{i}}}{dt} = (x^{\mathrm{i}+1} - x^{\mathrm{i}-2})x^{\mathrm{i}-1} - x^{\mathrm{i}} + F, \quad \mathrm{i} = 1, 2, \cdots, p,$$

where $F = 8$, $p = 40$, and it is assumed that $x^{-1} = x^{p-1}$, $x^0 = x^p$, and $x^{p+1} = x^1$. Here F is known as a forcing constant, and $F = 8$ is a common value known to cause chaotic behavior. In order to generate the true state $\boldsymbol{X}_t = (X_t^1, \ldots, X_t^p)$ for $t = 1, 2, \ldots, T$, we initialized $\boldsymbol{X}_0$ by setting X_0^{i} to 20 for all i but adding to X_0^{20} a small perturbation of 0.1; we solved the differential equation using the fourth-order Runge-Kutta numerical method with a time interval of $\Delta t = 0.01$; and for each i and t, we added to X_t^{i} a random noise generated from $N(0, 1)$. At each stage t, data was observed for half of the state variables and masked with a standard Gaussian random noise, i.e.,

$$y_t = H_t \boldsymbol{X}_t + \epsilon_t, \quad t = 1, 2, \ldots, T,$$

where $\epsilon_t \sim N(0, I_{p/2})$, and H_t is a random selection matrix. Figure 4.5 shows the simulated path of the partial state variables (X_t^1, X_t^2, X_t^3) for $t = 1, 2, \ldots, T$, whose chaotic behavior indicates the challenge of the problem.

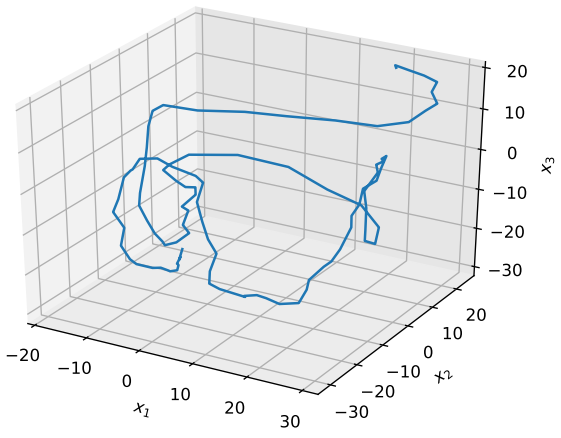


Figure 4.5. Chaotic path of the partial state variables (X_t^1, X_t^2, X_t^3) for $t = 1, 2, \ldots, 100$, simulated from the Lorenz-96 Model.

Algorithm 6 was applied to this example with the ensemble size $m = 50$, the iteration number $\mathcal{K} = 20$, $k_0 = \mathcal{K}/2$, and the learning rate $\epsilon_{t,k} = 0.5/k^{0.9}$ for $k = 1, 2, \ldots, \mathcal{K}$ and $t = 1, 2, \ldots, T$. At each stage t, the state was estimated by averaging over the ensembles

generated in iterations $k_0 + 1, k_0 + 2, \ldots, \mathcal{K}$. The accuracy of the estimate was measured using the root mean-squared error (RMSE) defined by

$$RMSE_t = ||\hat{\boldsymbol{X}}_t - \boldsymbol{X}_t||_2/\sqrt{p},$$

where $\hat{\boldsymbol{X}}_t$ denote the estimate of $\boldsymbol{X}_t$. For comparison, the EnKF algorithm was also applied to this example with the same ensemble size $m = 50$. To be fair, it was run in a similar way to LEnKF except that the Kalman gain matrix was estimated based on the ensemble, without the resampling step being performed, and the random error was drawn from $N(0, V_t)$ in the analysis step.

Figure 4.6 compares the estimates of X_t^1, X_t^2 and X_t^3 produced by LEnKF and EnKF for one simulated dataset. The plots are similar for the other components $X_t^4, \ldots, X_t^{40}$. The comparison shows that the EnKF and LEnKF produced comparable RMSE$_t$'s (see Figure 4.6 (d)). This is interesting as the EnKF is known to provide the optimal linear estimator of the conditional mean [26]. However, the LEnKF provided better uncertainty quantification for the estimates. For example, in Figure 4.6 (c), many state values are covered by the confidence band produced by the LEnKF, but not by the confidence band by the EnKF. This is consistent with the existing result that the EnKF scheme is known to underestimate the confidence intervals, see e.g. Saetrom and Omre [69].

Figure 4.7 (a) shows the coverage probabilities of the 95% confidence intervals produced by the EnKF and LEnKF, where the coverage probability was calculated by averaging over 40 state components at each stage $t \in \{1, 2, \ldots, 100\}$. Figure 4.7(b) shows the averaged coverage probabilities over 10 datasets. The comparison shows that the LEnKF produces the coverage probabilities closing to their nominal level, while the EnKF does not. This implies that the LEnKF is able to correctly quantify uncertainty of the estimates as t becomes large. This is a remarkable result given the nonlinear and dynamic nature of the Lorenz-96 model!

Table 4.2 summarizes the results produced by the two methods on 10 datasets. In LEnKF, two choices of k_0 were tried. For each dataset, we calculated the MeanRMSE by averaging RMSE$_t$ over the stages $t = 21, 22, \ldots, 100$. Similarly, we calculated the MeanCP by averaging CP$_t$'s over the stages $t = 21, 22, \ldots, 100$, where CP$_t$ denotes the coverage

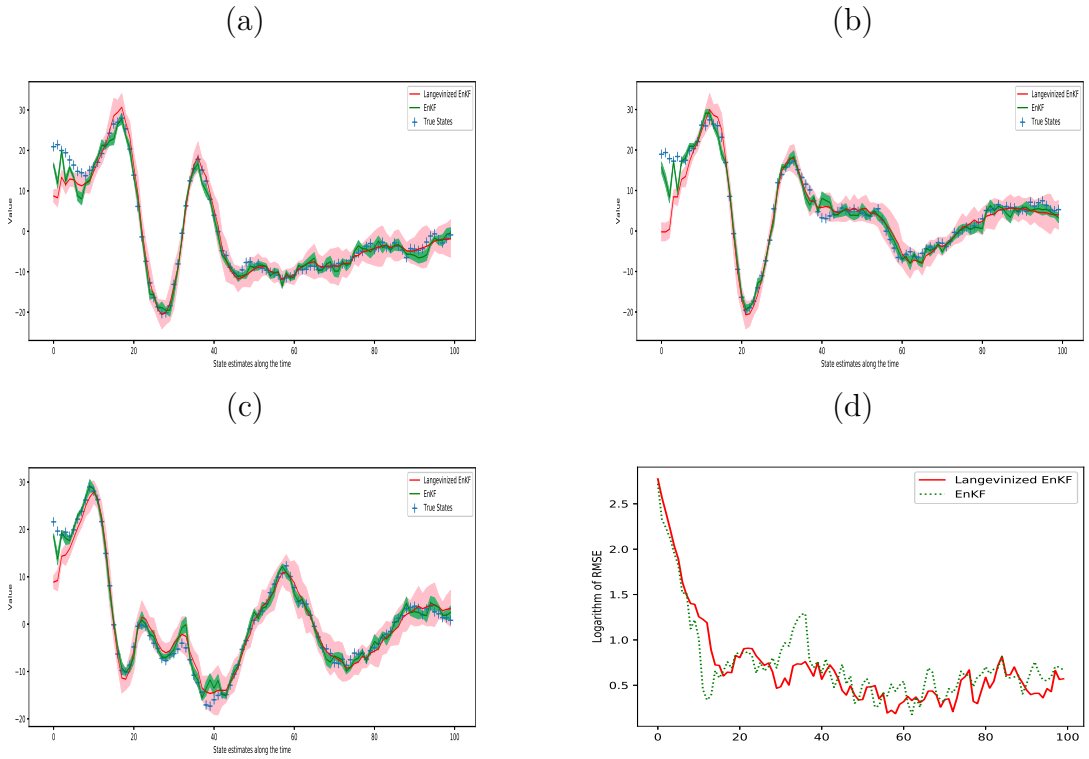


Figure 4.6. State estimates produced by the EnKF and LEnKF for the Lorenz-96 model with $t = 1, 2, \ldots, 100$: plots (a)-(c) show, respectively, the estimates of X_t^1, X_t^2 and X_t^3, where the true state values are represented by '+', the estimates are represented by solid lines, and their 95% confidence intervals are represented by shaded bands; plot (d) shows $\log(\text{RMSE}_t)$ along with stage t.

probability calculated for one dataset at stage t. Then their values were averaged over 10 datasets and denoted by "Ave-MeanRMSE" and "Ave-MeanCP", respectively. Table 4.2 also gives the CPU time cost by each method. Compared to EnKF, LEnKF produced slightly lower RMSEt's, but much more accurate uncertainty quantification for the estimates. We note that LEnKF also produced very good results with $k_0 = \mathcal{K} - 1$, which cost much less CPU time than with $k_0 = \mathcal{K}/2$.

In summary, the LEnKF can significantly outperform the EnKF for the Lorenz-96 model. The LEnKF can produce the same accurate state estimates as EnKF, but outperforms EnKF in uncertainty quantification.

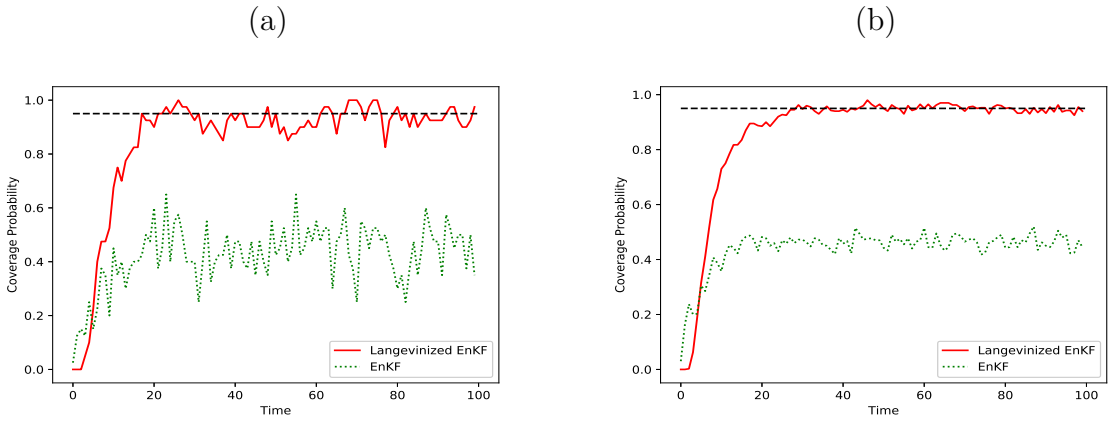


Figure 4.7. Coverage probabilities of the 95% confidence intervals produced by EnKF and LEnKF for Lorenz-96 Model for stage $t = 1, 2, \ldots, 100$: (a) coverage probabilities with one dataset; (b) coverage probabilities averaged over 10 datasets.

Table 4.2. Comparison of the EnKF and LEnKF, where the averages over 10 independent datasets are reported with the standard deviation given in the parentheses.

	k_0	Ave-MeanRMSE	Ave-MeanCP	CPU(s)
LEnKF	$\mathcal{K}/2$	1.702(0.0343)	0.948(0.0028)	6.377(0.3942)
	$\mathcal{K}-1$	1.714(0.0360)	0.947(0.0034)	3.350(0.0807)
EnKF		1.722(0.0230)	0.460(0.0029)	0.817(0.0426)

4.2.2 Online learning with LSTM neural networks

1) Reformulation of LSTM network

The LSTM network is a recurrent neural network architecture proposed by Hochreiter and Schmidhuber [70], which has been widely used for machine learning tasks in dealing with time series data. Compared to traditional recurrent neural networks, hidden Markov models and other sequence learning methods, LSTM is less sensitive to gap length of the data sequence. In addition, it is easy to train, less bothered by exploding and vanishing gradient problems. The LSTM network has been successfully used in natural language processing and handwriting recognition. It won the ICDAR handwriting competition 2009 [71] and achieved

a record 17.7% phoneme error rate on the classic TIMIT natural speech dataset [72]. In this section we show that the LEnKF is not only able to train LSTM networks as well as the stochastic gradient descent (SGD) method does, but also able to quantify uncertainty of the resulting estimates.

Consider an autoregressive model of order q, denoted by AR(q). Let $\boldsymbol{z}_t = (z_{t-q+1}, \cdots, z_{t-1}, z_t)$ denote the regression vector at stage t, and let $y_t = z_{t+1} \in \mathbb{R}^d$ denote the target output at stage t. The LSTM network with s hidden neurons is defined by the following set of equations:

$$\begin{aligned}
\eta_t &= h\left(\boldsymbol{W}^{(\eta)}\boldsymbol{z}_t + \boldsymbol{R}^{(\eta)}\boldsymbol{\psi}_{t-1} + \boldsymbol{b}^{(\eta)}\right), \\
\mathrm{i}_t &= \sigma\left(\boldsymbol{W}^{(\mathrm{i})}\boldsymbol{z}_t + \boldsymbol{R}^{(\mathrm{i})}\boldsymbol{\psi}_{t-1} + \boldsymbol{b}^{(\mathrm{i})}\right), \\
\boldsymbol{f}_t &= \sigma\left(\boldsymbol{W}^{(f)}\boldsymbol{z}_t + \boldsymbol{R}^{(f)}\boldsymbol{\psi}_{t-1} + \boldsymbol{b}^{(f)}\right), \\
\boldsymbol{c}_t &= \boldsymbol{\Lambda}_t^{(\mathrm{i})}\eta_t + \boldsymbol{\Lambda}_t^{(f)}\boldsymbol{c}_{t-1}, \\
\boldsymbol{o}_t &= \sigma\left(\boldsymbol{W}^{(o)}\boldsymbol{z}_t + \boldsymbol{R}^{(o)}\boldsymbol{\psi}_{t-1} + \boldsymbol{b}^{(o)}\right), \\
\boldsymbol{\psi}_t &= \boldsymbol{\Lambda}_t^{(o)} h\left(\boldsymbol{c}_t\right),
\end{aligned} \tag{4.6}$$

where $\boldsymbol{\Lambda}_t^{(f)} = \mathrm{diag}\left(\boldsymbol{f}_t\right), \boldsymbol{\Lambda}_t^{(\mathrm{i})} = \mathrm{diag}\left(\mathrm{i}_t\right)$, and $\boldsymbol{\Lambda}_t^{(o)} = \mathrm{diag}\left(\boldsymbol{o}_t\right)$. The activation function $h(\cdot)$ applies to vectors pointwisely and is commonly set to $tanh(\cdot)$. The sigmoid function $\sigma(\cdot)$ also applies pointwisely to the vector elements. In terms of LSTM networks, $\boldsymbol{z}_t \in \mathbb{R}^{qd}$ is called input vector, $\boldsymbol{c}_t \in \mathbb{R}^s$ is called the state vector, $\boldsymbol{\psi}_t \in \mathbb{R}^s$ is called the output vector, and i_t, $\boldsymbol{f}_t$ and $\boldsymbol{o}_t$ are called the input, forget and output gates, respectively. For the coefficient matrices and weight vectors, we have $\boldsymbol{W}^{(\eta)}, \boldsymbol{W}^{(\mathrm{i})}, \boldsymbol{W}^{(f)}, \boldsymbol{W}^{(o)} \in \mathbb{R}^{s\times qd}$, $\boldsymbol{R}^{(\eta)}, \boldsymbol{R}^{(\mathrm{i})}, \boldsymbol{R}^{(f)}, \boldsymbol{R}^{(o)} \in \mathbb{R}^{s\times s}$, and $\boldsymbol{b}^{(\eta)}, \boldsymbol{b}^{(\mathrm{i})}, \boldsymbol{b}^{(f)}, \boldsymbol{b}^{(o)} \in \mathbb{R}^s$. For initialization, we set $\boldsymbol{\psi}_0 = \boldsymbol{0}$, and $\boldsymbol{c}_0 = \boldsymbol{0}$. Given the output vector $\boldsymbol{\psi}_t$, we can model the target output y_t as

$$y_t = \boldsymbol{W}\boldsymbol{\psi}_t + \boldsymbol{b} + \boldsymbol{u}_t, \tag{4.7}$$

where $\boldsymbol{W} \in \mathbb{R}^{d\times s}, \boldsymbol{b} \in \mathbb{R}^d$, and $\boldsymbol{u}_t \sim N(0, \Gamma_t)$.

For convenience, we group the parameters of the LSTM model as $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{W}^{(\eta)}, \boldsymbol{R}^{(\eta)}, \boldsymbol{b}^{(\eta)}, \boldsymbol{W}^{(\mathrm{i})}, \boldsymbol{R}^{(\mathrm{i})}, \boldsymbol{b}^{(\mathrm{i})}, \boldsymbol{W}^{(f)}, \boldsymbol{R}^{(f)}, \boldsymbol{b}^{(f)}, \boldsymbol{W}^{(o)}, \boldsymbol{R}^{(o)}, \boldsymbol{b}^{(o)}\} \in \mathbb{R}^{n_\theta}$, where $n_\theta = 4s^2 + 4sqd + 4s + sd + d$. With

the state-augmentation approach, we can rewrite the LSTM model as a state-space model with a linear measurement equation as follows:

$$\begin{aligned}\begin{bmatrix}\boldsymbol{\theta}_t\\ \boldsymbol{c}_t\\ \boldsymbol{\psi}_t\\ \boldsymbol{\gamma}_t\end{bmatrix} &= \begin{bmatrix}\boldsymbol{\theta}_{t-1}\\ \boldsymbol{\Omega}\left(\boldsymbol{c}_{t-1}, \boldsymbol{z}_t, \boldsymbol{\psi}_{t-1}\right)\\ \tau\left(\boldsymbol{c}_t, \boldsymbol{z}_t, \boldsymbol{\psi}_{t-1}\right)\\ \boldsymbol{W}_t\boldsymbol{\psi}_t + \boldsymbol{b}_t\end{bmatrix} + \begin{bmatrix}\mathrm{e}_t\\ \boldsymbol{\zeta}_t\\ \boldsymbol{\xi}_t\\ \boldsymbol{\varepsilon}_t\end{bmatrix},\\ y_t &= \boldsymbol{\gamma}_t + \boldsymbol{v}_t,\end{aligned} \tag{4.8}$$

where $\boldsymbol{\varepsilon}_t \sim N(0, \alpha\Gamma_t)$ for some constant $0 < \alpha_t < 1$, $\boldsymbol{v}_t \sim N(0, (1-\alpha)\Gamma_t)$, and Γ_t is as defined in (4.7). Let $\boldsymbol{x}_t^T = (\boldsymbol{\theta}_t^T, \boldsymbol{c}_t^T, \boldsymbol{\psi}_t^T, \boldsymbol{\gamma}_t^T)$. Then

$$\boldsymbol{\pi}(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{z}_t) = \boldsymbol{\pi}(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}, \boldsymbol{z}_t)\boldsymbol{\pi}(\boldsymbol{c}_t|\boldsymbol{\theta}_t, \boldsymbol{c}_{t-1}, \boldsymbol{\psi}_{t-1}, \boldsymbol{z}_t)\boldsymbol{\pi}(\boldsymbol{\psi}_t|\boldsymbol{\theta}_t, \boldsymbol{c}_t, \boldsymbol{\psi}_{t-1}, \boldsymbol{z}_t)\boldsymbol{\pi}(\boldsymbol{\gamma}_t|\boldsymbol{\theta}_t, \boldsymbol{\psi}_t).$$

As in (3.22), we can rewrite the state-space model (4.8) as a dynamic system at each stage t:

$$\begin{aligned}\boldsymbol{x}_{t,k} &= \boldsymbol{x}_{t,k-1} + \frac{\epsilon_t}{2}\nabla_{\boldsymbol{x}} \log \boldsymbol{\pi}(\boldsymbol{x}_{t,k-1}|\boldsymbol{x}_{t-1}^s, \boldsymbol{z}_t) + \boldsymbol{\omega}_{t,k}\\ y_{t,k} &= H_t\boldsymbol{x}_{t,k} + \boldsymbol{v}_{t,k},\end{aligned} \tag{4.9}$$

where $\boldsymbol{x}_{t,k}$ denote an estimate of $\boldsymbol{x}_t$ obtained at iteration k for $k = 1, 2, \ldots, \mathcal{K}$, $y_{t,k} = y_t$ for $k = 1, 2, \ldots, \mathcal{K}$, $H_t = (0, I)$ such that $H_t x_t = \boldsymbol{\gamma}_t$, $\boldsymbol{\omega}_{t,k} \sim N(0, \epsilon_t I_p)$, p is the dimension of $\boldsymbol{x}_t$, and $\boldsymbol{v}_{t,k} \sim N(0, (1-\alpha)\Gamma_t)$. With this formulation, Algorithm 6 can be applied to train the LSTM model and and quantify uncertainty of the resulting estimate.

2) Wind stress data

We considered a wind stress dataset, which consists of gridded (at a 2×2 degrees resolution and corresponding to $d = 1470$ spatial locations) monthly summaries of meridional wind pseudo-stress collected from Jan 1961 to Feb 2002. For this dataset, we set $q = 6$ and $T = 300$, i.e., modeling the data of the first 300 months using an AR(6) LSTM model. The

data was scaled into the range $(-1, 1)$ in preprocessing and then scaled back to the original range in results reporting.

The LEnKF was first applied to this example. For the model part, we set $\mathrm{e}_t \sim N(0, 0.0001I)$, $\boldsymbol{\zeta}_t \sim N(0, 0.0001I)$, $\boldsymbol{\xi}_t \sim N(0, 0.0001I)$, $\boldsymbol{u}_t \sim N(0, 0.0001I)$. These model parameters are assumed to be known, although this is not necessary as discussed at Section 1.2. For this example, we have tried different settings for the model parameters. In general, a smaller variance setting will lead to a better fitting to the observations. For the algorithmic part, we set the ensemble size $m = 100$, $\mathcal{K} = 10$, $k_0 = 9$, $\alpha = 0.9$, the number of hidden neurons $s = 20$, and the learning rate $\epsilon_{t,k} = 0.0001/\max\{\kappa_b, k\}^{0.95}$ with $\kappa_b = 8$ for $k = 1, \cdots, \mathcal{K}$ and $t = 1, \cdots, T$. At each stage t, the wind stress was estimated by averaging over $\hat{y}_{t,k} = H_t \boldsymbol{x}_{t,k}$ for last $\mathcal{K}/2$ iterations. In addition, the credible interval for each component of $\boldsymbol{x}_t$ was calculated based on the ensemble obtained at stage t. Each run cost about 5334.5 CPU seconds. The results are summarized in Figure 4.8, where the wind stress estimates at four selected spatial locations and their 95% credible intervals are plotted along with stages.

For comparison, SGD was also applied to this example with the same setting as the LEnKF, i.e., they share the same learning rate and the same iteration number $\mathcal{K} = 10$ at each stage. The results are also summarized in Figure 4.8, where the wind stress estimates at three selected spatial locations are plotted along with stages. Each run of SGD cost about 15.9 CPU seconds. Since the LEnKF had an ensemble size $m = 100$, each chain cost only 53.3 CPU seconds. The LEnKF cost more CPU time and as return, it produced more samples for uncertainty quantification.

Further, we calculated the mean squared fitting error $||\hat{y}_t - y_t||_2^2$ for stages $t = 1, 2, \ldots, T$ and for both methods. The results are summarized in Figure 4.9, which indicates that the LEnKF produced slightly smaller fitting errors than SGD. Figure 4.10 shows the heat maps of the wind stress fitted by the LEnKF and SGD for six different months, August 1965, October 1969, December 1973, February 1978, April 1982, and June 1986. The comparison with the true heat maps indicates that both SGD and LEnKF can train the LSTM very well for this example.

In summary, this example shows that the LEnKF is not only able to train LSTM networks as does SGD, but also able to quantify uncertainty of the resulting estimates.

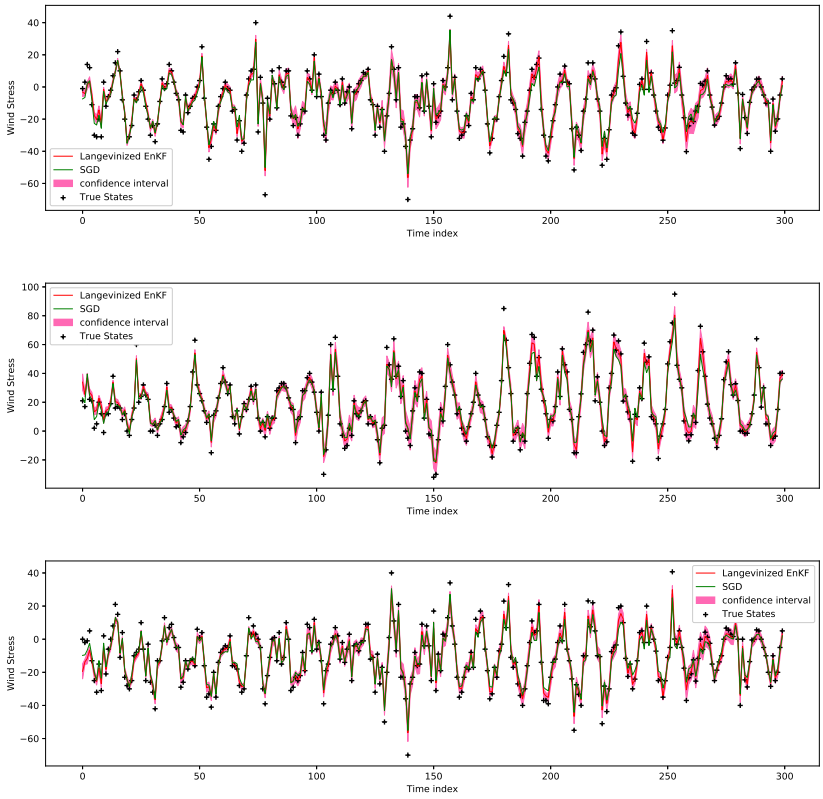


Figure 4.8. Wind stress estimates at three spatial locations and their 95% credible interval along with stages: the red line is for the LEnKF estimate; the pink shaded band is for credible intervals of the LEnKF, the green line is for the SGD estimate; and the blue cross '+' is for the true wind stress value.

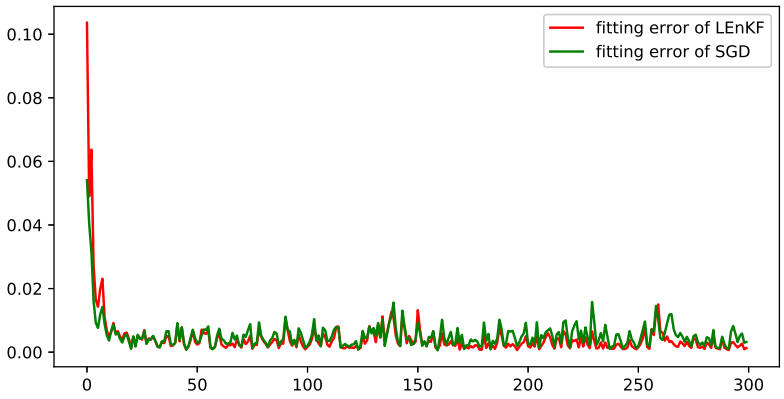


Figure 4.9. Comparison of the mean squared fitting errors produced by SGD, and LEnKF.

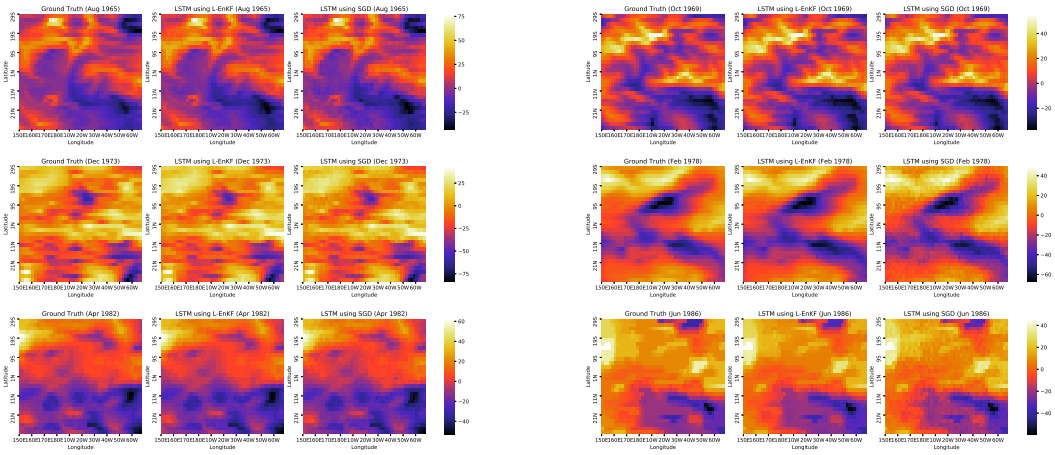


Figure 4.10. Heat maps of the wind stress fitted by the LEnKF and SGD for six different months, August 1965, October 1969, December 1973, February 1978, April 1982, and June 1986: For both left and right panels, the left, middle and right columns show the true heat map, the heat map fitted by LEnKF, and the heat map fitted by SGD, respectively.

5. EXTENSIONS OF THE LANGEVINIZED ENSEMBLE KALMAN FILTER

In the previous two chapters, we discussed the LEnKF algorithm for dynamic system (1.1), where the model error u_t and observation error η_t are zero-mean Gaussian random variables, and that the covariance matrices U_t and Γ_t and the propagator $g(\cdot)$ and H_t are all fully specified, i.e. containing no unknown parameters.

In the following two chapters, we focus on extensions of the LEnKF algorithm. In Section 5.1, we consider the second SSM (1.2), in which, unknown parameters exist. Simultaneously estimating the state variables $\{x_1, x_2, \ldots, x_t, \ldots\}$ and the parameters $\boldsymbol{\theta} = (\alpha, \beta, \zeta_x, \zeta_y)$ under the high-dimensional, big data and long series scenario is important to us. We propose the so-called stochastic approximation-Langevinized EnKF (SA-LEnKF) for simultaneously estimating the states and parameters of the dynamic system (1.2), where the parameters are estimated on the fly based on state variables simulated by the LEnKF algorithm under the framework of stochastic approximation [44]. The proposed algorithm is general; it work well for all types of parameters: additive, multiplicative and stochastic. Under mild conditions, we establish the consistency of the resulting parameter estimator and the ergodicity of the SA-LEnKF. In Section 5.2, we consider the third SSM (1.3), where $f(\cdot)$ deviate from Gaussian but no unknown parameters exist. We provide details of the Extended LEnKF algorithm for this non-Gaussian dynamic system as well as convergence analysis.

These two proposed extended methods can be efficiently used in uncertainty quantification for dynamic systems. As an advantage inherited from the LEnKF, they can work well for long series, large scale and high-dimensional dynamic systems.

5.1 Langevinized Ensemble Kalman filter with unknown parameters

5.1.1 The SA-LEnKF algorithm

The SA-LEnKF is a combination of the stochastic approximation algorithm [44] and the LEnKF, where the parameters in (1.2) are estimated on the fly based on state variables simulated by the LEnKF under the framework of stochastic approximation. As in Section

3.2.1, we assume that at each stage t, y_t has been partitioned into $B_t = N_t/n_t$ blocks such that $y_{t,k} = H_{t,k}x_t + v_{t,k}$, $k = 1, 2, \ldots B_t$, where $y_{t,k}$ is a block of n_t observations randomly drawn from $y_t = \{y_{t,1}, \ldots, y_{t,B_t}\}$, $v_{t,k} \sim N(0, V_t(\zeta_y))$ for all k, and $v_{t,k}$'s are mutually independent. It is easy to derive that for any given estimate of $\boldsymbol{\theta}$, conditioned on the state x_{t-1}, $y_{t,k}$ follows a multivariate normal distribution $N_{n_t}(H_{t,k}(\beta)g(x_{t-1}, \alpha), H_{t,k}(\beta)U(\zeta_x)H_{t,k}^T(\beta) + V_t(\zeta_y))$ with the density function denoted by $p(y_{t,k}|x_{t-1}, \boldsymbol{\theta})$. Let $\psi(y_{t,k}, x_{t-1}, \boldsymbol{\theta}) = \partial \log p(y_{t,k}|x_{t-1}, \boldsymbol{\theta})/\partial\boldsymbol{\theta}$. Let $\{\gamma_t\}$ be a positive and non-increasing sequence satisfying Assumption 5.1 (given in Section 5.1.3). The SA-LEnKF algorithm can be summarized as Algorithm 7, where, for notational simplicity, we have depressed the parameters of $H_{t,k}(\beta)$, $g(x_{t-1}, \alpha)$, $U(\zeta_x)$ and $V_t(\zeta_y)$.

Remark 5.1. *In the scenario that $\boldsymbol{\pi}(y_t|x_{t-1}, \boldsymbol{\theta})$ is not analytically available, $\nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_{t,k}|\tilde{x}^{\mathrm{i}}_{t-1,k-1}, \boldsymbol{\theta}_{t,k-1})$ can be estimated based on Fisher's identity (see, e.g., [73]):*

$$\nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_t|x_{t-1}, \boldsymbol{\theta}) = \int \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_t, x_t|x_{t-1}, \boldsymbol{\theta})\boldsymbol{\pi}(x_t|y_t, x_{t-1}, \boldsymbol{\theta})dx_t,$$

which implies the following procedure (in the notation of Algorithm 7):

- *For each $\mathrm{i} = 1, 2, \ldots, m$, simulate m samples from the distribution*

$$\boldsymbol{\pi}(x_{t,k}|y_{t,k}, \tilde{x}^{\mathrm{i}}_{t-1,k-1}, \boldsymbol{\theta}_{t,k-1}) \propto \boldsymbol{\pi}(x_{t,k}|\tilde{x}^{\mathrm{i}}_{t-1,k-1}, \boldsymbol{\theta}_{t,k-1})\boldsymbol{\pi}(y_{t,k}|x_{t,k}, \boldsymbol{\theta}_{t,k-1}),$$

  and denote the samples by $\{\tilde{x}^{\mathrm{j}}_{t,k} : \mathrm{j} = 1, 2, \ldots, m\}$. The simulation can be done by a short run of the Metropolis-Hastings algorithm or SGLD.

- *Calculate $\Psi(y_{t,k}, \tilde{\boldsymbol{x}}_{t-1,k-1}, \boldsymbol{\theta}_{t,k-1})$ by*

$$\Psi(y_{t,k}, \tilde{\boldsymbol{x}}_{t-1,k-1}, \boldsymbol{\theta}_{t,k-1}) = \frac{N_t}{n_t m m} \sum_{\mathrm{i}=1}^{m} \sum_{\mathrm{j}=1}^{m} \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_t, \tilde{x}^{\mathrm{j}}_{t,k}|\tilde{x}^{\mathrm{i}}_{t-1,k-1}, \boldsymbol{\theta}_{t,k-1})$$
$$= \frac{N_t}{n_t m m} \sum_{\mathrm{i}=1}^{m} \sum_{\mathrm{j}=1}^{m} [\nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_t|\tilde{x}^{\mathrm{j}}_{t,k}, \boldsymbol{\theta}_{t,k-1}) + \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(\tilde{x}^{\mathrm{j}}_{t,k}|\tilde{x}^{\mathrm{i}}_{t-1,k-1}, \boldsymbol{\theta}_{t,k-1})].$$

Alternative to on-line estimation, one can estimate the parameters $\boldsymbol{\theta}$ in an off-line manner as described in Algorithm 8.

Algorithm 7: SA-LEnKF-Online for simultaneous state and parameter estimation

(i) Initialization: Start with an initial ensemble $x_{1,0}^{a,1}, x_{1,0}^{a,2}, \ldots, x_{1,0}^{a,m}$ drawn from the prior distribution $\boldsymbol{\pi}(x_1)$, where m denotes the ensemble size.

for *t=1,2,…, T* **do**

Set $\mathcal{X}_t = \emptyset$ and k_0 as the common burn-in period.

for *k=1,2,…, K* **do**

(ii) Subsampling: Draw without replacement a mini-batch data (i.e., a block), denoted by $(y_{t,k}, H_{t,k})$, of size n_t from the full dataset of size N_t. Set $Q_{t,k} = \epsilon_{t,k} I_p$, $R_t = 2V_t$, and the Kalman gain matrix $K_{t,k} = Q_{t,k} H_{t,k}^T (H_{t,k} Q_{t,k} H_{t,k}^T + R_t)^{-1}$.

for *i=1,2,…, m* **do**

(iii) Importance resampling: If $t > 1$, calculate importance weights $\omega_{t,k-1,\mathrm{j}}^{\mathrm{i}} = \boldsymbol{\pi}(x_{t,k-1}^{a,\mathrm{i}} | x_{t-1,\mathrm{j}}) = \phi(x_{t,k-1}^{a,\mathrm{i}} : g(x_{t-1,\mathrm{j}}), U_t)$ for $\mathrm{j} = 1, 2, \ldots, |\mathcal{X}_{t-1}|$, where $\phi(\cdot)$ denotes a Gaussian density, and $x_{t-1,\mathrm{j}} \in \mathcal{X}_{t-1}$ denotes the jth sample in $\mathcal{X}_{t-1}$; if $k = 1$, set $x_{t,0}^{a,\mathrm{i}} = g(x_{t-1,\mathcal{K}}^{a,\mathrm{i}}) + u_t^{a,\mathrm{i}}$ and $u_t^{a,\mathrm{i}} \sim N(0, U_t)$. Resample $s \in \{1, 2, \ldots, |\mathcal{X}_{t-1}|\}$ with a probability $\propto \omega_{t,k-1,s}^{\mathrm{i}}$, i.e., $P(S_{t,k,\mathrm{i}} = s) = \omega_{t,k-1,s}^{\mathrm{i}} / \sum_{\mathrm{j}=1}^{|\mathcal{X}_{t-1}|} \omega_{t,k-1,\mathrm{j}}^{\mathrm{i}}$, and denote the sample drawn from $\mathcal{X}_{t-1}$ by $\tilde{x}_{t-1,k-1}^{\mathrm{i}}$.

(iv) Forecast: Draw $w_{t,k}^{\mathrm{i}} \sim N_p(0, \frac{n_t}{N_t} Q_{t,k})$. If $t = 1$, set

$$x_{t,k}^{f,\mathrm{i}} = x_{t,k-1}^{a,\mathrm{i}} - \epsilon_{t,k} \frac{n_t}{2N_t} \nabla \log \boldsymbol{\pi}(x_{t,k-1}^{a,\mathrm{i}}) + w_{t,k}^{\mathrm{i}}, \tag{5.1}$$

where $\boldsymbol{\pi}(\cdot)$ denotes the prior distribution of x_1. If $t > 1$, set

$$x_{t,k}^{f,\mathrm{i}} = x_{t,k-1}^{a,\mathrm{i}} - \epsilon_{t,k} \frac{n_t}{2N_t} U_t^{-1} (x_{t,k-1}^{a,\mathrm{i}} - g(\tilde{x}_{t-1,k-1}^{\mathrm{i}})) + w_{t,k}^{\mathrm{i}}. \tag{5.2}$$

(v) Analysis: Draw $v_{t,k}^{\mathrm{i}} \sim N_n(0, \frac{n_t}{N_t} R_t)$ and set

$$x_{t,k}^{a,\mathrm{i}} = x_{t,k}^{f,\mathrm{i}} + K_{t,k}(y_{t,k} - H_{t,k} x_{t,k}^{f,\mathrm{i}} - v_{t,k}^{\mathrm{i}}) \stackrel{\Delta}{=} x_{t,k}^{f,\mathrm{i}} + K_{t,k}(y_{t,k} - y_{t,k}^{f,\mathrm{i}}). \tag{5.3}$$

(vi) Sample collection: If $k > k_0$, add the sample $x_{t,k}^{a,\mathrm{i}}$ into the set $\mathcal{X}_t$.

(vii) Parameter update: If $t > 1$, calculate

$$\boldsymbol{\theta}_{t,k} = \boldsymbol{\theta}_{t,k-1} + \gamma_{t,k} \Psi(y_{t,k}, \tilde{\boldsymbol{x}}_{t-1,k-1}, \boldsymbol{\theta}_{t,k-1}), \tag{5.4}$$

where $\boldsymbol{\theta}_{t,k}$ denotes the estimate of $\boldsymbol{\theta}$ obtained at iteration k of stage t, $\tilde{\boldsymbol{x}}_{t-1,k-1} = (\tilde{x}_{t-1,k-1}^1, \tilde{x}_{t-1,k-1}^2, \ldots, \tilde{x}_{t-1,k-1}^m)$, and

$$\Psi(y_{t,k}, \tilde{\boldsymbol{x}}_{t-1,k-1}, \boldsymbol{\theta}_{t,k-1}) = \frac{N_t}{m n_t} \sum_{\mathrm{i}=1}^{m} \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_{t,k} | \tilde{x}_{t-1,k-1}^{\mathrm{i}}, \boldsymbol{\theta}_{t,k-1}), \tag{5.5}$$

Algorithm 8: SA-LEnKF-offline for simultaneous state and parameter estimation

for *l=0,1,2,…,L* **do**

Initialize the estimate $\boldsymbol{\theta}_0$ and set k_0 as the common burn-in period.

(i) Initialization: If $l = 0$, draw an ensemble $\{x_{l,1,0}^{a,1}, x_{l,1,0}^{a,2}, \ldots, x_{l,1,0}^{a,m}\}$ from the prior distribution $\boldsymbol{\pi}(x_1)$; otherwise, set $x_{l,1,0}^{a,\mathrm{i}} = x_{l-1,1,0}^{a,\mathrm{i}}$ for $\mathrm{i} = 1, 2, \ldots, m$.

for *t = 1, 2, …, T* **do**

Set $\mathcal{X}_{l,t} = \emptyset$.

for *k = 1, 2, …, K* **do**

(ii) Subsampling: Draw without replacement a mini-batch data, denoted by $(y_{l,t,k}, H_{l,t,k})$, of size n_t from the full dataset of size N_t. Set $Q_{l,t,k} = \epsilon_{l,t,k} I_p$, $R_{l,t} = 2V_{l,t}$, and the Kalman gain matrix $K_{l,t,k} = Q_{l,t,k} H_{l,t,k}^T (H_{l,t,k} Q_{l,t,k} H_{l,t,k}^T + R_{l,t})^{-1}$.

for *i = 1, 2, …, m* **do**

(iii) Importance resampling: If $t > 1$, calculate importance weights $\omega_{l,t,k-1,\mathrm{j}}^{\mathrm{i}} = \boldsymbol{\pi}(x_{l,t,k-1}^{a,\mathrm{i}} | x_{l,t-1,\mathrm{j}}) = \phi(x_{l,t,k-1}^{a,\mathrm{i}} : g(x_{l,t-1,\mathrm{j}}), U_{l,t})$ for $\mathrm{j} = 1, 2, \ldots, |\mathcal{X}_{l,t-1}|$, where $\phi(\cdot)$ denotes a Gaussian density, and $x_{l,t-1,\mathrm{j}} \in \mathcal{X}_{l,t-1}$ denotes the jth sample in $\mathcal{X}_{l,t-1}$; if $k = 1$, set $x_{l,t,0}^{a,\mathrm{i}} = g(x_{l,t-1,\mathcal{K}}^{a,\mathrm{i}}) + u_{l,t}^{a,\mathrm{i}}$ and $u_{l,t}^{a,\mathrm{i}} \sim N(0, U_{l,t})$. Resample $s \in \{1, 2, \ldots, |\mathcal{X}_{l,t-1}|\}$ with a probability $\propto \omega_{l,t,k-1,s}^{\mathrm{i}}$, i.e., $P(S_{l,t,k,\mathrm{i}} = s) = \omega_{l,t,k-1,s}^{\mathrm{i}} / \sum_{\mathrm{j}=1}^{|\mathcal{X}_{l,t-1}|} \omega_{l,t,k-1,\mathrm{j}}^{\mathrm{i}}$, and denote the sample drawn from $\mathcal{X}_{l,t-1}$ by $\tilde{x}_{l,t-1,k-1}^{\mathrm{i}}$.

(iv) Forecast: Draw $w_{l,t,k}^{\mathrm{i}} \sim N_p(0, \frac{n_t}{N_t} Q_{l,t,k})$. If $t = 1$, set

$$x_{l,t,k}^{f,\mathrm{i}} = x_{l,t,k-1}^{a,\mathrm{i}} - \epsilon_{l,t,k} \frac{n_t}{2N_t} \nabla \log \boldsymbol{\pi}(x_{l,t,k-1}^{a,\mathrm{i}}) + w_{l,t,k}^{\mathrm{i}}, \tag{5.6}$$

where $\boldsymbol{\pi}(\cdot)$ denotes the prior distribution of x_1. If $t > 1$, set

$$x_{l,t,k}^{f,\mathrm{i}} = x_{l,t,k-1}^{a,\mathrm{i}} - \epsilon_{l,t,k} \frac{n_t}{2N_t} U_{l,t}^{-1} (x_{l,t,k-1}^{a,\mathrm{i}} - g(\tilde{x}_{l,t-1,k-1}^{\mathrm{i}})) + w_{l,t,k}^{\mathrm{i}}. \tag{5.7}$$

(v) Analysis: Draw $v_{l,t,k}^{\mathrm{i}} \sim N_n(0, \frac{n_t}{N_t} R_{l,t})$ and set

$$x_{l,t,k}^{a,\mathrm{i}} = x_{l,t,k}^{f,\mathrm{i}} + K_{l,t,k}(y_{l,t,k} - H_{l,t,k} x_{l,t,k}^{f,\mathrm{i}} - v_{l,t,k}^{\mathrm{i}}) \triangleq x_{l,t,k}^{f,\mathrm{i}} + K_{l,t,k}(y_{l,t,k} - y_{l,t,k}^{f,\mathrm{i}}). \tag{5.8}$$

(vi) Sample collection: If $k > k_0$, add the sample $x_{l,t,k}^{a,\mathrm{i}}$ into $\mathcal{X}_{l,t}$.

(vii) Parameter update: Calculate

$$\boldsymbol{\theta}_{l+1} = \boldsymbol{\theta}_l + \frac{\gamma_{l+1}}{(T-1)(\mathcal{K}-k_0)} \sum_{t=2}^{T} \sum_{k=k_0+1}^{\mathcal{K}} \Psi(y_{l,t,k}, \tilde{\boldsymbol{x}}_{l,t-1,k-1}, \boldsymbol{\theta}_l), \tag{5.9}$$

where $\tilde{\boldsymbol{x}}_{l,t-1,k-1} = (\tilde{x}_{l,t-1,k-1}^1, \tilde{x}_{l,t-1,k-1}^2, \ldots, \tilde{x}_{l,t-1,k-1}^m)$, and

$$\Psi(y_{l,t,k}, \tilde{\boldsymbol{x}}_{l,t-1,k-1}, \boldsymbol{\theta}_l) = \frac{N_t}{m n_t} \sum_{\mathrm{i}=1}^{m} \nabla_{\boldsymbol{\theta}} \log p(y_{l,t,k} | \tilde{x}_{l,t-1,k-1}^{\mathrm{i}}, \boldsymbol{\theta}_l).$$

As shown in Algorithm 8, in each sweep l of the algorithm, all state variables are first estimated based on the current estimate of $\boldsymbol{\theta}$, and then the estimate of $\boldsymbol{\theta}$ is updated based on all state estimates and all data $\boldsymbol{y}_{1:T}$. By contrast, in Algorithm 7, each update of $\boldsymbol{\theta}$ is based on only the state estimates and the data obtained at the current stage. Compared to the on-line algorithm, the offline algorithm can be much slower. Due to its similarity with the particle buffered stochastic gradient MCMC method [38], the offline algorithm is treated as a baseline in this dissertation.

5.1.2 Convergence analysis

In this section, we analyze the consistency of the resulting parameter estimator and ergodicity of the SA-LEnKF algorithm. Detailed proofs are outlined in Section 5.1.3.

The SA-LEnKF is essentially an adaptive SGRLD algorithm if we view the LEnKF at each stage as a SGRLD sampler by the explanation given in Section 3.1.1. The algorithm is said "adaptive" because the parameter vector $\boldsymbol{\theta}$ changes from iteration to iteration. The link between the SA-LEnKF and stochastic approximation can be explained as follows. Since the joint distribution of $\boldsymbol{y}_{1:T} = \{y_1, y_2, \ldots, y_T\}$ can be generally factorized as $\boldsymbol{\pi}(\boldsymbol{y}_{1:T}|\boldsymbol{\theta}) = \boldsymbol{\pi}(y_1)\boldsymbol{\pi}(y_2|y_1, \boldsymbol{\theta})\boldsymbol{\pi}(y_3|\boldsymbol{y}_{1:2}, \boldsymbol{\theta}) \cdots \boldsymbol{\pi}(y_T|\boldsymbol{y}_{1:T-1}, \boldsymbol{\theta})$, we have

$$\nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(\boldsymbol{y}_{1:T}|\boldsymbol{\theta}) = \sum_{t=2}^{T} \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}).$$

Therefore, by assuming that the dynamic system (1.2) is stationary, $\boldsymbol{\theta}$ can be estimated in an on-line manner by solving the following equation at each stage t:

$$\nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) = 0, \tag{5.10}$$

where the estimate obtained at stage $t-1$ is passed on to stage t as the initial value. Note that solving (5.10) is to estimate $\boldsymbol{\theta}$ using only the information contained in the data transition

from $\boldsymbol{y}_{1:t-1}$ to $\boldsymbol{y}_t$. By recalling that the state variable x_t is a latent variable in the model (1.2), we have

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) &= \int \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_t, x_{t-1}|\boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \boldsymbol{\pi}(x_{t-1}|\boldsymbol{y}_{1:t}, \boldsymbol{\theta}) dx_{t-1} \\
&= \int \int \nabla_{\boldsymbol{\theta}} \log \left\{ \boldsymbol{\pi}(y_t|x_{t-1}, \boldsymbol{\theta}) \boldsymbol{\pi}(x_{t-1}|\boldsymbol{y}_{1:t-1}) \right\} \boldsymbol{\pi}(x_t|\boldsymbol{y}_{1:t}, \boldsymbol{\theta}) \boldsymbol{\pi}(x_{t-1}|x_t, \boldsymbol{y}_{1:t}, \boldsymbol{\theta}) dx_{t-1} dx_t \\
&= \int \int \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_t|x_{t-1}, \boldsymbol{\theta}) \boldsymbol{\pi}(x_t|\boldsymbol{y}_{1:t}, \boldsymbol{\theta}) \boldsymbol{\pi}(x_{t-1}|x_t, \boldsymbol{y}_{1:t}, \boldsymbol{\theta}) dx_{t-1} dx_t \\
&= \int \int \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_t|x_{t-1}, \boldsymbol{\theta}) \boldsymbol{\pi}(x_t|\boldsymbol{y}_{1:t}, \boldsymbol{\theta}) \boldsymbol{\pi}(x_{t-1}|x_t, \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) dx_{t-1} dx_t,
\end{aligned} \tag{5.11}$$

where the first equality follows from the Fisher identity, and $\boldsymbol{\pi}(x_{t-1}|x_t, \boldsymbol{y}_{1:t}, \boldsymbol{\theta}) = \boldsymbol{\pi}(x_{t-1}|x_t, \boldsymbol{y}_{1:t-1}, \boldsymbol{\theta})$ used in obtaining the last equality follows from the Markov property of the model (1.2). This justifies the the parameter updating step of Algorithm 7, which is to solve (5.10) under the framework of stochastic approximation.

Theorem 5.1.1 concerns the convergence of the parameter estimates, whose proof follows that of Theorem 5.1.3 (given in Section 5.1.3) directly.

Theorem 5.1.1 (Convergence of $\boldsymbol{\theta}_{t,k}$)**.** *Suppose that the dynamic system (1.1) is stationary; the covariance matrix V_t satisfies the condition $\lambda_l \leq \inf_t \lambda_{\min}(V_t) \leq \sup_t \lambda_{\max}(V_t) \leq \lambda_u$ for some positive constants λ_l and λ_u, where $\lambda_{\max}(\cdot)$ and $\lambda_{\min}(\cdot)$ denote, respectively, the largest and smallest eigenvalues of a matrix; the learning rate sequence $\{\epsilon_{t,k} : k = 1, 2, \ldots, \mathcal{K}, t = 1, 2, \ldots\}$ is sufficiently small such that $\epsilon_{t,k} \leq 2m/(3M^2)$ for all t and k; and Assumptions 5.1-5.6 (given in Section 5.1.3) hold for each stage t. Then there exists a constant ζ and a root $\boldsymbol{\theta}^* \in \{\boldsymbol{\theta} : \sum_{t=2}^{T} \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(y_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) = 0\}$ such that*

$$E\|\boldsymbol{\theta}_{t,k} - \boldsymbol{\theta}^*\|^2 \leq \zeta \gamma_{t,k}.$$

Theorem 5.1.1 provides a finite iteration analysis for the convergence of $\boldsymbol{\theta}_{t,k}$, which holds for each stage t. By Assumption 5.1 (given in Section 5.1.3), $\{\gamma_{t,k} : k = 1, 2, \ldots, \mathcal{K},\ t = 1, 2, \ldots\}$ decreases in both t and k, we have

$$\lim_{t \to \infty} E\|\boldsymbol{\theta}_{t,k} - \boldsymbol{\theta}^*\|^2 \to 0, \quad \forall\ k = 1, 2, \ldots, \mathcal{K}.$$

A practical choice of $\{\gamma_{t,k}\}$ is $\gamma_{t,k} = c_1/(tk)^\zeta$ for some constants $c_1 > 0$ and $\zeta \in (0.5, 1]$. Another possible choice is $\gamma_{t,k} = c_2/\max\{c_2, (t-1)\mathcal{K} + k\}^\zeta$ for some constants $c_2 > 0$, $\mathcal{K} > 0$, and $\zeta \in (0.5, 1]$.

Theorem 5.1.2 concerns the convergence of the state samples, whose proof follows that of Theorem 5.1.4 (given in Section 5.1.3) directly.

Theorem 5.1.2 (Ergodicity of SA-LEnKF-Online)**.** *Suppose the conditions of Theorem 5.1.1 and Assumption 5.7 (given in Section 5.1.3) hold. Assume that* $\epsilon_{t,k} \to 0$ *as* $k \to \infty$*, and* $\sum_{k=1}^{\mathcal{K}} \epsilon_{t,k} \to \infty$ *and* $[\sum_{k=1}^{\mathcal{K}} \epsilon_{t,k}^2]/[\sum_{k=1}^{\mathcal{K}} \epsilon_{t,k}] \to 0$ *as* $\mathcal{K} \to \infty$*. For a smooth function* ϕ*, define* $\hat{\phi}_t = \frac{1}{\mathcal{K}} \sum_{k=1}^{\mathcal{K}} \phi(x_{t,k})$ *and* $\bar{\phi}_t = \int \phi(x_t)\boldsymbol{\pi}(x_t|\boldsymbol{\theta}^*, \boldsymbol{y}_{1:t})dx_t$*. Then,*

$$E|\hat{\phi}_t - \bar{\phi}_t|^2 \to 0, \quad \text{as } \mathcal{K} \to \infty. \tag{5.12}$$

Theorems 5.1.1 and 5.1.2 imply that the SA-LEnKF-Online can be used for uncertainty quantification for dynamic systems.

5.1.3 Proofs for the convergence of the SA-LEnKF algorithm

In this section, we provide the detailed proofs of the SA-LEnKF algorithm.

1) Stochastic approximation

The stochastic approximation algorithm [44] is the prototype of many adaptive algorithms, which aims to solve an expectation equation given by

$$h(\boldsymbol{\theta}) = \int_{\mathcal{X}} \mathcal{H}(\boldsymbol{\theta}, \boldsymbol{x})\boldsymbol{\pi}_{\boldsymbol{\theta}}(d\boldsymbol{x}) = 0,$$

where $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$, $\boldsymbol{\theta} \in \boldsymbol{\Theta} \subset \mathbb{R}^m$, $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{x})$ denotes a distribution parameterized by $\boldsymbol{\theta}$, and $\mathcal{H}(\boldsymbol{\theta}, \boldsymbol{x})$ and $h(\boldsymbol{\theta})$ are called the random-field and mean-filed functions, respectively. The algorithm works by iterating between the following two steps:

(i) Simulate $\boldsymbol{x}_{k+1}$ from the transition kernel $\Pi_{\boldsymbol{\theta}_k}(\boldsymbol{x}_k, \cdot)$, which admits $\boldsymbol{\pi}_{\boldsymbol{\theta}_k}(\boldsymbol{x})$ as the invariant distribution.

(ii) Update $\boldsymbol{\theta}_k$ by setting $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \omega_{k+1}\mathcal{H}(\boldsymbol{\theta}_k, \boldsymbol{x}_{k+1}) + \omega_{k+1}^2\rho(\boldsymbol{\theta}_k, \boldsymbol{x}_{k+1})$, where $\rho(\cdot,\cdot)$ denotes a bias term.

In particular, the algorithm samples $\boldsymbol{x}$ from a transition kernel $\Pi_{\boldsymbol{\theta}_k}(\cdot,\cdot)$ instead of the distribution $\boldsymbol{\pi}_{\boldsymbol{\theta}_k}(\cdot)$ exactly, which leads to a Markov state-dependent noise $\mathcal{H}(\boldsymbol{\theta}_k, \boldsymbol{x}_{k+1}) - h(\boldsymbol{\theta}_k)$. In addition, it allows for a higher-order bias term $\rho(\cdot,\cdot)$ in parameter updating. The limiting value of $\boldsymbol{\theta}_k$ will not be affected by the bias term, provided that the bias term satisfies certain conditions, e.g., $\rho(\boldsymbol{\theta}, \boldsymbol{x})$ is bounded. See Benveniste, Métivier, and Priouret [74] for more discussions on this issue.

2) An Adaptive LEnKF algorithm

By the proof of Theorem 3.3.1, the LEnKF algorithm with parameter estimation forms a type of adaptive stochastic gradient Riemannian Langevin dynamics (SGRLD) algorithm:

$$\begin{aligned} x_k &= x_{k-1} + \frac{\epsilon_k}{2}\Sigma_k\nabla_x\tilde{l}(\boldsymbol{\theta}_{k-1}, x_{k-1}) + \sqrt{\epsilon_k}\mathrm{e}_k, \\ \boldsymbol{\theta}_k &= \boldsymbol{\theta}_{k-1} + w_k\mathcal{H}(\boldsymbol{\theta}_{k-1}, x_k), \end{aligned} \tag{5.13}$$

where e_k is a zero mean Gaussian random error with covariance $\mathrm{Var}(\mathrm{e}_k) = \Sigma_k$ and $\nabla_x\tilde{l}(\boldsymbol{\theta}_{k-1}, x_{k-1})$ denotes an unbiased estimate of $\nabla_x l(\boldsymbol{\theta}_{k-1}, x_k) = \nabla_x \log \boldsymbol{\pi}(x_{k-1}|y_k, \boldsymbol{\theta}_{k-1})$. Convergence of adaptive stochastic gradient MCMC algorithms has been studied in a number of papers (see e.g. [75], [76] and [77]).

For the LEnKF algorithm, as shown in Section 3.3, Σ_k can be expressed as

$$\Sigma_k = I - K_kH_k = I - \epsilon_kH_k^T(\epsilon_kH_kH_k^T + R_k)^{-1}H_k,$$

which implies that Σ_k has bounded eigenvalues for all $t \geq 1$ as long as R_k is positive definite and has bounded eigenvalues for all $t \geq 1$. Moreover, $0 < \lambda_{\min}(\Sigma_k) \leq \lambda_{\max}(\Sigma_k) < 1$ for all $t \geq 1$, where $\lambda_{\min}(\cdot)$ and $\lambda_{\max}(\cdot)$ denote the smallest and largest eigenvalues, respectively.

Since Σ_k can be a function of $\boldsymbol{\theta}_{k-1}$, we define $L(\boldsymbol{\theta}_{k-1}, x_{k-1}) = \Sigma_k l(\boldsymbol{\theta}_{k-1}, x_{k-1})$, and $\tilde{L}(\boldsymbol{\theta}_{k-1}, x_{k-1}) = \Sigma_k\tilde{l}(\boldsymbol{\theta}_{k-1}, x_k)$ in what follows. Obviously, $\nabla_x\tilde{L}(\boldsymbol{\theta}_{k-1}, x_{k-1}) = \Sigma_k\nabla_x\tilde{l}(\boldsymbol{\theta}_{k-1}, x_k)$.

3) Convergence of $\boldsymbol{\theta}_k$

The following conditions are assumed for the dynamic system (1.2).

Assumption 5.1. *There exist a constant δ and a vector $\boldsymbol{\theta}^*$ such that $\langle \boldsymbol{\theta} - \boldsymbol{\theta}^*, h(\boldsymbol{\theta}) \rangle \leq -\delta \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$ for any $\boldsymbol{\theta} \in \Theta$. The step sizes $\{\gamma_{t,k}\}_{t,k\in\mathbb{N}}$ is a positive sequence decreasing in both t and k such that*

$$\lim_{t\to\infty} \gamma_{t,k} = 0, \quad \lim_{k\to\infty} \gamma_{t,k} \to 0, \tag{5.14}$$

and for any $t \in \mathbb{N}$, the following conditions hold:

$$\sum_{k=1}^{\infty} \gamma_{t,k} = \infty, \quad \liminf_{k\to\infty} 2\delta \frac{\gamma_{t,k}}{\gamma_{t,k+1}} + \frac{\gamma_{t,k+1} - \gamma_{t,k}}{\gamma_{t,k+1}^2} > 0. \tag{5.15}$$

Assumption 5.2. *$L(\boldsymbol{\theta}, x)$ is M-smooth on $\boldsymbol{\theta}$ and x with $M > 0$, and (m, b) dissipative on x. In other words, for any $x, x_1, x_2 \in \mathcal{X}$ and $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \Theta$, the following inequalities are satisfied:*

$$\|\nabla_x L(\boldsymbol{\theta}_1, x_1) - \nabla_x L(\boldsymbol{\theta}_2, x_2)\| \leq M\|x_1 - x_2\| + M\|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|, \tag{5.16}$$

$$\langle \nabla_x L(\boldsymbol{\theta}, x), x \rangle \leq b - m\|x\|^2. \tag{5.17}$$

The smoothness and dissipativity assumptions are standard to the study of the convergence of stochastic gradient MCMC algorithms, and they have been used in many works such as Deng, Zhang, Liang, *et al.* [75] and Raginsky, Rakhlin, and Telgarsky [78].

Assumption 5.3. *Let $\xi_k = \nabla_x \tilde{L}(\boldsymbol{\theta}_k, x_k) - \nabla_x L(\boldsymbol{\theta}_k, x_k)$ denote the white noise contained in the stochastic gradient. The white noises $\xi_1, \xi_2, \ldots$ are mutually independent and satisfy the conditions:*

$$\begin{aligned} &E(\xi_k|\mathcal{F}_k) = 0, \\ &E\|\xi_k\|^2 \leq M^2 E\|x\|^2 + M^2 E\|\boldsymbol{\theta}\|^2 + B^2, \end{aligned} \tag{5.18}$$

where $\mathcal{F}_k = \sigma\{\boldsymbol{\theta}_1, x_1, \boldsymbol{\theta}_2, x_2, \ldots, \boldsymbol{\theta}_k, x_k\}$ denotes a σ-filter.

Assumption (A.3) is a regularity condition for the gradient noise. Similar conditions have been used in the literature, see e.g. Raginsky, Rakhlin, and Telgarsky [78].

Assumption 5.4. *Assume that the trajectory of* $\boldsymbol{\theta}$ *always belongs to a compact set* Θ*, i.e.* $\{\boldsymbol{\theta}_k\}_{k=1}^{\infty} \subset \Theta$ *and* $\|\boldsymbol{\theta}_k\| \leq M$ *for some constant* M*.*

This assumption is made for the simplicity of proof. Otherwise, a varying truncation technique (see e.g. [79] and [80]), can be used in the algorithm for ensuring that $\{\boldsymbol{\theta}_k : k = 1, 2, \ldots\}$ is contained in a compact space.

Assumption 5.5. *For any* $\boldsymbol{\theta} \in \Theta$*, there exists a function* $\mu_{\boldsymbol{\theta}}$ *on* x *which solves the Poisson equation* $\mu_{\boldsymbol{\theta}}(x) - \mathcal{T}_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}(x) = \mathcal{H}(\boldsymbol{\theta}, x) - h(\boldsymbol{\theta})$ *such that*

$$\mathcal{H}(\boldsymbol{\theta}_k, x_{k+1}) = h(\boldsymbol{\theta}_k) + \mu_{\boldsymbol{\theta}_k}(x_{k+1}) - \mathcal{T}_{\boldsymbol{\theta}_k}\mu_{\boldsymbol{\theta}_k}(x_{k+1}), \quad k = 1, 2, \ldots. \tag{5.19}$$

In addition, there exists a constant C *such that* $\|\mathcal{T}_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}\| \leq C$*.*

This assumption has often been used in the study for the convergence of stochastic gradient Langevin dynamics, see e.g. Teh, Thiery, and Vollmer [81] and Deng, Zhang, Liang, *et al.* [75].

Assumption 5.6. $\mathcal{H}(\boldsymbol{\theta}, x)$ *is continuous and there exists a constant* M *such that*

$$\|\mathcal{H}(\boldsymbol{\theta}, x_1) - \mathcal{H}(\boldsymbol{\theta}, x_2)\| \leq M\|x_1 - x_2\|.$$

Theorem 5.1.3 (Convergence of $\boldsymbol{\theta}_k$)**.** *Suppose Assumptions 5.1-5.6 hold;* $\lambda_l \leq \inf_k \lambda_{\min}(V_k) \leq \sup_k \lambda_{\max}(V_k) \leq \lambda_u$ *for some positive constants* λ_l *and* λ_u*, where* $\lambda_{\max}(\cdot)$ *and* $\lambda_{\min}(\cdot)$ *denote the largest and smallest eigenvalues, respectively; and the learning rate sequence* $\{\epsilon_k : k = 1, 2, \ldots\}$ *is sufficiently small such that* $\epsilon_k \leq 2m/(3M^2)$ *for all* $k \geq 1$*, then there exists a constant* ζ *and a root* $\boldsymbol{\theta}^* \in \{\boldsymbol{\theta} : h(\boldsymbol{\theta}) = 0\}$ *such that*

$$E\|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|^2 \leq \zeta w_k,$$

where t *indexes iterations, and* w_k *is the step size satisfying Assumption 5.1.*

Proof. This theorem can be proved as Theorem 1 of Deng, Zhang, Liang, *et al.* [75], but the proofs of Lemma 1, Proposition 2 and Proposition 4 of Deng, Zhang, Liang, *et al.* [75] no

longer hold for the LEnKF algorithm. In what follows we re-establish them under the above assumptions. □

Lemma 5.1 is a replacement of Proposition 2 of Deng, Zhang, Liang, *et al.* [75].

Lemma 5.1. $\|\nabla_x L(\boldsymbol{\theta}, x)\|^2 \leq 3M^2\|x\|^2 + 3M^2\|\boldsymbol{\theta}\|^2 + 3\bar{B}^2$ *for some constant* $\bar{B}$.

Proof. Let $(\boldsymbol{\theta}^*, x^*)$ be the minimizer such that $\nabla_x L(\boldsymbol{\theta}^*, x^*) = 0$ and $\boldsymbol{\theta}^*$ be the stationary point. By the dissipative assumption in Assumption 5.2, $\|x^*\|^2 \leq \frac{b}{m}$.

$$\begin{aligned}\|\nabla_x L(\boldsymbol{\theta}, x)\| &\leq \|\nabla_x L(\boldsymbol{\theta}^*, x^*)\| + M\|x^* - x\| + M\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| \\ &\leq M\|\boldsymbol{\theta}\| + M\|x\| + \bar{B},\end{aligned}$$

where $\bar{B} = M(\sqrt{\frac{b}{m}} + \|\boldsymbol{\theta}^*\|)$. Therefore, $\|\nabla_x L(\boldsymbol{\theta}, x)\|^2 \leq 3M^2\|x\|^2 + 3M^2\|\boldsymbol{\theta}\|^2 + 3\bar{B}^2$ holds. □

Lemma 5.2 is a replacement of Lemma 1 of Deng, Zhang, Liang, *et al.* [75].

Lemma 5.2. *(Uniform L^2 bounds) Suppose Assumptions 5.1-5.6 hold. If the learning rate sequence $\{\epsilon_k : t = 1, 2, \ldots\}$ is sufficiently small such that $\epsilon_k \leq 2m/(3M^2)$ for all $t \geq 1$, then there exists a constant $G > 0$ such that*

$$\sup_k \mathbb{E}\|x_k\|^2 \leq G. \tag{5.20}$$

Proof. By the evolution equation (5.13), we have

$$\begin{aligned}\mathbb{E}\|x_{k+1}\|^2 &= \mathbb{E}\|x_k + \epsilon_{k+1}\nabla_x \tilde{L}(\boldsymbol{\theta}_k, x_k)\|^2 + \mathbb{E}\|\mathrm{e}_{k+1}\|^2 \\ &= \mathbb{E}\|x_k + \epsilon_{k+1}\nabla_x \tilde{L}(\boldsymbol{\theta}_k, x_k)\|^2 + \epsilon_{k+1}\|\Sigma_{k+1}\|.\end{aligned} \tag{5.21}$$

By the gradient noise assumption, the first item of (5.21) can be further expanded as

$$\begin{aligned}&\mathbb{E}\|x_k + \epsilon_{k+1}\nabla_x \tilde{L}(\boldsymbol{\theta}_k, x_k)\|^2 \\ =&\mathbb{E}\|x_k + \epsilon_{k+1}\nabla_x L(\boldsymbol{\theta}_k, x_k)\|^2 + \epsilon_{k+1}^2\mathbb{E}\|\xi_k\|^2 + 2\epsilon_{k+1}\mathbb{E}[\mathbb{E}[\langle x_k + \epsilon_{k+1}\nabla_x L(\boldsymbol{\theta}_k, x_k), \xi_k\rangle|\mathcal{F}_k]] \\ =&\mathbb{E}\|x_k + \epsilon_{k+1}\nabla_x L(\boldsymbol{\theta}_k, x_k\|^2 + \epsilon_{k+1}^2\mathbb{E}\|\xi_k\|^2.\end{aligned} \tag{5.22}$$

By the dissipivatity and boundedness lemmas, the first item of (5.22) can be further expanded as

$$\begin{aligned}
&\mathbb{E}\|x_k + \epsilon_{k+1}\nabla_x L(\boldsymbol{\theta}_k, x_k)\|^2 \\
=&\mathbb{E}\|x_k\|^2 + 2\epsilon_{k+1}\mathbb{E}\langle x_k, \nabla_x L(\boldsymbol{\theta}_k, x_k)\rangle + \epsilon_{k+1}^2\mathbb{E}\|\nabla_x L(\boldsymbol{\theta}_k, x_k)\|^2 \\
\leq&\mathbb{E}\|x_k\|^2 + 2\epsilon_{k+1}(b - m\mathbb{E}\|x_k\|^2) + \epsilon_{k+1}^2(3M^2\mathbb{E}\|x_k\|^2 + 3M^2\|\boldsymbol{\theta}_k\|^2 + 3\bar{B}^2) \\
\leq&(1 - 2\epsilon_{k+1}m + 3\epsilon_{k+1}^2M^2)\mathbb{E}\|x_k\|^2 + 2\epsilon_{k+1}b + 3\epsilon_{k+1}^2\tilde{B}^2,
\end{aligned} \tag{5.23}$$

where $\tilde{B}^2 = M^4 + \max\{\bar{B}^2, B^2\}$.

By the gradient noise assumption, the second item of (5.22) is bounded by

$$\mathbb{E}\|\xi_k\|^2 \leq M^2\mathbb{E}\|x_k\|^2 + M^2E\|\boldsymbol{\theta}_k\|^2 + B^2 < M^2\mathbb{E}\|x_k\|^2 + \tilde{B}^2, \tag{5.24}$$

which leads to

$$\mathbb{E}\|x_{k+1}\|^2 \leq (1 - 2\epsilon_{k+1}m + 3\epsilon_{k+1}^2M^2)\mathbb{E}\|x_k\|^2 + 2\epsilon_{k+1}b + 3\epsilon_{k+1}^2\tilde{B}^2 + \epsilon_{k+1}C_0, \tag{5.25}$$

where $C_0 = \sup_{k\geq 0}\|\Sigma_{k+1}\|$.

Suppose that ϵ_{k+1} is sufficiently small such that $\epsilon_{k+1} < 2m/(3M^2)$ holds for all $t \geq 0$. Then the lemma can be established using an induction method.

The case with $t = 0$ is trivial. Assuming $\mathbb{E}\|x_k\|^2 \leq G$, then $\mathbb{E}\|x_{k+1}\|^2$ can be bounded by

$$\mathbb{E}\|x_{k+1}\|^2 \leq (1 - 2\epsilon_{k+1}m + 3\epsilon_{k+1}^2M^2)G + 2\epsilon_{k+1}b + 3\epsilon_{k+1}^2\tilde{B}^2 + \epsilon_{k+1}C_0.$$

Therefore, if $G \geq \sup_{t\geq 0}\frac{2\epsilon_{k+1}b + 3\epsilon_{k+1}^2\tilde{B}^2 + \epsilon_{k+1}C_0}{2\epsilon_{k+1}m - 3\epsilon_{k+1}^2M^2}$, then $\mathbb{E}\|x_{k+1}\|^2 \leq G$ holds. □

Lemma 5.3 is a replacement of Proposition 4 of Deng, Zhang, Liang, *et al.* [75].

Lemma 5.3. *There exists a constant C_1 such that $E_{\boldsymbol{\theta}_k}\|H(\boldsymbol{\theta}_k, x_{k+1})\|^2 \leq C_1$.*

Proof. By Assumption 5.6,

$$\|\mathcal{H}(\boldsymbol{\theta}_k, x_{k+1})\|^2 \leq 2M\|x_{k+1}\|^2 + 2\|\mathcal{H}(\boldsymbol{\theta}_k, 0)\|^2.$$

Because $\boldsymbol{\theta}_k$ belongs to a compact set, $H(\boldsymbol{\theta}, 0)$ is continuous function and $E\|x_{k+1}\|$ is uniformly bounded, $E_{\boldsymbol{\theta}_k}\|\mathcal{H}(\boldsymbol{\theta}_k, x_{k+1})\|^2$ is uniformly bounded. □

4) Ergodicity of the Parameter-Adaptive LEnKF

To establish the ergodicity of the parameter-adaptive LEnKF, we need more assumptions. Let the fluctuation between ϕ and $\bar{\phi}$:

$$\mathcal{L}f(\boldsymbol{\theta}) = \phi(\boldsymbol{\theta}) - \bar{\phi}, \tag{5.26}$$

where $f(\boldsymbol{\theta})$ is the solution to the Poisson equation, and $\mathcal{L}$ is the infinitesimal generator of the Langevin diffusion. Following Chen, Ding, and Carin [59] and Li, Chen, Carlson, *et al.* [82], we made the following assumption:

Assumption 5.7. *Given a sufficiently smooth function $f(\boldsymbol{\theta})$ as defined in (5.26) and a function $\mathcal{V}(\boldsymbol{\theta})$ such that the derivatives satisfy the inequality $\|D^{\mathrm{j}} f\| \lesssim \mathcal{V}^{p_{\mathrm{j}}}(\boldsymbol{\theta})$ for some constant $p_{\mathrm{j}} > 0$, where $\mathrm{j} \in \{0, 1, 2, 3\}$. In addition, $\mathcal{V}^p$ has a bounded expectation, i.e., $\sup_k E[\mathcal{V}^p(\boldsymbol{\theta}_k)] < \infty$; and $\mathcal{V}^p$ is smooth, i.e. $\sup_{s\in(0,1)} \mathcal{V}^p(s\boldsymbol{\theta} + (1-s)\boldsymbol{\vartheta}) \lesssim \mathcal{V}^p(\boldsymbol{\theta}) + \mathcal{V}^p(\boldsymbol{\vartheta})$ for all $\boldsymbol{\theta}, \boldsymbol{\vartheta} \in \Theta$ and $p \leq 2\max_{\mathrm{j}}\{p_{\mathrm{j}}\}$.*

Theorem 5.1.4 (Ergodicity of the Parameter-Adaptive LEnKF)**.** *Suppose that the conditions of Theorem 5.1.3 and Assumptions 5.7 hold. For a smooth test function ϕ, define*

$$\hat{\phi} = \frac{1}{T}\sum_{k=1}^{T} \phi(x_k), \tag{5.27}$$

where T is the total number of iterations. Let $\bar{\phi} = \int \phi(x)\boldsymbol{\pi}(x|\boldsymbol{\theta}^*, \mathcal{D})dx$. Then

$$E|\hat{\phi} - \bar{\phi}|^2 \to 0, \quad \text{as } T \to \infty, \tag{5.28}$$

provided that $\epsilon_k \to 0$ as $k \to \infty$, and $\sum_{k=1}^{T} \epsilon_k \to \infty$ and $[\sum_{k=1}^{T} \epsilon_k^2]/[\sum_{k=1}^{T} \epsilon_k] \to 0$ as $T \to \infty$.

Proof. The update of the state variable x can be rewritten as

$$x_{k+1} = x_k + \epsilon_{k+1}\Sigma_{k+1}(\nabla_x L(\boldsymbol{\theta}^*, x_k) + \Delta V_k) + \sqrt{2\epsilon_{k+1}\tau}N(0, \Sigma_{k+1}), \tag{5.29}$$

where $\Delta V_k = \nabla_x \tilde{L}(\boldsymbol{\theta}_k, x_k) - \nabla_x L(\boldsymbol{\theta}_k^*, x_k) + \xi_k$ can be viewed as the estimation error for the "true" gradient $\nabla_x L(\boldsymbol{\theta}^*, x_k)$.

Since Σ_k has bounded eigenvalues as analyzed in Section 5.1.3, by Theorem 1 of Li, Chen, Carlson, *et al.* [82], it is sufficient to show that $\sum_{k=1}^T (\epsilon_k^2 E\|\Delta V_k\|^2)/(\sum_{k=1}^T \epsilon_k)^2 \to 0$. Note that

$$\begin{aligned} E\|\Delta V_k\|^2 &\le 2(E\|\nabla_x L(\boldsymbol{\theta}_k, x_k) - \nabla_x L(\boldsymbol{\theta}^*, x_k)\|^2 + E\|\xi_k\|^2) \\ &\le 2(M^2 E\|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|^2 + E\|\xi_k\|^2). \end{aligned}$$

By Theorem 5.1.3, $E\|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|^2 \to 0$; By Assumption 5.3 and Lemma 5.2, $E\|\xi_k\|^2 \le M^2 E\|x\|^2 + M^2 E\|\boldsymbol{\theta}\|^2 + B^2$ is bounded. Hence $E\|\Delta V_k\|^2$ is bounded. Further, by the conditions that $\sum_{k=1}^T \epsilon_k \to \infty$ and $[\sum_{k=1}^T \epsilon_k^2]/[\sum_{k=1}^T \epsilon_k] \to 0$, we can easily conclude that $\sum_{k=1}^T (\epsilon_k^2 E\|\Delta V_k\|^2)/(\sum_{k=1}^T \epsilon_k)^2 \to 0$. □

Remark 5.2. *It is easy to extend Theorem 5.1.3 and Theorem 5.1.4 to the case that* $\nabla_x \tilde{L}(\boldsymbol{\theta}_k, x_k)$ *is biased. In this case, we can assume that* $\nabla_x \tilde{L}(\boldsymbol{\theta}_k, x_k) - \nabla_x L(\boldsymbol{\theta}_k, x_k) = \varsigma_k + \xi_k$, *where* $\varsigma_k = E(\nabla_x \tilde{L}(\boldsymbol{\theta}_k, x_k) - \nabla_x L(\boldsymbol{\theta}_k, x_k)|\mathcal{F}_k) \neq 0$ *is the bias term and* ξ_k *is the random noise term with mean* $E(\xi_k|\mathcal{F}_k) = 0$, *and*

$$E\|\varsigma_k\|^2 \le ME\|x_k\|^2 + M^2 E\|\boldsymbol{\theta}_k\|^2 + B^2.$$

This extension accommodates the bias introduced in the importance resampling step of the LEnKF algorithm for data assimilation problems.

Remark 5.3. *When the LEnKF algorithm is applied to the data assimilation problems, we always assume that the parameters* $\boldsymbol{\theta}$ *are invariant with respect to stage* t. *In this case, Theorem 5.1.3 holds trivially, while Theorem 5.1.4 holds for each stage* t *as the number of iterations becomes large.*

5.2 Langevinized Ensemble Kalman filter with non-Gaussian data

5.2.1 The Extended LEnKF algorithm

In practice, we often encounter dynamic systems where the measurement variable follows a non-Gaussian distribution, e.g., multinomial or Poisson. The LEnKF can be extended to these systems by introducing some Gaussian latent variables. The extension is described, separately, for the inverse problem and data assimilation in what follows.

1) Inverse problems

Consider an inverse problem for which a latent variable model can be formulated as

$$z \sim \psi(\cdot|y), \quad y = Hx + \eta, \quad \eta \sim N(0, \Gamma), \tag{5.30}$$

where z is observed data following a non-Gaussian distribution $\psi(\cdot)$, y is the latent Gaussian variable, and x is the parameter. For this model, we have $\boldsymbol{\pi}(y|x,z) \propto \psi(z|y)\phi(y|x)$, where $\phi(\cdot)$ denotes a Gaussian density function.

Same as in Section 3.1.1, we assume that z can be partitioned into $B = N/n$ independent and identically distributed blocks $\{z_1, \ldots, z_B\}$, where each block is of size n and the corresponding latent variables have the covariance matrix V such that $\Gamma = \text{diag}[V, \cdots, V]$. To adapt the LEnKF to simulating from the posterior distribution $\boldsymbol{\pi}(x|z)$, we only need to add an imputation step in Algorithm 4. The extended algorithm is described in Algorithm 9.

2) Data assimilation problems

For non-Gaussian data assimilation problems, the corresponding state space model (1.3) can be reformulated as follows:

$$\begin{aligned} x_t &= g(x_{t-1}) + u_t, \quad u_t \sim N(0, U_t), \\ y_t &= H_t x_t + \eta_t, \quad \eta_t \sim N(0, \Gamma_t), \\ z_t &\sim \psi(\cdot|y_t), \end{aligned} \tag{5.33}$$

Algorithm 9: Extended LEnKF for non-Gaussian Inverse Problems

(i) Initialization: Initialize an ensemble $\{x_0^{a,1}, x_0^{a,2}, \ldots, x_0^{a,m}\}$, where m is the ensemble size.

for *t=1,2,…, T* **do**

(ii) Subsampling: Draw a mini-batch sample, denoted by (z_t, H_t), of size n from the full data set of size N. Set $Q_t = \epsilon_t I_p$, $R_t = 2V_t$, and the Kalman gain matrix $K_t = Q_t H_t^T (H_t Q_t H_t^T + R_t)^{-1}$.

for *i=1,2,…,m* **do**

(iii) Forecast: Draw $w_t^{\mathrm{i}} \sim N_p(0, \frac{n}{N} Q_t)$ and calculate

$$x_t^{f,\mathrm{i}} = x_{t-1}^{a,\mathrm{i}} + \epsilon_t \frac{n}{2N} \nabla \log \boldsymbol{\pi}(x_{t-1}^{a,\mathrm{i}}) + w_t^{\mathrm{i}}. \tag{5.31}$$

(iv) Imputation: Draw $y_t^{\mathrm{i}} \sim \boldsymbol{\pi}(y|x_t^{f,\mathrm{i}}, z_t) \propto \psi(z_t|y)\phi(y|x_t^{f,\mathrm{i}})$, where $y_t^{\mathrm{i}}|x_t^{f,\mathrm{i}} \sim N(H_t x_t^{f,\mathrm{i}}, V_t)$.

(v) Analysis: Draw $v_t^{\mathrm{i}} \sim N_n(0, \frac{n}{N} R_t)$ and calculate

$$x_t^{a,\mathrm{i}} = x_t^{f,\mathrm{i}} + K_t(y_t^{\mathrm{i}} - H_t x_t^{f,\mathrm{i}} - v_t^{\mathrm{i}}) \stackrel{\Delta}{=} x_t^{f,\mathrm{i}} + K_t(y_t^{\mathrm{i}} - y_t^{f,\mathrm{i}}). \tag{5.32}$$

where H_t is an appropriately chosen matrix, and $y_t \in \mathbb{R}^{N_t}$ represents the latent Gaussian random variable following the conditional distribution $\boldsymbol{\pi}(y_t|x_t, z_t) \propto \psi(z_t|y_t)\phi(y_t|x_t)$, and $\phi(\cdot)$ denotes a Gaussian density function. Then the data assimilation LEnKF algorithm can be extended to the model (5.33) by including an imputation step for the latent variable y_t at each stage t. As in Section 3.2.1, to accommodate the case that N_t is extremely large at each stage t, we assume that z_t can be partitioned into $B = N/n$ independent and identically distributed blocks $\{z_{t,1}, \ldots, z_{t,B}\}$, where each block is of size n and the corresponding latent variable has the covariance matrix V_t such that $\Gamma_t = \mathrm{diag}[V_t, \cdots, V_t]$. The resulting algorithm is described in Algorithm 10.

Algorithm 10: Extended LEnKF for non-Gaussian Data Assimilation Problems

(i) Initialization: Start with an initial ensemble $x_{1,0}^{a,1}, x_{1,0}^{a,2}, \ldots, x_{1,0}^{a,m}$ drawn from the prior distribution $\boldsymbol{\pi}(x_1)$, where m denotes the ensemble size.

for *t=1,2,…,T* **do**

Set $\mathcal{X}_t = \emptyset$ and k_0 as common burn-in period.

for *k=1,2,…,K* **do**

(ii) Subsampling: Draw without replacement a mini-batch sample, denoted by $z_{t,k}$, of size n_t from the full data set $\boldsymbol{z}_t$ of size N_t. Set $Q_{t,k} = \epsilon_{t,k} I_p$, $R_t = 2V_t$, and the Kalman gain matrix $K_{t,k} = Q_{t,k} H_{t,k}^T (H_{t,k} Q_{t,k} H_{t,k}^T + R_t)^{-1}$.

for *i=1,2,…, m* **do**

(iii) Importance resampling: If $t > 1$, calculate importance weights $\omega_{t,k-1,\mathrm{j}}^{\mathrm{i}} = \boldsymbol{\pi}(x_{t,k-1}^{a,\mathrm{i}} | x_{t-1,\mathrm{j}}) = \phi(x_{t,k-1}^{a,\mathrm{i}} : g(x_{t-1,\mathrm{j}}), U_t)$ for $\mathrm{j} = 1, 2, \ldots, |\mathcal{X}_{t-1}|$, where $\phi(\cdot)$ denotes a Gaussian density, and $x_{t-1,\mathrm{j}} \in \mathcal{X}_{t-1}$ denotes the jth sample in $\mathcal{X}_{t-1}$; if $k = 1$, set $x_{t,0}^{a,\mathrm{i}} = g(x_{t-1,\mathcal{K}}^{a,\mathrm{i}}) + u_t^{a,\mathrm{i}}$ and $u_t^{a,\mathrm{i}} \sim N(0, U_t)$. Resample $s \in \{1, 2, \ldots, |\mathcal{X}_{t-1}|\}$ with a probability $\propto \omega_{t,k-1,s}^{\mathrm{i}}$, i.e., $P(S_{t,k,\mathrm{i}} = s) = \omega_{t,k-1,s}^{\mathrm{i}} / \sum_{\mathrm{j}=1}^{|\mathcal{X}_{t-1}|} \omega_{t,k-1,\mathrm{j}}^{\mathrm{i}}$, and denote the sample drawn from $\mathcal{X}_{t-1}$ by $\tilde{x}_{t-1,k-1}^{\mathrm{i}}$.

(iv) Forecast: Draw $w_{t,k}^{\mathrm{i}} \sim N_p(0, \frac{n_t}{N_t} Q_{t,k})$. If $t = 1$, set

$$x_{t,k}^{f,\mathrm{i}} = x_{t,k-1}^{a,\mathrm{i}} - \epsilon_{t,k} \frac{n_t}{2N_t} \nabla \log \boldsymbol{\pi}(x_{t,k-1}^{a,\mathrm{i}}) + w_{t,k}^{\mathrm{i}}, \tag{5.34}$$

where $\boldsymbol{\pi}(\cdot)$ denotes the prior distribution of x_1. If $t > 1$, set

$$x_{t,k}^{f,\mathrm{i}} = x_{t,k-1}^{a,\mathrm{i}} - \epsilon_{t,k} \frac{n_t}{2N_t} U_t^{-1} (x_{t,k-1}^{a,\mathrm{i}} - g(\tilde{x}_{t-1,k-1}^{\mathrm{i}})) + w_{t,k}^{\mathrm{i}}. \tag{5.35}$$

(v) Imputation: Draw $y_{t,k}^{\mathrm{i}} \sim \boldsymbol{\pi}(y | x_{t,k}^{f,\mathrm{i}}, z_{t,k}) \propto \phi(y | x_{t,k}^{f,\mathrm{i}}) \psi(y | z_{t,k})$, where $z_{t,k}$ is an n-vector representing a mini-batch of data and $\phi(\cdot)$ is a Gaussian density function.

(vi) Analysis: Draw $v_{t,k}^{\mathrm{i}} \sim N_n(0, \frac{n_t}{N_t} R_t)$ and set

$$x_{t,k}^{a,\mathrm{i}} = x_{t,k}^{f,\mathrm{i}} + K_{t,k}(y_{t,k}^{\mathrm{i}} - H_{t,k} x_{t,k}^{f,\mathrm{i}} - v_{t,k}^{\mathrm{i}}) \triangleq x_{t,k}^{f,\mathrm{i}} + K_{t,k}(y_{t,k}^{\mathrm{i}} - y_{t,k}^{f,\mathrm{i}}). \tag{5.36}$$

(vii) Sample collection: If $k > k_0$, add the sample $x_{t,k}^{a,\mathrm{i}}$ into the set $\mathcal{X}_t$.

5.2.2 Convergence analysis

To justify the convergence of the extended LEnKF algorithms, we consider the identity (3.15) again, which is introduced in Section 3.2.1. With this identity, it is easy to show that

$$\nabla_x \log \boldsymbol{\pi}(x|z_t) = \int \nabla_x \log \boldsymbol{\pi}(x|y, z_t)\boldsymbol{\pi}(y|x, z_t)dy = \int \nabla_x \log \boldsymbol{\pi}(x|y)\boldsymbol{\pi}(y|x, z_t)dy, \tag{5.37}$$

where the last equality holds as $\boldsymbol{\pi}(x|y, z) = \boldsymbol{\pi}(x|y)$ given the hierarchical structure (5.30). That is, $\nabla_x \log \boldsymbol{\pi}(x|y)$ forms an unbiased estimator of $\nabla_x \log \boldsymbol{\pi}(x|z_t)$, provided that $y \sim \boldsymbol{\pi}(y|x, z_t)$. Further, by (3.5), we can show that Algorithm 9 leads to a SGRLD algorithm for simulating from the posterior distribution $\boldsymbol{\pi}(x|z)$ even when z is not Gaussian. In summary, we have the following theorem concerning the convergence of Algorithm 9.

Theorem 5.2.1. *(Convergence of extended LEnKF for non-Gaussian inverse problems) Let x_t^a denote a generic member of the ensemble produced by Algorithm 9 in the analysis step of stage t. If the eigenvalues of V_t are uniformly bounded with respect to t, $\log \boldsymbol{\pi}(x)$ is differentiable with respect to x, and the learning rate $\epsilon_t = O(t^{-\varpi})$ for some $0 < \varpi < 1$, then $\lim_{t\to\infty} W_2(\tilde{\boldsymbol{\pi}}_t, \boldsymbol{\pi}_*) = 0$, where $\tilde{\boldsymbol{\pi}}_t$ denotes the empirical distribution of x_t^a, $\boldsymbol{\pi}_* = \boldsymbol{\pi}(x|y)$ denotes the target posterior distribution, and $W_2(\cdot, \cdot)$ denotes the second-order Wasserstein distance.*

The proof of Theorem 5.2.1 directly follows from (5.37) and the proof of Theorem 3.3.1, and thus is omitted. When an exact sampler for $\boldsymbol{\pi}(y|x, z_t)$ is not available, the imputation step can be done by a short run of the Metropolis-Hastings algorithm. In this case, (5.32) can be replaced by

$$x_t^{a,\mathrm{i}} = x_t^{f,\mathrm{i}} + K_t\Big(\bar{y}_t^{\mathrm{i}} - H_t x_t^{f,\mathrm{i}} - v_t^{\mathrm{i}}\Big), \tag{5.38}$$

where $\bar{y}_t^{\mathrm{i}} = \frac{1}{r}\sum_{\mathrm{j}=1}^{r} y_t^{\mathrm{i,j}}$ and $y_t^{\mathrm{i},1}, \ldots, y_t^{\mathrm{i},r}$ are the samples simulated in a short Metropolis-Hastings run. The validity of the resulting algorithm can be justified by noting that $\boldsymbol{\pi}(y|x)$ is Gaussian, which implies that $\nabla_x \log \boldsymbol{\pi}(x|y)$ is a linear function of y. As a result, (5.38) leads to an asymptotically unbiased estimator of $\nabla_x \log \boldsymbol{\pi}(x|z)$ given by $N/nH_t^T V_t^{-1}(\bar{y}_t -$

$H_t x_{t-1}^a) + \nabla_x \log \boldsymbol{\pi}(x_{t-1}^a)$, which is similar to the one given in (3.5). When r is small, this estimator can be biased as the Metropolis-Hastings run might have not yet reached its equilibrium. By the standard theory of MCMC, it is easy to figure out that the bias is of the order $O(1/r)$. Further, by Corollary 1 in Section 3.3, this bias will not affect much on the validity of the algorithm as long as r is reasonably large. To be more precise, we have $\limsup_{t\to\infty} W_2(\tilde{\boldsymbol{\pi}}_t, \boldsymbol{\pi}_*) = O(1/r)$ in this case. Similar results can be found in Song, Sun, Ye, *et al.* [58], Dalalyan and Karagulyan [63] and Bhatia, Ma, Dragan, *et al.* [83], where the convergence of the stochastic gradient Langevin dynamic (SGLD) algorithm was established with an inaccurate gradient. In general, the averaging estimator reduces the variation of the stochastic gradient and improves the convergence of the SGMCMC algorithm, see Nemeth and Fearnhead [8] for more discussions on this issue. Alternative to the averaging estimator, the last sample generated by the short short of the Metropolis-Hastings algorithm can also be used for simplicity. The validity of the resulting algorithm can be justified similarly.

Concerning the convergence of Algorithm 10, we have the following theorem, whose proof directly follows from (5.37) and the proof of Theorem 3.3.2, and is thus omitted.

Theorem 5.2.2. *(Convergence of extended LEnKF for non-Gaussian State Space Models) Assume that for each stage t, the matrices H_t, U_t and V_t, the state propagator $g(x_t)$, and the iteration number $\mathcal{K}$ satisfy the regularity conditions given in Theorem 3.3.2. If all N_t's are bounded away from 0, then $\limsup_{t\to\infty} W_2(\tilde{\boldsymbol{\pi}}_t, \boldsymbol{\pi}_t) = O(1/\liminf_t N_t)$, where $\boldsymbol{\pi}_t$ denotes the filtering distribution at stage t, and $\tilde{\boldsymbol{\pi}}_t$ denotes an empirical estimate of the filtering distribution $\boldsymbol{\pi}_t$.*

6. EMPIRICAL RESULTS OF THE SA-LENKF AND THE EXTENDED LENKF

In this chapter, we present multiple examples for both SA-LEnKF algorithm and the Extended LEnKF algorithm. The numerical results indicate that it is not only able to produce a better accurate point estimate as the existing methods do, but also able to quantify uncertainty of the resulting estimate.

6.1 Empirical results of the SA-LEnKF algorithm

This section includes three simulation studies, which are for stochastic parameters, multiplicative parameters and multi-parameters, and one real data analysis using LSTM networks.

6.1.1 Dynamic linear model with stochastic parameters

Consider a dynamic linear model given by

$$\begin{aligned} \mathbf{x}_t|\mathbf{x}_{t-1} &\sim \mathcal{N}_p(\mathbf{M}(\boldsymbol{\alpha})\mathbf{x}_{t-1}, \sigma^2\mathbf{I}_p), \\ \mathbf{y}_t|\mathbf{x}_t &\sim \mathcal{N}_{d_t}(\mathbf{H}_t\mathbf{x}_t, 0.1^2\mathbf{I}_{d_t}), \end{aligned} \tag{6.1}$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3) = (0.3, 0.3, 0.3)$, $p = 60$, $d_t = 54$ for all $t = 1, 2, \ldots, T$, $\mathbf{H}_t$ is a random selection matrix of size $d_t \times p$, $\mathbf{M}(\boldsymbol{\alpha})$ is a tri-diagonal matrix with α_1 on the main diagonal, and α_2 and α_3 on the first upper and lower sub-diagonals, respectively. In the simulation, we set $T = 100$ and $\sigma = 0.2$, and drew $\mathbf{x}_0 \sim \mathcal{N}_n(\boldsymbol{\mu_0}, \boldsymbol{\Sigma}_0)$, where $\boldsymbol{\mu}_0 = -2\mathbf{1}_p$, $\boldsymbol{\Sigma}_0 = \mathbf{I}_p$, and $\mathbf{1}_p$ denotes a constant vector of ones. Our goal in this example is to simultaneously estimate the states $x_1, x_2, \ldots, x_T$ and the stochastic parameter $\boldsymbol{\theta} = \{\sigma\}$.

The SA-LEnKF-Online (Algorithm 7), SA-LEnKF-Offline (Algorithm 8), and state-augmented EnKF (Algorithm 2) were applied to this example. For SA-LEnKF-Online, we set the ensemble size $m = 20$, stage iteration number $\mathcal{K} = 20$, the state learning rate $\epsilon_{t,k} = 0.01/k^{0.6}$ for $k = 1, 2, \ldots, \mathcal{K}$ and $t = 1, 2, \ldots, 100$, and the parameter learning rate $\gamma_{t,k} = 0.0002/(tk)^{0.6}$ for $k = 1, 2, \ldots, \mathcal{K}$ and $t = 1, 2, \ldots, T$. At each stage t, the state was estimated by averaging over the ensembles generated during the last $\mathcal{K} - k_0$ iterations,

and the resulting estimate was denoted by $\hat{x}_t$. The accuracy of $\hat{x}_t$ is measured by the root mean-squared error $RMSE_t(x) = \|\hat{x}_t - x_t\|_2/\sqrt{p}$. Similarly, the parameter $\boldsymbol{\theta}$ was estimated by averaging over those obtained during the last $\mathcal{K} - k_0$ iterations and denoted by $\hat{\boldsymbol{\theta}}_t$, whose accuracy was measured by $RMSE_t(\boldsymbol{\theta}) = \|\hat{\boldsymbol{\theta}}_t - \boldsymbol{\theta}\|_2/\sqrt{d_{\boldsymbol{\theta}}}$, where $d_{\boldsymbol{\theta}}$ denotes the dimension of $\boldsymbol{\theta}$. The final estimate of $\boldsymbol{\theta}$ was given by $\hat{\boldsymbol{\theta}} = 2/T \sum_{t=T/2+1}^{T} \hat{\boldsymbol{\theta}}_t$.

For the SA-LEnKF-Offline algorithm, we set the ensemble size $m = 20$, the sweep number $L = 100$, the stage iteration number $\mathcal{K} = 20$, the state learning rate $\epsilon_{l,t,k} = 0.01/k^{0.6}$ for $k = 1, 2, \ldots, \mathcal{K}$ and $l, t = 1, 2, \ldots, 100$, and the parameter learning rate $\gamma_l = 0.001/l^{0.6}$ for $l = 1, 2, \ldots, L$. At each sweep l and each stage t, the state was estimated by averaging over the ensembles generated during the last $\mathcal{K} - k_0$ iterations, and the resulting estimate was denoted by $\hat{x}_{l,t}$. In default, the last sweep estimate $\hat{x}_{L,t}$ was used as the final estimate of x_t. The accuracy of $\hat{x}_{L,t}$ is measured by $RMSE_t(x) = \|\hat{x}_{L,t} - x_t\|_2/\sqrt{p}$. At each sweep, the resulting parameter estimate is denoted by $\hat{\boldsymbol{\theta}}_l$, whose accuracy was measured by $RMSE_l(\boldsymbol{\theta}) = \|\hat{\boldsymbol{\theta}}_l - \boldsymbol{\theta}\|_2/\sqrt{d_{\boldsymbol{\theta}}}$. The final estimate of $\boldsymbol{\theta}$ is given by $\tilde{\boldsymbol{\theta}} = 2/L \sum_{l=L/2+1}^{L} \hat{\boldsymbol{\theta}}_l$. Since for each sweep, the offline algorithm had the same settings of m and $\mathcal{K}$ as the online algorithm, the offline algorithm is expected to be about L times slower than the online algorithm. This setting is certainly not optimal. For many problems, SA-LEnKF-Offline can be significantly accelerated by reducing the stage iteration number or the number of sweeps.

For the state-augmented EnKF, we set the ensemble size $m = 20$ and $\sigma_{\boldsymbol{\theta}} = 0.01$, which makes the parameter estimate converge fast and most accurate for this example. At each stage, the state and parameter were estimated by averaging over the ensemble. The final estimate of $\boldsymbol{\theta}$ was obtained by averaging over the estimates produced in the stages $T/2 + 1, T/2 + 2, \ldots, T$. The above state and parameter estimation procedures prescribed for the three algorithms have been used in all examples of this dissertation.

Figure 6.1 compares the above three algorithms in state estimation for one dataset simulated from (6.1). It shows the estimates of $x_t^{(k)}$ and their 95% confidence bands along with stage t for $k = 10$, 25 and 40, where $x_t^{(k)}$ denotes the kth component of x_t, and the confidence bands were calculated based on the asymptotic normality of the state estimates. Both SA-LEnKF-Online and SA-LEnKF-Offline worked very well for this example, providing good point estimates as well as uncertainty quantification for the estimates. Unfortunately, the

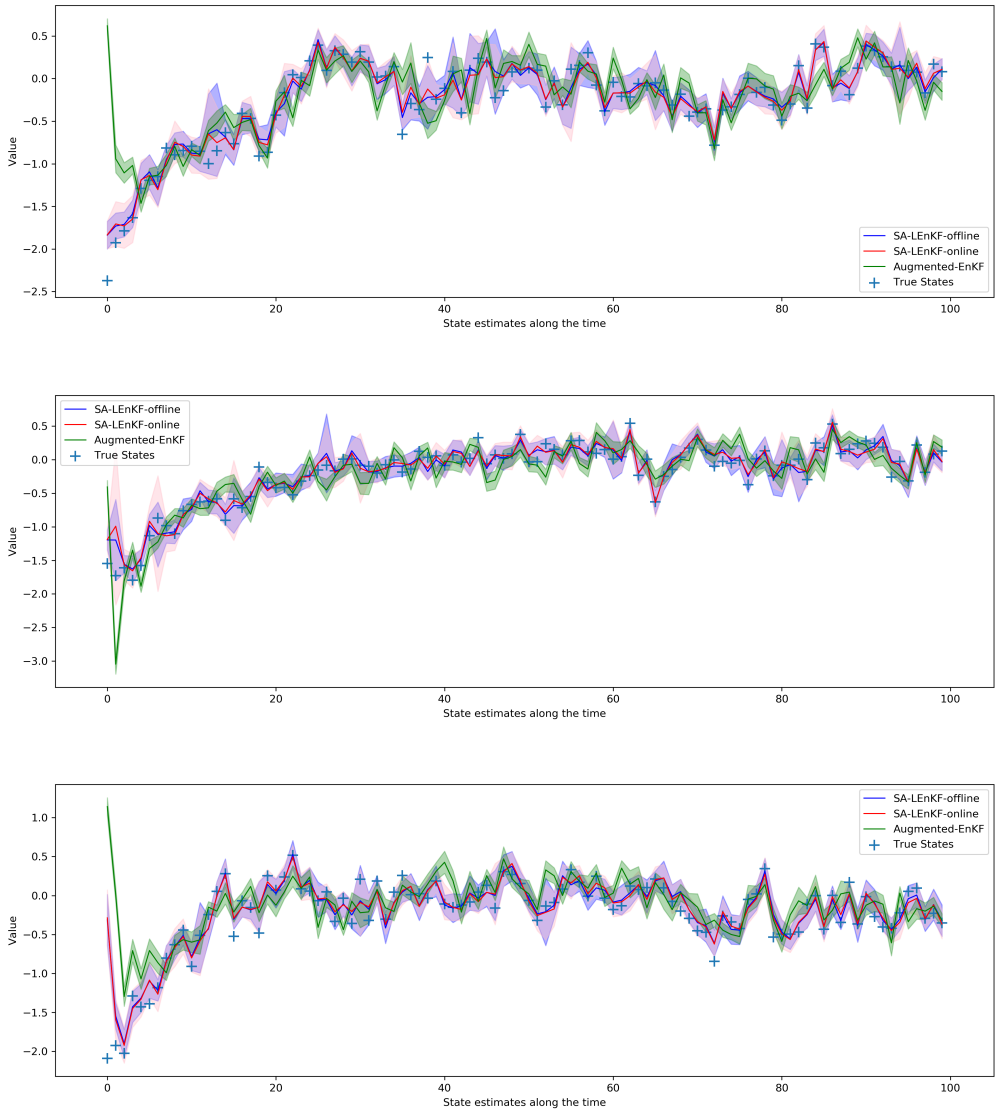
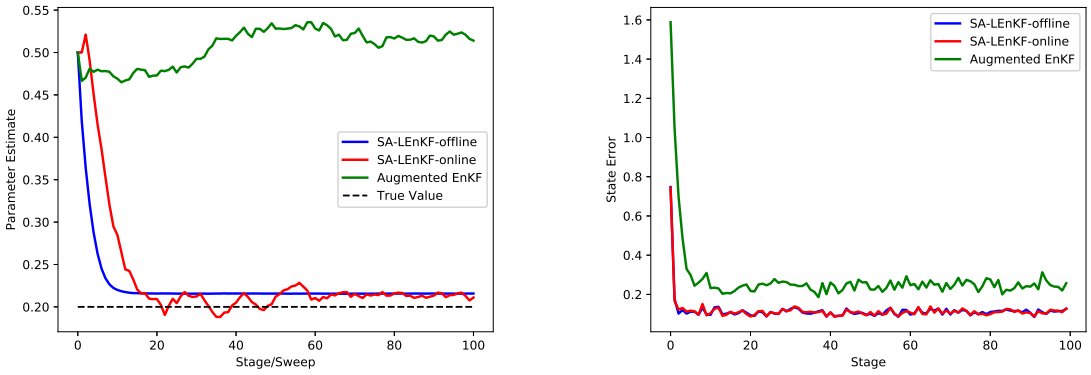


Figure 6.1. Comparison of SA-LEnKF-Online, SA-LEnKF-Offline and state-augmented EnKF in state estimation for the model (6.1): the plots in the upper, middle and lower panels show, respectively, the estimates of $x_t^{(10)}$, $x_t^{(25)}$ and $x_t^{(40)}$, where the true state values are represented by '+', the estimates are represented by solid lines, and the 95% confidence bands are represented by the shaded area.

state-augmented EnKF performed less satisfactorily for this example, which failed to follow the underlying true pattern of the observation in the early stages.

Figure 6.2 summarizes the performance of the three algorithms in parameter and state estimation, which indicate that both the SA-LEnKF-Online and SA-LEnKF-Offline algorithms



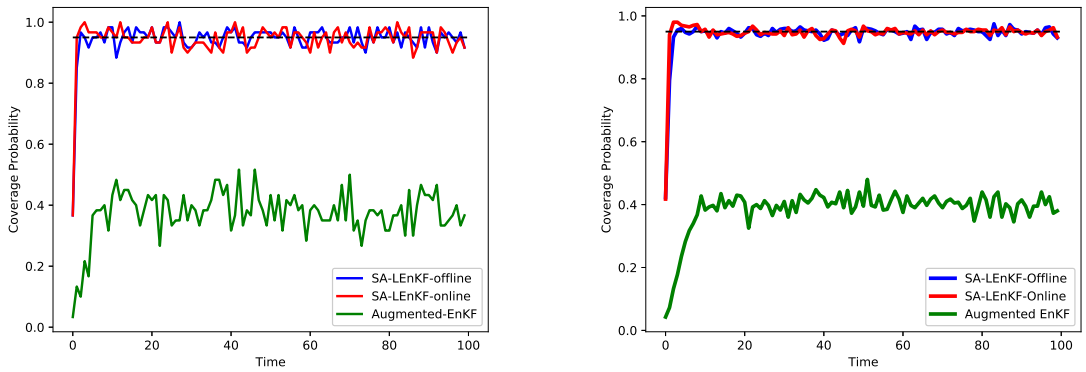
(a) Estimates of σ (b) $\text{RMSE}_t(x)$

Figure 6.2. Comparison of SA-LEnKF-Online, SA-LEnKF-Offline and state-augmented EnKF in parameter and state estimation for the model (6.1): plot (a) shows the estimate of σ, where x-axis represents the sweep number for SA-LEnKF-Offline and the stage number for SA-LEnKF-Online and state-augmented EnKF; and plot (b) shows $\text{RMSE}_t(x)$ of the state estimates along with stage t.

significantly outperform the state-augmented EnKF algorithm. The SA-LEnKF-Online and SA-LEnKF-Offline performed similarly for this example; they converged to the same parameter value and have almost identical $RMSE_t(x)$ at all stages. As mentioned previously, the stochastic parameter is usually hard to be estimated. It is encouraging that both the SA-LEnKF-Online and SA-LEnKF-Offline algorithms produced estimates very close to the true parameter value.

Figure 6.3 compares the coverage probabilities of the confidence intervals produced by the three algorithms for the states $x_1, x_2, \ldots, x_T$. Figure 6.3(a) shows the results for one dataset, where, for each stage t, the coverage probability was calculated by averaging over the coverage status of each component of x_t. Figure 6.3(b) shows the coverage probabilities averaged over 10 datasets. The coverage probabilities produced by the two SA-EnKF algorithms are very close to the nominal level 95%, while those by state-augmented EnKF are only about 40%.

Table 6.1 summarizes the numerical results of the three algorithms, where $k_0 = \mathcal{K} - 1$ was set for the two SA-LEnKF algorithms, MRMSE(x), MCP and MRMSE($\boldsymbol{\theta}$) denote the results for one dataset; and Ave-MRMSE(x), Ave-MCP and Ave-MRMSE($\boldsymbol{\theta}$) denote the results



(a) Coverage probabilities with one dataset (b) Coverage probabilities with ten datasets

Figure 6.3. Coverage probabilities of the 95% confidence intervals produced by the SA-LEnKF-Online, SA-LEnKF-Offline and state-augmented EnKF algorithms for the states $x_1, x_2, \ldots, x_T$ of the model (6.1), where plots (a) and (b) show the results for one and ten datasets, respectively. In the estimation, we set $k_0 = \mathcal{K} - 1$ to reduce the effect of pre-converged parameter estimates.

averaged over 10 datasets. For the online method, MRMSE(x) denotes the averaged root-mean-squared-error of the state estimate, MCP denotes the averaged coverage probability of the state, and MRMSE($\boldsymbol{\theta}$) denotes the averaged root-mean-squared-error of the estimate of $\boldsymbol{\theta}$, where the average was taken over the stages $t = 51, 52, \ldots, T$. For the offline method, they are defined in the same way, but only the estimates obtained in the last sweep was used. The comparison shows that for this example, SA-LEnKF-Online and SA-LEnKF-Offline produced almost the same state and parameter estimates, while the latter is much more costly in CPU time. The state-augmented EnKF is poor in both state and parameter estimation, although it is very fast.

6.1.2 Dynamic nonlinear model with multiplicative parameters

Consider the Lorenz-96 model [33] introduced in Section 4.2.1 again. Let $\mathcal{L}(\cdot)$ denote the state evolution equation obtained by the Runge-Kutta numerical method. At each stage t and for each component i, we added to $x_t^{(\mathrm{i})}$ a white noise, i.e., standard Gaussian random

Table 6.1. Comparison of SA-EnKF-Online, SA-LEnKF-Offline and state-augmented EnKF in state and parameter estimation for the model (6.1), where the average values over 10 independent datasets were reported with the standard deviation given in the parentheses. The CPU time (in seconds) was recorded for a single run of each algorithm.

	Ave-MRMSE(x)	Ave-MCP	Ave-MRMSE($\boldsymbol{\theta}$)	CPU Time
SA-LEnKF-Offline	0.1112(0.0006)	0.9499(0.0016)	0.0166(0.0008)	101.200(3.909)
SA-LEnKF-Online	0.1108(0.0007)	0.9496(0.0024)	0.0131(0.0006)	1.151(0.099)
Augmented EnKF	0.2312(0.0013)	0.4027(0.0048)	0.2953(0.0140)	0.176(0.009)

error. At each stage t, there are only $N_t = p/2$ randomly chosen state values observable, which are masked with a white noise. We re-write the model as

$$\begin{aligned} \boldsymbol{y}_t|\boldsymbol{X}_t &\sim \mathbb{N}(H_t\boldsymbol{X}_t, \boldsymbol{I}_n) \\ \boldsymbol{X}_t|\boldsymbol{X}_{t-1}, \boldsymbol{\theta} &\sim \mathbb{N}(g(\boldsymbol{X}_{t-1}; \boldsymbol{\theta}), \sigma^2\boldsymbol{I}_n), \end{aligned} \tag{6.2}$$

where $g(\boldsymbol{X}_{t-1}; \boldsymbol{\theta}) = \boldsymbol{\theta}\mathcal{L}(\boldsymbol{X}_{t-1})$ is the nonlinear state evolution operator, H_t is a random selection matrix, $T = 100$, $\sigma = 1$ is known, and $\boldsymbol{\theta}$ is an unknown parameter. Our goal for this example is to simultaneously estimate the states and the parameter $\boldsymbol{\theta}$.

The SA-LEnKF-Online (Algorithm 7), SA-LEnKF-Offline (Algorithm 8), and state-augmented EnKF (Algorithm 2) were applied to this example. In SA-LEnKF-Online, we set the ensemble size $m = 50$, the stage iteration number $\mathcal{K} = 20$, the state learning rate $\epsilon_{t,k} = 0.5/k^{0.8}$ for $k = 1, 2, \ldots, \mathcal{K}$ and $t = 1, 2, \ldots, 100$, and the parameter learning rate $\gamma_{t,k} = 0.0002/(tk)^{0.9}$ for $t = 1, 2, \ldots, L$ and $k = 1, 2, \ldots, \mathcal{K}$. In SA-LEnKF-Offline, we set the ensemble size $m = 50$, the number of seeps $L = 100$, the stage iteration number $\mathcal{K} = 20$, the state learning rate $\epsilon_{l,t,k} = 0.5/k^{0.8}$ for $k = 1, 2, \ldots, \mathcal{K}$ and $l, t = 1, 2, \ldots, 100$, and the parameter learning rate $\gamma_l = 0.001/l^{0.9}$ for $l = 1, 2, \ldots, L$. In state-augmented EnKF, we set the ensemble size $m = 50$ and $\sigma_{\boldsymbol{\theta}} = 0.01$.

Figure 6.4 compares the three algorithms in state estimation for one dataset simulated above. It indicates that the two SA-LEnKF algorithms worked very well in both state estimation and uncertainty quantification for the estimates. In contrast, the state-augmented

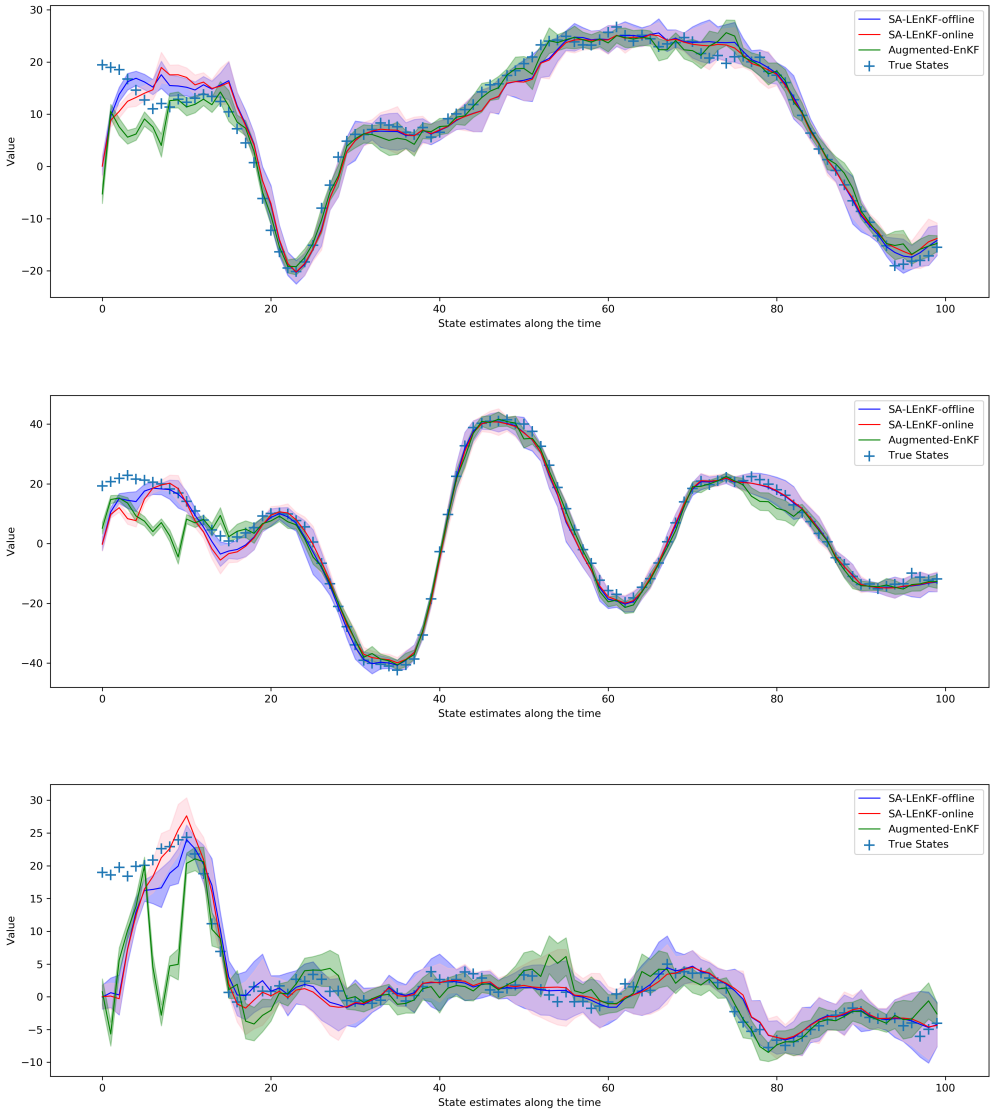
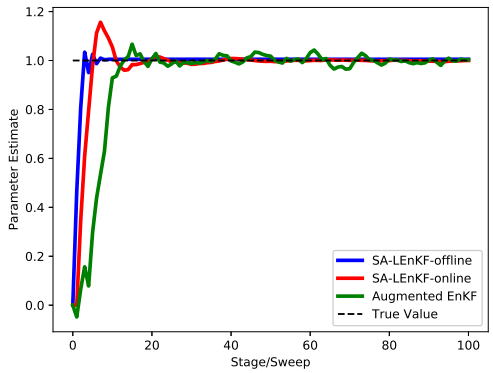
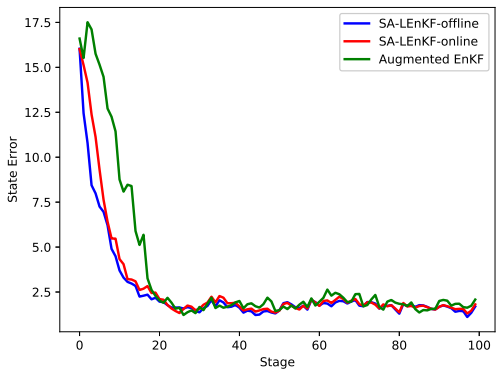


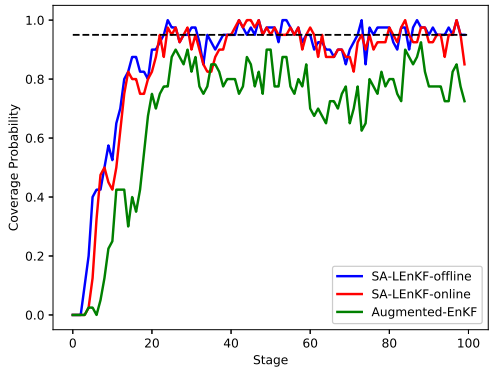
Figure 6.4. Comparison of SA-LEnKF-Online, SA-LEnKF-Offline and state-augmented EnKF in state estimation for the Lorenz-96 model: The upper, middle and lower panels show, respectively, the estimates of $x_t^{(1)}$, $x_t^{(15)}$ and $x_t^{(20)}$ along with stage t, where the true state values are represented by '+', the estimates are represented by solid lines, and the 95% confidence bands are represented by shaded areas.



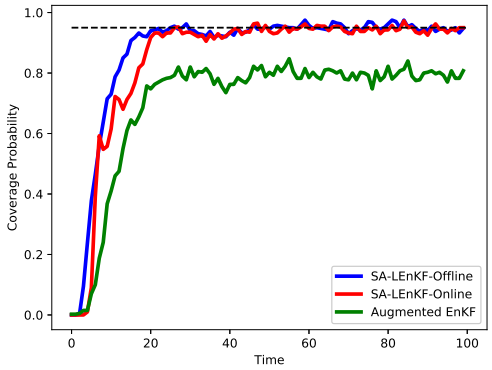
(a) Estimates of $\boldsymbol{\theta}$



(b) $\text{RMSE}_t(x)$



(c) Coverage Probabilities with one dataset



(d) Coverage probabilities with ten datasets

Figure 6.5. Comparison of SA-LEnKF-Online, SA-LEnKF-Offline and state-augmented EnKF in state and parameter estimation for the Lorenz-96 model: (a) estimates of $\boldsymbol{\theta}$; (b) $\text{RMSE}_t(x)$ of the state estimates along with stage t; (c) state coverage probabilities along with stage t, which were calculated based on one dataset; (d) state coverage probabilities along with stage t, which were averaged over 10 datasets.

EnKF performed less satisfactorily, which needs more stages to capture the pattern of the observations than the SA-LEnKF algorithms.

Figure 6.5(a) shows that the two SA-LEnKF algorithms produced very accurate estimates of $\boldsymbol{\theta}$, while the estimates by the state-augmented EnKF have relatively large variations. Figure 6.5(b) shows that the three algorithms produced about the same accurate state estimates, but the state-augmented EnKF converged slightly slowly. Figure 6.5(c) shows the state coverage probabilities calculated based on one dataset, where the coverage probability at each

stage was calculated by averaging the coverage status of all components of the state variable. Figure 6.5(d) shows the state coverage probabilities calculated based on 10 datasets, which were obtained by further averaging the coverage probabilities obtained in plot (c) over 10 datasets. The state estimation results by the two SA-LEnKF algorithms are very impressive, for which the coverage probabilities are almost identical to the nominal level 95%. In contrast, the coverage probabilities produced by the state-augmented EnKF algorithm are much lower, which are about 80% over all stages.

Table 6.2. Comparison of SA-EnKF-Offline, SA-LEnKF-Online and state-augmented EnKF in state and parameter estimation for the Lorenz-96 model, where the average values over 10 independent datasets were reported with the standard deviation given in the parentheses. The CPU time (in seconds) was recorded for a single run of each algorithm. Refer to Section 6.1.1 for notations of the table.

	Ave-MRMSE(x)	Ave-MCP	Ave-MRMSE ($\boldsymbol{\theta}$)	CPU Time
SA-LEnKF-Offline	1.605(0.0183)	0.9569(0.0027)	0.0046(0.0003)	110.93(0.7726)
SA-LEnKF-Online	1.612(0.0256)	0.9500(0.0032)	0.0038(0.0005)	1.235(0.1578)
Augmented EnKF	1.757(0.0231)	0.7974(0.0042)	0.0142(0.0006)	0.143(0.004)

Table 6.2 summarizes the numerical results of the three algorithms, where $k_0 = \mathcal{K} - 1$ was set for the two SA-LEnKF algorithms. In summary, the two SA-LEnKF algorithms worked extremely well for the Lorenz-96 model, while the state-augmented EnKF worked less satisfactorily. In terms of CPU time, the SA-LEnKF-Offline is much more costly than the other two algorithms.

6.1.3 Dynamic linear model with multiple unknown parameters

Consider the dynamic linear model (6.1) again, which has the same settings as in Section 6.1.1 except that $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3)$ is treated as unknown parameters. For this example, our goal is to simultaneously estimate the states $x_1, x_2, \ldots, x_T$ and the parameters $\boldsymbol{\theta} = (\alpha_1, \alpha_2, \alpha_3, \sigma)$.

The SA-LEnKF-Online (Algorithm 7), SA-LEnKF-Offline (Algorithm 8), and state-augmented EnKF (Algorithm 2) were applied to this example. For the SA-LEnKF-Online

algorithm, we set the ensemble size $m = 20$, the stage iteration number $\mathcal{K} = 20$, $k_0 = \mathcal{K} - 1$, the state learning rate $\epsilon_{t,k} = 0.01/k^{0.6}$ for $k = 1, 2, \ldots, \mathcal{K}$ and $t = 1, 2, \ldots, 100$, and the parameter learning rate $\gamma_{t,k} = 0.0002/(tk)^{0.6}$ for $k = 1, 2, \ldots, \mathcal{K}$ and $t = 1, 2, \ldots, T$. For the SA-LEnKF-Offline algorithm, we set the ensemble size $m = 20$, the stage iteration number $\mathcal{K} = 20$, $k_0 = \mathcal{K} - 1$, the sweep number $L = 100$, the state learning rate $\epsilon_{l,t,k} = 0.01/k^{0.6}$ for $k = 1, 2, \ldots, \mathcal{K}$ and $l, t = 1, 2, \ldots, 100$, and the parameter learning rate $\gamma_l = 0.002/l^{0.6}$ for $l = 1, 2, \ldots, L$. For the state-augmented EnKF algorithm, we set the ensemble size $m = 20$ and $\sigma_{\boldsymbol{\theta}} = 0.01$.

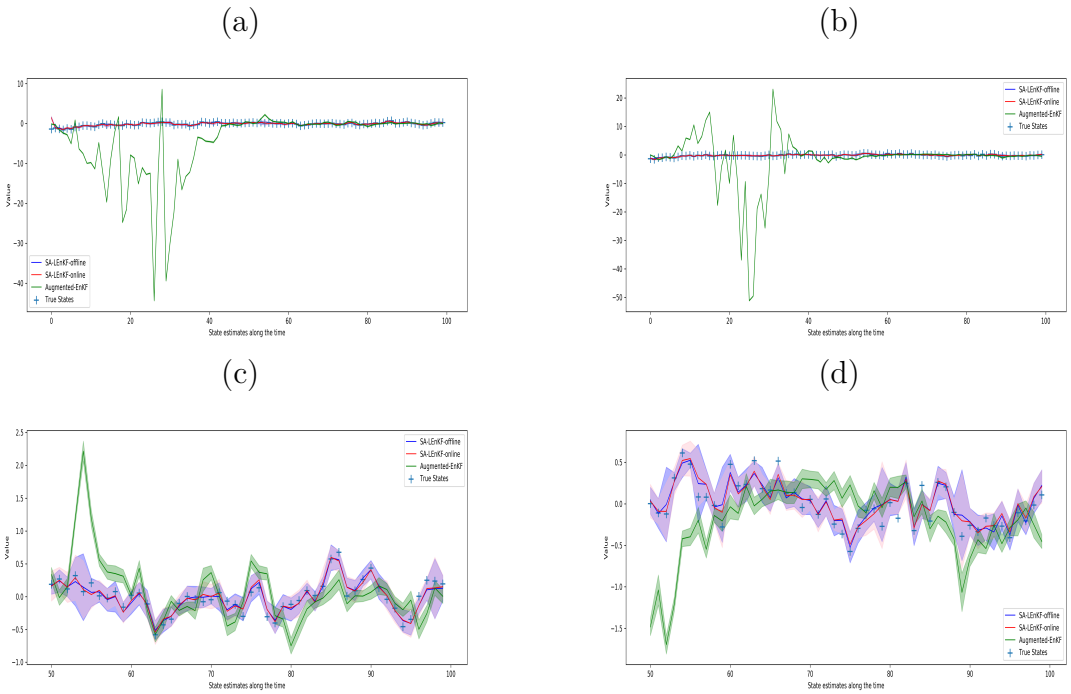
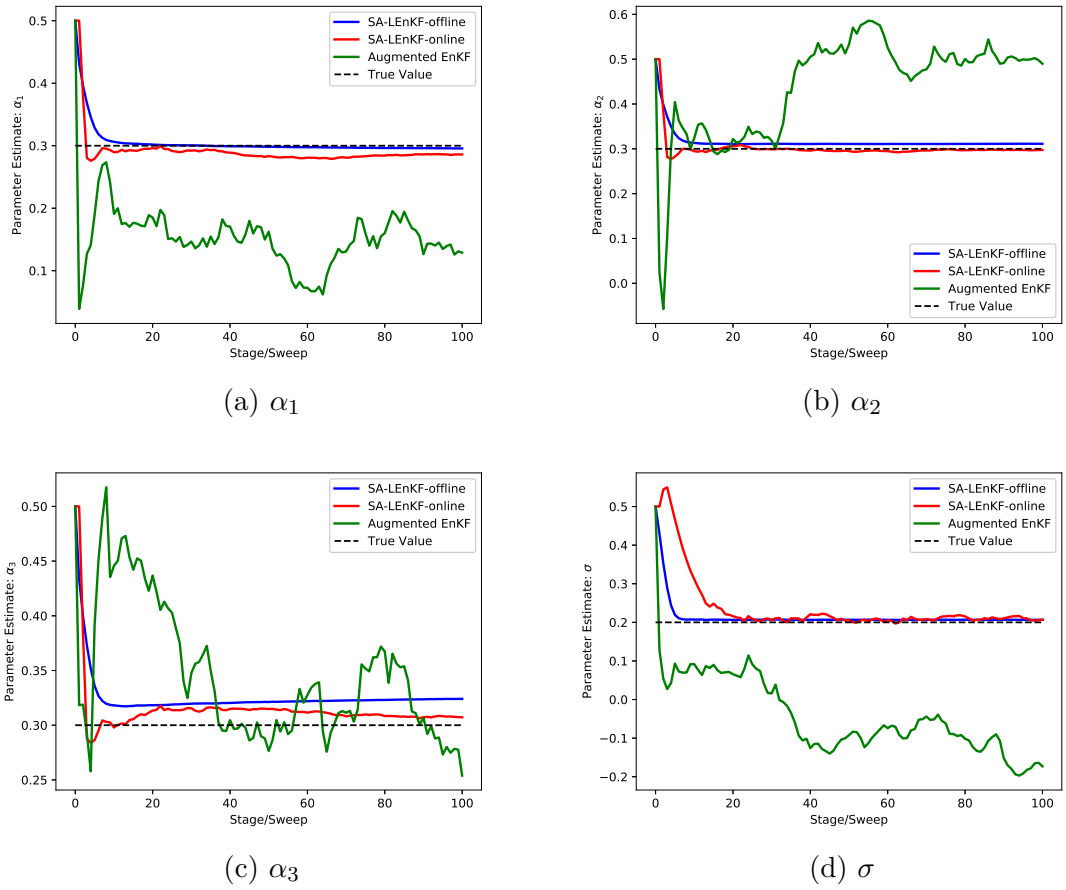


Figure 6.6. Comparison of SA-LEnKF-Online, SA-LEnKF-Offline (last sweep) and state-augmented EnKF in state estimation for the model (6.1) with multiple parameters: Plots (a) and (b) show, respectively, the estimates of $x_t^{(10)}$ and $x_t^{(50)}$ for $t = 1, 2, \ldots, 100$; and plots (c) and (d) show, respectively, the estimates of $x_t^{(10)}$ and $x_t^{(50)}$ for $t = 51, 52, \ldots, 100$, where the the 95% confidence bands are represented by shaded areas.

Figure 6.6 compares the three algorithms in state estimation for one dataset simulated from (6.1). Both SA-LEnKF-Offline and SA-LEnKF-Online worked very well for this example, providing good point estimates as well as uncertainty quantification for the estimates.



(a) α_1 (b) α_2

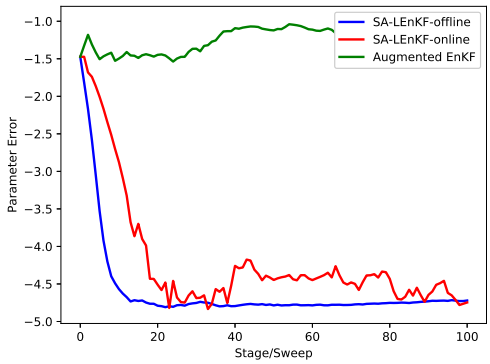
(c) α_3 (d) σ

Figure 6.7. Comparison of SA-LEnKF-Online, SA-LEnKF-Offline and state-augmented EnKF in parameter estimation for the model (6.1) with multiple parameters, where x-axis represents the sweep number for SA-LEnKF-Offline and the stage number for SA-LEnKF-Online and state-augmented EnKF.

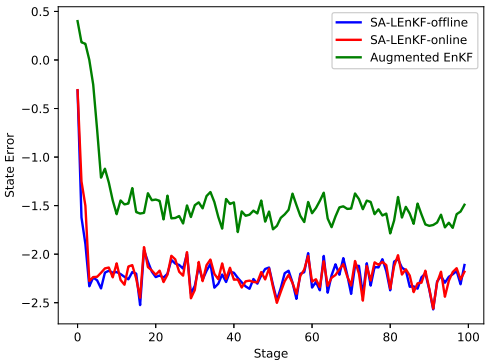
Unfortunately, the state-augmented EnKF performed less satisfactorily for this example, which failed to follow the pattern of the observations in the early stages. Compared to Figure 6.1, we might conclude that including more parameters significantly affects the performance of the state-augmented EnKF in an inverse manner, while it does not for the two SA-EnKF algorithms.

Figure 6.7 compares the three algorithms in parameter estimation, where the plots (a), (b), (c), and (d) show the estimates of α_1, α_2, α_3, and σ, respectively. Both SA-LEnKF algorithms worked reasonably well for this example, while the state-augmented EnKF did

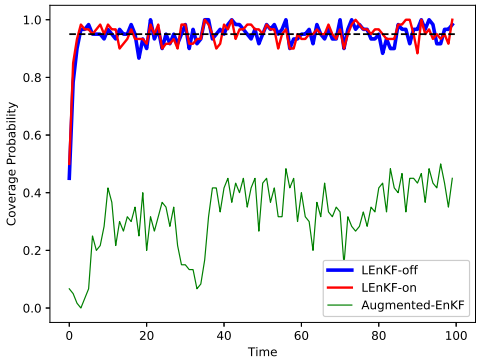
not. The parameter estimates produced by the state-augmented EnKF can be far from the true values even with a large number of stages.



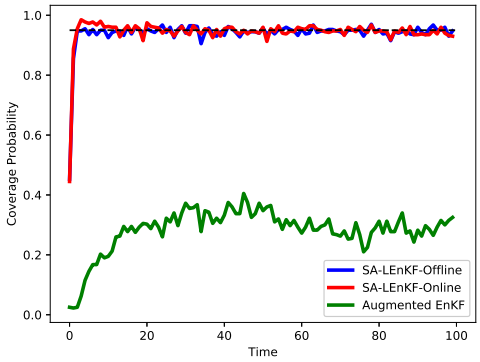
(a) $\log(\text{RMSE}_{t/l}(\boldsymbol{\theta}))$



(b) $\log(\text{RMSE}_t(x))$



(c) Coverage Probabilities with one dataset



(d) Coverage probabilities with ten datasets

Figure 6.8. Comparison of SA-LEnKF-Online, SA-LEnKF-Offline and state-augmented EnKF in parameter and state estimation for the model (6.1) with multiple parameters.

Figure 6.8 (a) shows that in parameter estimation, the two SA-LEnKF algorithms performed similarly and they both significantly outperformed the state-augmented EnKF. Figure 6.8 (b) shows that in state estimation, SA-LEnKF-Offline and SA-LEnKF-Online performed almost the same, both significantly outperforming the state-augmented EnKF. Figure 6.8 (c) shows coverage probabilities of the 95% confidence intervals of the state estimates, where, for each stage t, the coverage probability was calculated by averaging over the coverage status of each component of x_t. Figure 6.8 (d) shows coverage probabilities averaged

over 10 datasets. The coverage probabilities produced by the two SA-LEnKF algorithms are very close to the nominal level 95%, while those by state-augmented EnKF are only about 30%.

Table 6.3. Comparison of SA-LEnKF-Offline, SA-LEnKF-Online and state-augmented EnKF in state and parameter estimation for the model (6.1) with multiple parameters, where the average values over 10 independent datasets were reported with the standard deviation given in the parentheses. The CPU time (in seconds) was recorded for a single run of each algorithm.

	Ave-MRMSE	Ave-MCP	Ave-MRMSE ($\boldsymbol{\theta}$)	CPU Time
SA-LEnKF-Offline	0.1106(0.0007)	0.9467(0.0014)	0.0104(0.0007)	145.796(1.421)
SA-LEnKF-Online	0.1109(0.0004)	0.9465(0.0010)	0.0095(0.0009)	1.757(0.068)
Augmented EnKF	0.2226(0.0026)	0.2916(0.0211)	0.1558(0.0187)	0.163(0.002)

Table 6.3 summarizes the numerical results of the three algorithms. In summary, the two SA-LEnKF algorithms worked very well for this example, and they both significantly outperformed the state-augmented EnKF in state and parameter estimation. Again, in terms of CPU time, the SA-LEnKF-Offline is much more costly than the other two algorithms.

6.1.4 Sea surface temperature modeling

Consider the reformulated LSTM network in Section 4.2.2 again. Previously we set $\mathrm{e}_t \sim N(0, \sigma_1^2 I)$, $\boldsymbol{\zeta}_t \sim N(0, \sigma_2^2 I)$, $\boldsymbol{\xi}_t \sim N(0, \sigma_3^2 I)$, $\boldsymbol{u}_t \sim N(0, \sigma^2 I)$, where $\sigma_1 = \sigma_2 = \sigma_3 = \sigma = 0.01$ are known. In this example, since σ_1, σ_2 and σ_3 are all artificial parameters introduced for the purpose of state-space modeling, we fix them to small constants in simulations, while we treat σ as the unknown parameter, which measures variation of the observed data. To make the notation consistent with other parts of the dissertation, we denote by $\boldsymbol{\theta} = \{\sigma\}$ the collection of the unknown parameters.

The SA-LEnKF-Online algorithm was applied to train the LSTM model and quantify uncertainty for the resulting state estimates. In the parameter updating step, Remark 5.1 is followed in evaluating the gradient $\nabla_{\boldsymbol{\theta}} \log p(y_t | \boldsymbol{x}_{t-1}, \boldsymbol{\theta})$. The SA-LEnKF-Offline algorithm

can also be applied to the model, but it is too CPU time costly for such a large-scale model. In principle, the state-augmented EnKF algorithm can also be applied to train the model.

The dataset was downloaded from *http://iridl.ldeo.columbia.edu*, which records the sea temperatures of 200 months since January 1960 with the latitude from 14.5S to 5.5S and the longitude from 150.5E to 159.5E. The sea surface has been gridded by every single latitude and longitude, and thus the dimension of h_t is $d = 100$.

For this data set, we set $q = 6$ and $T = 200$, i.e., modeling the data of the first 200 months using an $AR(6)$ LSTM model. The data was scaled into the range $(-1, 1)$ in preprocessing and then scaled back to the original range in results reporting. The SA-LEnKF-Online algorithm was applied to the dataset with the ensemble size $m = 10$, the stage iteration number $\mathcal{K} = 20$, $\alpha = 0.9$, the number of hidden neurons $s = 400$, the state learning rate is $\epsilon_{t,k} = 0.00004/max\{8, k\}^{0.6}$ for $k = 1, \cdots, \mathcal{K}$ and $t = 1, \cdots, T$, and the parameter learning rate is $\gamma_{t,k} = 0.000006/(tk)^{0.9}$ for $k = 1, 2, \ldots, \mathcal{K}$ and $t = 1, 2, \ldots, T$. In the simulation, we fix $\log \sigma_1$, $\log \sigma_2$ and $\log \sigma_3$ to -10.01, and initialized $\log \sigma$ by -10.01. The Metropolis-Hastings algorithm [46], [47] was used in the imputation step as prescribed in Remark 5.1, where a Gaussian random walk proposal with a variance of 0.01 was employed, the sampler was run for 20 iterations, and the last sample was output as the imputed value. The proposed method took 9,091 CPU seconds. The results were summarized in Figures 6.9-6.11.

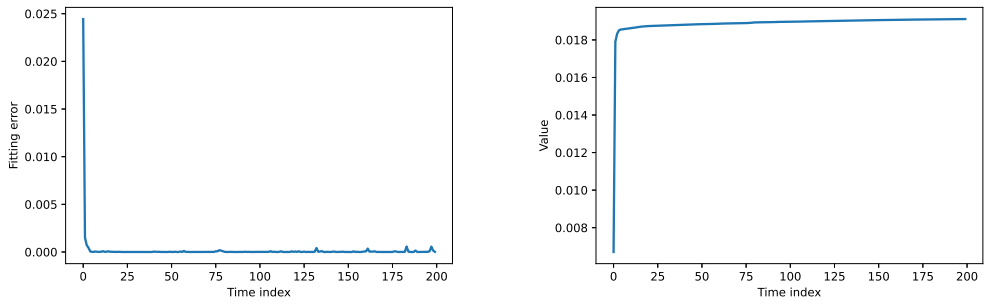


Figure 6.9. Results of SA-LEnKF-Online in state and parameter estimation for the LSTM model: (a) $\text{MSE}_t(x)$ of the state estimates along with stage t; (b) the estimates of σ along with stage t.

Figure 6.9 (a) shows the fitting error along with stages, which shows that the algorithm can converge very fast. The occasional small bumps in the fitting error curve reflect the

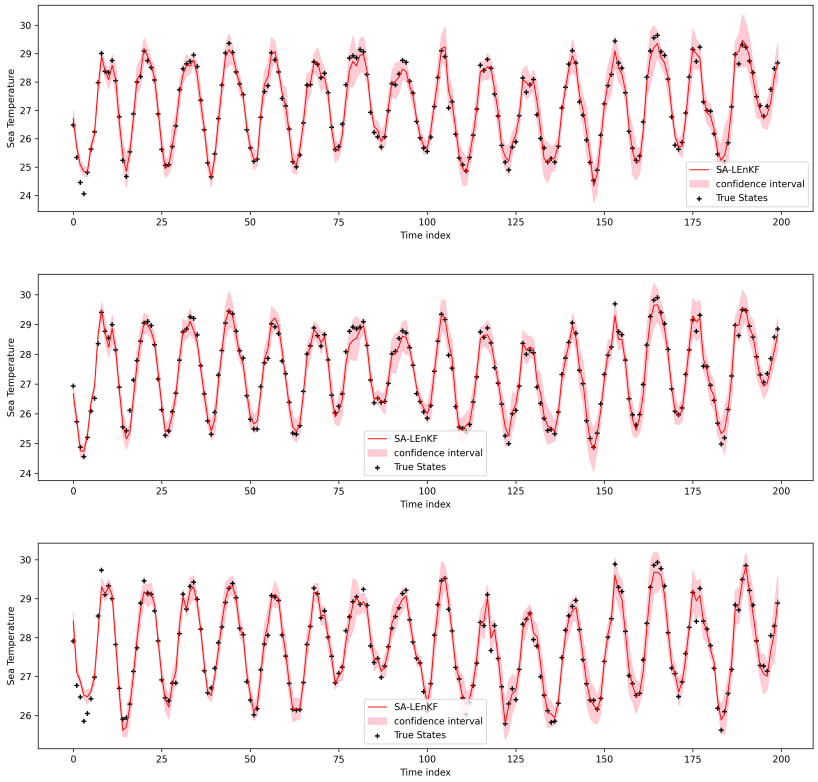
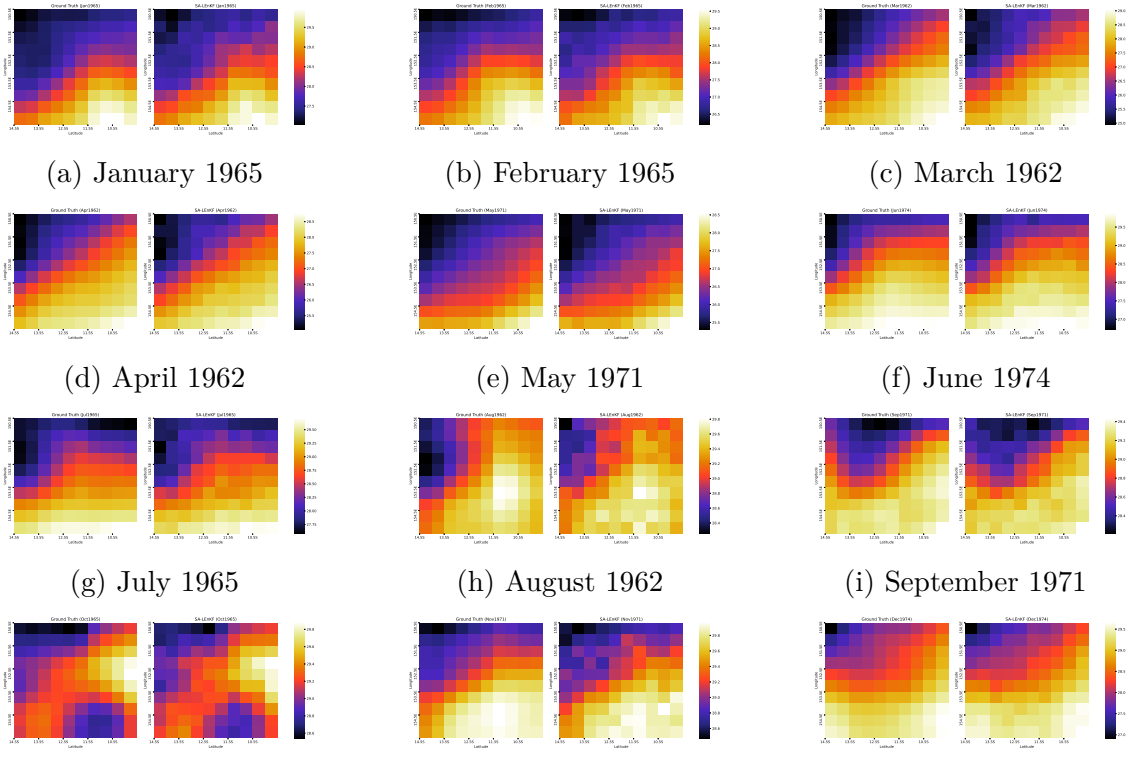


Figure 6.10. Confident bands generated by SA-LEnKF-Online for the LSTM model: The upper, middle and lower panels show, respectively, the estimates of $x_t^{(10)}$, $x_t^{(180)}$ and $x_t^{(190)}$ along with stages, where the true state values are represented by '+', the estimates are represented by solid lines, and the 95% confidence bands are represented by shaded areas.

dependence of the fitting error on observations instead of pre-convergence. Figure 6.9 (b) shows the estimates of the parameter σ along with stages. Figure 6.10 shows the 95% confidence bands produced by the SA-LEnKF-Online algorithm at three selected spatial locations along with stages, which indicate the excellence of the SA-LEnKF-Online algorithm in data fitting and uncertainty quantification. Figure 6.11 compares the heat maps of the true sea temperatures and the fitted sea temperatures by the SA-LEnKF-Online algorithm at 12 selected months. It indicates again the excellence of the SA-LEnKF-Online algorithm in data fitting.

For comparison, we have applied the state-augmented EnKF (Algorithm 2) to the example. However, in this algorithm, the covariance matrix C_t of augmented state needs to



(a) January 1965 (b) February 1965 (c) March 1962

(d) April 1962 (e) May 1971 (f) June 1974

(g) July 1965 (h) August 1962 (i) September 1971

(j) October 1965 (k) November 1971 (l) December 1974

Figure 6.11. Heat maps of the true sea temperatures (left in each subfigure) and the fitted sea temperatures (right in each subfigure) by the SA-LEnKF-Online algorithm at 12 selected months.

be calculated at each stage. Since the augmented state has a dimension even higher than $d_x(> 1M)$, calculation of such a huge covariance matrix directly caused a collapse of our computers. By contrast, in the SA-LEnKF-Online algorithm, the corresponding matrix Q_t is diagonal, which avoids the memory issue suffered by the state-augmented EnKF. This example demonstrates that the SA-LEnKF-Online can work well for ultrahigh-dimensional problems.

6.2 Empirical results of the Extended LEnKF algorithm

This section contains three simulation examples, which are for Poisson regression, nonlinear classification example, and dynamic Poisson spatial model, respectively. The first example illustrates the application of Algorithm 9 for inverse problems, the third example illustrates the application of Algorithm 10 for data assimilation problems. For the second one, the nonlinear latent variable equation is converted to a linear one via a variance-splitting state augmentation method. At the last, we illustrate that the Extended LEnKF algorithm can be applied easily into dynamic network analysis and it possesses the scalability that is necessary for large-scale dynamic network analysis.

6.2.1 Poisson regression

Consider a Poisson regression reformulated in the following equations:

$$\begin{aligned} y_{\mathrm{i}} &= \boldsymbol{x}_{\mathrm{i}}\beta + \sigma_0\epsilon_{\mathrm{i}}, \\ z_{\mathrm{i}} &\sim \mathcal{P}ois(\exp(y_{\mathrm{i}})), \end{aligned} \tag{6.3}$$

for $\mathrm{i} = 1, 2, \ldots, N$, where $\boldsymbol{x}_{\mathrm{i}} = (x_{\mathrm{i},1}, x_{\mathrm{i},2}, \ldots, x_{\mathrm{i},p})$ is a p-dimensional vector of explanatory variables, β is a p-dimensional vector of regression coefficients, σ_0 is the standard deviation, and ϵ_{i}'s are iid standard Gaussian random variables.

We generated 10 data sets from (6.3) with $N = 50,000$, $p = 2000$, $\sigma_0 = 0.5$, $(\beta_1, \cdots, \beta_4) = (1.0, 1.0, 1.0, 1.0)$, $(\beta_5, \beta_6, \cdots, \beta_8) = (-1.0, -1.0, -1.0, -1.0)$, and $\beta_{\mathrm{j}} = 0$ for $\mathrm{j} = 9, \cdots, p$. All the variables are generated as in Section 4.1.1.

We let each component of β be subject to the mixture Gaussian prior distribution (4.2), where $\alpha_1 = 1/p = 0.0005$, $\tau_1^2 = 1$ and $\tau_0^2 = 0.001$, and assume that all the components are *a priori* independent. Algorithm 9 was applied to this example with the ensemble size $m = 50$, the mini-batch size $n = 100$, and the learning rate $\epsilon_t = 0.1/\max\{t_0, t\}^{0.6}$ and $t_0 = 200$. The Metroplis-Hastings sampler [46], [47] was used in the imputation step, where a Gaussian random walk proposal with a variance of 4 was employed, the sampler was run for 10 iterations, and the last sample was output as the imputed value. The algorithm was run for 2,000 iterations, which cost 61.2 CPU seconds.

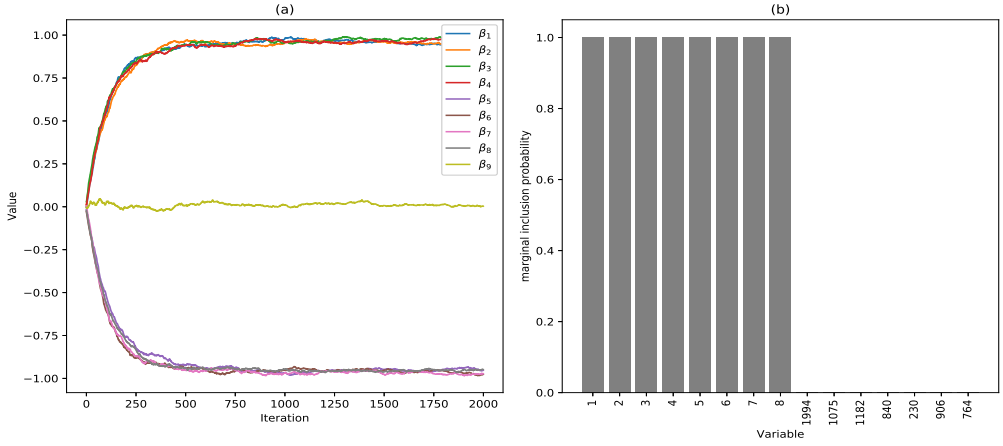


Figure 6.12. Extended LEnKF for large-scale generalized linear regression: (a) Trajectories of $\beta_1, \ldots, \beta_9$ along with iterations; (b) marginal inclusion probabilities of all covariates $x_1, \ldots, x_p$, where the covariates are shown in the rank of marginal inclusion probabilities.

Figure 6.12 summarizes the variable selection results for one data set. The results for other data sets are similar. Figure 6.12(a) shows the sample trajectories of $\beta_1, \beta_2, \ldots, \beta_9$ along with iterations, which indicate that they all have converged to the true values within 500 iterations. Figure 6.12(b) shows the marginal inclusion probability of the variables $\beta_1, \beta_2, \ldots, \beta_p$. From this graph, we can see that each of the 8 true variables (indexed 1-8) has a marginal inclusion probability close to 1, while all false variables have a marginal inclusion probability close to 0. By the median probability rule [84], the true model can be correctly identified. In summary, this example shows that the Extended LEnKF can be applied to non-Gaussian inverse problems.

6.2.2 Nonlinear classification

Consider a nonlinear logistic regression reformulated in the following equations:

$$\begin{aligned} y_{\mathrm{i}} &= 0x_{\mathrm{i},0} + \tfrac{10x_{\mathrm{i},1}^2}{1+x_{\mathrm{i},2}^2} + 5\sin(x_{\mathrm{i},3}x_{\mathrm{i},4}) + x_{\mathrm{i},5} + 0x_{\mathrm{i},6} + \cdots + 0x_{\mathrm{i},99} + \epsilon_{\mathrm{i}}, \\ z_{\mathrm{i}} &\sim \psi\left(\cdot|y_{\mathrm{i}}\right) = \mathcal{B}ernoulli(1/1+\exp(-y_{\mathrm{i}})), \end{aligned} \tag{6.4}$$

where $\mathrm{i} \in \{1, 2, \cdots, N\}$ indexes the observations, $\epsilon_{\mathrm{i}} \sim N(0, \sigma^2)$, and $x_{\mathrm{i},1}$, $x_{\mathrm{i},2}$, ..., $x_{\mathrm{i},99}$ are standard Gaussian random variables and have a mutual correlation coefficient of 0.5. The dataset was simulated with the sample size $N = 200,000$, where about half of z_{i}'s are 1 and the other half are 0.

Suppose that the state propagator in (6.4) is unknown, and we modeled it by a 3-hidden-layer neural network, which consists of 100 input units including a bias unit, 5 units on each hidden layer, and one unit on the output layer. The LeakyRelu $\sigma(x) = 1_{(x<0)}(0.1x) + 1_{(x>=0)}(x)$ is used as the activation function. Let $\boldsymbol{\beta}$ denote the parameter vector, including weights and bias, of the neural network. Let $g(\boldsymbol{x}, \boldsymbol{\beta})$ denote the neural network function. With the variance-splitting state augmentation method , we can rewrite the nonlinear logistic regression model as

$$\boldsymbol{y}_t^{(1)} = g(\boldsymbol{x}_t, \boldsymbol{\beta}) + \boldsymbol{\varepsilon}_t^{(1)}, \quad \boldsymbol{\theta}_t^T = (\boldsymbol{\beta}^T, (\boldsymbol{y}_t^{(1)})^T), \quad \boldsymbol{\varepsilon}_t^{(1)} \sim N(0, \alpha\sigma^2 I_n), \tag{6.5}$$

$$\boldsymbol{y}_t = H_t\boldsymbol{\theta}_t + \boldsymbol{\varepsilon}_t^{(2)} = \boldsymbol{y}_t^{(1)} + \boldsymbol{\varepsilon}_t^{(2)}, \quad \boldsymbol{\varepsilon}_t^{(2)} \sim N(0, (1-\alpha)\sigma^2 I_n), \tag{6.6}$$

$$\boldsymbol{z}_t \sim \psi(\cdot|\boldsymbol{y}_t), \tag{6.7}$$

where $(\boldsymbol{x}_t, \boldsymbol{z}_t)$ denotes a mini-batch of the dataset drawn at stage t, n is the mini-batch size, α is called the variance splitting proportion, $H_t = (0, I)$ is chosen such that $H_t\boldsymbol{\theta}_t = \boldsymbol{y}_t^{(1)}$, the density of $\boldsymbol{\theta}_t$ is given by $\boldsymbol{\pi}(\boldsymbol{\theta}_t) = \boldsymbol{\pi}(\boldsymbol{\beta})\boldsymbol{\pi}(\boldsymbol{y}_t^{(1)}|\boldsymbol{\beta})$, and $\boldsymbol{\pi}(\boldsymbol{\beta})$ denotes the prior density function of $\boldsymbol{\beta}$. We also assume that the components of $\boldsymbol{\beta}$ are *a priori* independent and each follows a mixture Gaussian distribution (4.2) with $\alpha_1 = 0.01$, $\tau_1^2 = 1$ and $\tau_0^2 = 0.05$.

Algorithm 9 was applied to solve the linear inverse problem formed by (6.6)-(6.7), where we set the variance split proportion $\alpha = 0.9$, the ensemble size $m = 20$, the mini-batch size $n = 100$, iteration time $\mathcal{K} = 5$, and the total number of stages $T = 20,000$. In addition, The

learning rate was set to $\epsilon_{t,k} = 5 \times 10^{-4}/k^{0.9}$ with for $k = 1, \cdots, \mathcal{K}$. Each run took about 4577 CPU seconds on a personal computer of 2.9GHz.

In the forecast step, if $t > 0$ and $k = 1$, we let

$$\boldsymbol{\theta}_{t,0}^{a,\mathrm{i}} = \begin{pmatrix} \boldsymbol{\beta}_{t-1,\mathcal{K}}^{a,\mathrm{i}} \\ \boldsymbol{y}_{t,0}^{(1),\mathrm{i}} \end{pmatrix}, \quad \boldsymbol{y}_{t,0}^{(1),\mathrm{i}} \sim \boldsymbol{\pi}(y|\boldsymbol{\beta}_{t-1,\mathcal{K}}^{a,\mathrm{i}}, \boldsymbol{z}_t) \propto \psi(\boldsymbol{z}_t|y)\boldsymbol{\pi}(y|\boldsymbol{\beta}_{t-1,\mathcal{K}}^{a,\mathrm{i}}),$$

where $\psi(\cdot)$ is a Bernoulli distribution, and $y|\boldsymbol{\beta}_{t-1,\mathcal{K}}^{a,\mathrm{i}} \sim N(g(\boldsymbol{x}_t, \boldsymbol{\beta}_{t-1,\mathcal{K}}^{a,\mathrm{i}}), \alpha\sigma^2 I_n)$. while in the imputation step of iteration t, we drew $\boldsymbol{y}_{t,k}^{\mathrm{i}} \sim \boldsymbol{\pi}(y|\boldsymbol{\theta}_{t,k}^{f,\mathrm{i}}, \boldsymbol{z}_t) \propto \psi(\boldsymbol{z}_t|y)\boldsymbol{\pi}(y|\boldsymbol{\theta}_{t,k}^{f,\mathrm{i}})$, where $y_{t,k}^{\mathrm{i}}|\boldsymbol{\theta}_{t,k}^{f,\mathrm{i}} \sim N(H_t\boldsymbol{\theta}_{t,k}^{f,\mathrm{i}}, (1-\alpha)\sigma^2 I_n)$, and $\boldsymbol{z}_{t,l}|y \sim \mathcal{B}ernulli(\exp(y_l))$ for $l = 1, \cdots, n$.

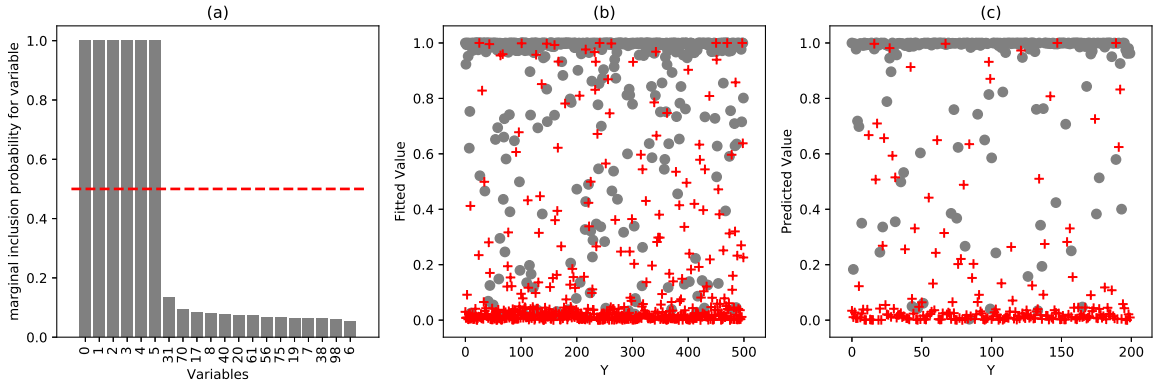


Figure 6.13. Extended LEnKF for a nonlinear classification example: (a) marginal inclusion probabilities of the variables, where the variables are shown in the rank of marginal inclusion probabilities; and (b) fitted value of Y (grey dot for true Y = 1, red cross sign for true Y = 0, randomly selected 500 observations for each class); (c) predicted value of Y (grey dot for true Y = 1, red cross sign for true Y = 0, randomly selected 200 observations for each class).

Figure 6.13 summarizes the results for one dataset. The results for other datasets are similar. Figure 6.13(a) shows the marginal inclusion probabilities of all input variables $x_1, x_2, \ldots, x_p$ with a cutoff value of 0.5 (red dash line). The results are very encouraging: Each of the true variables (indexed 1-5) has a marginal inclusion probability close to 1, while each of the false variables has a marginal inclusion probability close to 0. Figure 6.13(b) shows the scatter plot of the response Y and its fitted value for 1,000 randomly selected training samples, 500 for true $Y = 1$ and 500 for true $Y = 0$. Figure 6.13(c) shows the

scatter plot of the response Y and its predicted value for 400 test samples, 200 for true $Y = 1$ and 200 for true $Y = 0$.

In summary, this example shows that the Extended LEnKF provides an effective and feasible algorithm for training Bayesian neural networks for nonlinear non-Gaussian problems.

6.2.3 Dynamic Poisson spatial model

We illustrate the performance of Algorithm 10 using a synthetic cloud-motion data set, which represents cloud intensities (i.e., counts) at $p = 300$ locations along with a spatial transect at T = 80 time points. The data approximately follows an over-dispersed Poisson distribution, and can be modeled as follows:

$$\begin{aligned} \boldsymbol{x}_t|\boldsymbol{x}_{t-1} &\sim \mathcal{N}_p(\mathbf{M}(\boldsymbol{\gamma})\boldsymbol{x}_{t-1}, \mathbf{Q}(\tau^2, \lambda)), \\ \boldsymbol{y}_t|\boldsymbol{x}_t &\sim \mathcal{N}_{n_t}(\mathbf{H}_t\boldsymbol{x}_t, \sigma^2\mathbf{I}_{n_t}), \\ z_{t,l}|\boldsymbol{y}_t &\sim \mathcal{P}ois(\exp(y_{t,l})), \quad l = 1, \ldots, n_t, \end{aligned} \tag{6.8}$$

where $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \gamma_3)$, $\mathbf{M}(\boldsymbol{\gamma})$ and $\mathbf{Q}$ are defined as in Section 6.1.1. Ten data sets were simulated from (6.8). In the simulation, we set $n_t = 270$ and H_t as an $n_t \times p$-selection matrix; that is, the observation locations were different for different stages. For the parameters, we set $\boldsymbol{\gamma} = (0.3, 0.3, 0.3)$, $\tau = 1$, $\sigma = 0.1$, and $\lambda = 1$. For the initial state values, we set $\boldsymbol{x}_0 \sim \mathcal{N}_n(\boldsymbol{\mu_0}, \boldsymbol{\Sigma}_0)$, where $\boldsymbol{\mu}_0$ is a constant vector of -2, and $(\boldsymbol{\Sigma}_0)_{\mathrm{i,j}} = 0.2\,\mathrm{Mat}(|\mathrm{i} - \mathrm{j}|/5)$. Figure 6.14 shows the state values at 300 locations for $t = 1, 2, \ldots, 80$, whose chaotic behavior implies the challenge of the problem.

Algorithm 10 was applied to the data sets to re-estimate the states $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T$. The algorithm was run with the ensemble size $m = 20$, the stage iteration number $\mathcal{K} = 20$, $k_0 = \mathcal{K}/2$, and the learning rate $\epsilon_{t,k} = 0.1/\max(10, k)^{0.6}$ for $k = 1, 2, \ldots, \mathcal{K}$ and $t = 1, 2, \ldots, T$. The imputation step was accomplished using the Metropolis-Hastings sampler, where each component of $\boldsymbol{y}_t$ was imputed independently and a Gaussian random walk proposal with a variance of 0.01 was employed. For each component of $\boldsymbol{y}_t$, the Metropolis-Hastings sampler was run for 20 iterations and the last sample was imputed as the imputed value. At each stage t, the state was estimated by averaging over the ensembles generated in the last $\mathcal{K}/2$

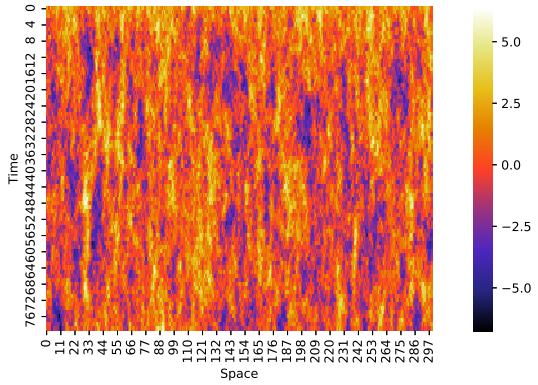


Figure 6.14. State values for $t = 1, 2, \ldots, T$

iterations, and the accuracy of the estimate was measured by RMSE. For comparison, the MCMC-EnKF algorithm [45] (see Algorithm S2 in the supplementary material for the detail) was applied to this example, where the imputation was done as for the Extended LEnKF, but the states were estimated using the EnKF. The ensemble size was set to $m = 20$ as well.

Figure 6.15 compares the estimates of four randomly selected components of $\boldsymbol{x}_t$ produced by the Extended LEnKF and the MCMC-EnKF for one simulated data set. The comparison indicates that LEnKF provides better state estimates as well as better quantification for the uncertainty of the estimates. For example, in Figure 6.15(b), the state values around stages $30 - 40$ are covered by the confidence band of LEnKF, but not by that of MCMC-EnKF. More importantly, as t becomes large, LEnKF can capture pattern of each component of $\boldsymbol{x}_t$, while MCMC-EnKF failed to do so.

Figure 6.16 (a) compares the coverage rates of the 95% confidence intervals produced by LEnKF and MCMC-EnKF, where the coverage rate was calculated by averaging over 300 state components at each stage $t \in \{1, 2, \ldots, 80\}$. Figure 6.16(b) shows the averaged coverage rates over 10 data sets. The comparison shows that LEnKF produced the coverage rates closing to their nominal level, while MCMC-EnKF did not. This implies that LEnKF is able to correctly quantify uncertainty of the estimates as t becomes large. Figure 6.16(c) shows that the LEnKF produced smaller values of RMSE_t's than MCMC-EnKF.

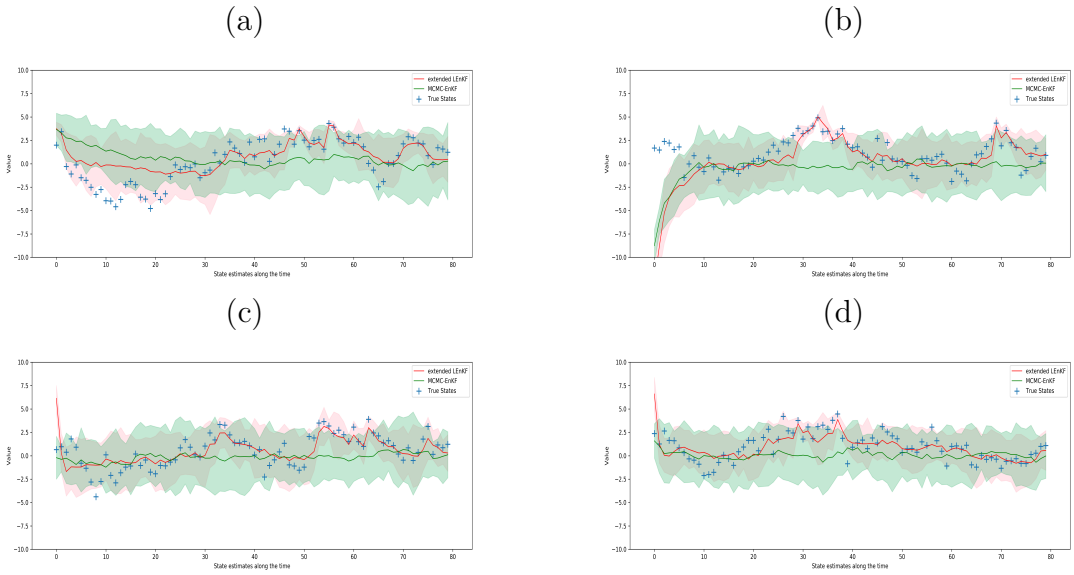


Figure 6.15. State estimates produced by the Extended LEnKF and the MCMC-EnKF for a simulated cloud-motion data set along with stages $t = 1, 2, \ldots, 80$: each plot corresponds to one randomly selected component of $\boldsymbol{x}_t$, where the true state values are represented by '+', the estimates by LEnKF are represented by red lines, the estimates by MCMC-EnKF are represented by green lines, and and their 95% confidence intervals are represented by shaded bands.

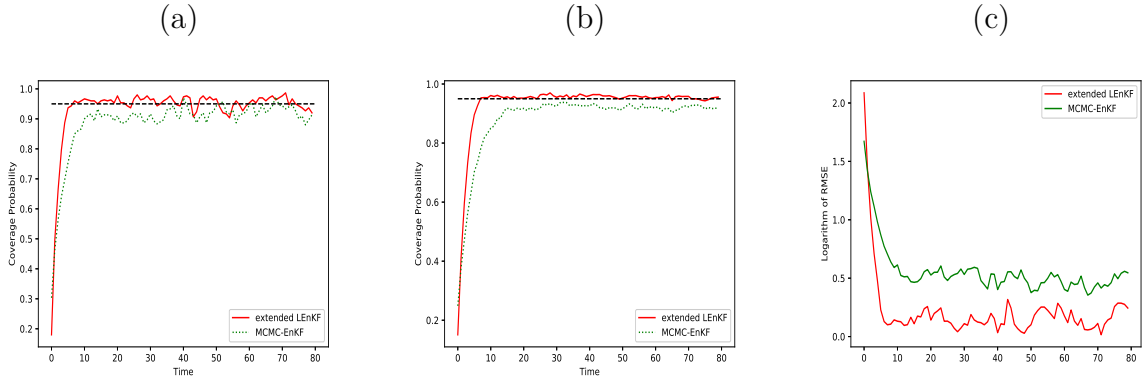


Figure 6.16. Coverage rates of the 95% confidence intervals produced by the Extended LEnKF and MCMC-EnKF for the stages $t = 1, 2, \ldots, 80$: (a) coverage rates with one data set; (b) coverage rates averaged over 10 data sets; (c) $\log(\text{RMSE}_t)$ along with stage t.

Table 6.4 summarizes the numerical results of Extended LEnKF and MCMC-EnKF algorithms for example, where both choices $k_0 = \mathcal{K}/2$ and $k_0 = \mathcal{K} - 1$ have been tried for

Table 6.4. Comparison of the Extended LEnKF and the MCMC-EnKF in state estimation for the dynamic spatial model, where the average values over 10 data sets were reported with the standard deviation given in the parentheses, and the CPU time (in seconds) was recorded for a single run of the algorithm.

	k_0	Ave-MRMSE(x)	Ave-MCP	CPU Time
Extended LEnKF	$\mathcal{K}/2$	1.1699(0.0126)	0.9561(0.0015)	31.686(0.195)
Extended LEnKF	$\mathcal{K}-1$	1.1725(0.0133)	0.9525(0.0017)	26.622(0.215)
MCMC-EnKF	-	1.6327(0.0132)	0.9229(0.0023)	2.704(0.033)

Extended LEnKF. In the table, we used MRMSE(x) to denote the mean of the root-mean-squared-error of the estimates for the states $\boldsymbol{x}_{30}, \boldsymbol{x}_{31}, \ldots, \boldsymbol{x}_T$ calculated on one data set, and used ave-MRMSE(x) to denote the average of MRMSE(x) over 10 data sets. Similarly, we used MCP to denote the coverage probability averaged over states $\boldsymbol{x}_{30}, \boldsymbol{x}_{31}, \ldots, \boldsymbol{x}_T$ calculated on one data set, and used Ave-MCP to denote the average MCP over 10 data sets. The comparison shows that for this example, Extended LEnKF works well with both $k_0 = \mathcal{K}/2$ and $k_0 = \mathcal{K}-1$, and it significantly outperforms MCMC-EnKF. MCMC-EnKF is faster than Extended LEnKF as it performs only one iteration at each stage, while Extended LEnKF performs $\mathcal{K}$ iterations.

6.2.4 Dynamic network analysis

Dynamic networks have been studied in a variety of fields, such as social network analysis, recommendation systems and epidemiology. Latent representation learning (or embedding) has been recognized as one of the most promising approaches for dynamic network analysis. The basic idea is to learn a low-dimensional vector for each node, which encodes the structural properties of a node and its neighborhood. Such low-dimensional representations can benefit a variety of network analytical tasks such as node clustering, link prediction, and graph visualization. The existing methods of latent representation learning for dynamic networks can be roughly grouped into two categories, namely, latent space model-based methods and

dynamic graph neural network-based methods. Refer to Kim, Lee, Xue, *et al.* [85], Kazemi, Goel, Jain, *et al.* [86] and Skarding, Gabrys, and Musial [87] for recent surveys on this topic.

Although the existing methods work well for many dynamic networks, they are usually optimization based and fail to capture uncertainty of the latent representation by noting that the asymptotic distribution of the latent representation is still unclear. An exception is [88], where the latent representation is learned using a Metropolis-within-Gibbs sampler. However, the sampler is not scalable and thus only applicable to small networks. In what follows, we illustrate that the Extended LEnKF algorithm can be used to learn the latent representation for dynamic networks. More precisely, we employ the Extended LEnKF algorithm to sample from the filtering distributions of the latent representation, which facilitates downstream statistical inference for dynamic networks. It is important to note that the Extended LEnKF algorithm possesses the scalability that is necessary for large-scale dynamic network analysis.

1) Two latent space models

Let G_t be the network of observed pairwise links at time t with N_t nodes. Each node can be represented by a d-dimensional latent space, where d is pre-specified. Let $\boldsymbol{x}_t$ be an dN_t-vector, where the subvector $\boldsymbol{x}_{\mathrm{i}t} = (x_{d(\mathrm{i}-1)+1,t}, \ldots, x_{d\mathrm{i},t})$ represents the latent/embedding vector of node i at stage t. We assume that the nodes can move in the latent space along with time/stages, and the latent vector at time $t+1$ only depends on the latent vector at time t, which is the standard Markov assumption.

(i) Dynamic Social Network Latent (DSNL) model

The first latent space model we considered in this dissertation is the DSNL model developed in Sarkar and Moore [89]. In the form of state space models with the subsampling scheme, the DSNL model is given by

$$\begin{aligned} &\boldsymbol{x}_t|\boldsymbol{x}_{t-1} \sim N(\boldsymbol{x}_{t-1}, \sigma_1^2 I_{dN_t}), \\ &\boldsymbol{y}_t|\boldsymbol{x}_t \sim N(H_t\boldsymbol{x}_t, \sigma_2^2 I_{dn_t}), \\ &P(\boldsymbol{z}_t|\boldsymbol{y}_t) = \prod_{\mathrm{i}\sim\mathrm{j},\mathrm{i},\mathrm{j}\in S_t} p_{\mathrm{ij}t} \prod_{\mathrm{i}\not\sim\mathrm{j},\mathrm{i},\mathrm{j}\in S_t} (1-p_{\mathrm{ij}t}), \end{aligned} \tag{6.9}$$

where S_t denotes a set of n_t randomly selected nodes from the network at stage t; $\boldsymbol{z}_t$ denotes the adjacency matrix formed by the nodes in S_t; H_t is the selection matrix corresponding to S_t; i $\sim$ j and i $\not\sim$ j denote existence and absence of a link, respectively; and $p_{\mathrm{ij}t}$ is the probability of a link at stage t. Specifically, $p_{\mathrm{ij}t}$ is defined by

$$p_{\mathrm{ij}t} = \frac{1}{1+\mathrm{e}^{(d_{\mathrm{ij}t}-r_{\mathrm{ij}t})}} K(d_{\mathrm{ij}t}) + \rho(1-K(d_{\mathrm{ij}t})),$$

where $d_{\mathrm{ij}t} = \|\boldsymbol{y}_{\mathrm{i}t} - \boldsymbol{y}_{\mathrm{j}t}\|$ is the Euclidean distance between node i and node j in the latent space at stage t, $\boldsymbol{y}_{\mathrm{i}t}$ is the latent vector of node i in S_t, $r_{\mathrm{ij}t} = \max(\delta_{\mathrm{i},t}, \delta_{\mathrm{j},t})+1$, $\delta_{\mathrm{i},t}$ is the degree of freedom of node i in the network G_t, $K(\cdot)$ is a biquadratic kernel $K(d_{\mathrm{ij}t}) = (1-(d_{\mathrm{ij}t}/r_{\mathrm{ij}t})^2)^2$ if $d_{\mathrm{ij}t} < r_{\mathrm{ij}t}$ and 0 otherwise, and ρ is a constant noise.

The intuition behind this model is that for any two nodes i and j, we can compare their distance with their social reach represented by $r_{\mathrm{ij}t}$. When the distance is smaller than the social reach, the probability of connecting node i and node j is high. When the distance is greater than the social reach, the probability of connection is ρ, which can be considered as a noise probability.

(ii) Dynamic Latent Distance (DLD) model

Another latent space model we considered in the dissertation is the DLD model developed in Sewell and Chen [88]. In the form of state space models with the subsampling scheme, the DLD model is given by

$$\begin{aligned} \boldsymbol{x}_t|\boldsymbol{x}_{t-1} &\sim N(\boldsymbol{x}_{t-1}, \sigma_1^2 I_{dN_t}), \\ \boldsymbol{y}_t|\boldsymbol{x}_t &\sim N_{m_t}(H_t\boldsymbol{x}_t, \sigma_2^2 I_{dn_t}), \\ P(\boldsymbol{z}_t|\boldsymbol{y}_t) &= \prod_{\mathrm{i}\neq\mathrm{j},\mathrm{i},\mathrm{j}\in S_t} \frac{\exp\left(z_{\mathrm{ij}t}\eta_{\mathrm{ij}t}\right)}{1+\exp\left(\eta_{\mathrm{ij}t}\right)}, \\ \eta_{\mathrm{ij}t} :&= \log\left(\frac{\mathbb{P}\left(z_{\mathrm{ij}t}=1|\boldsymbol{y}_t\right)}{\mathbb{P}\left(z_{\mathrm{ij}t}=0|\boldsymbol{y}_t\right)}\right) = \beta_{\mathrm{in}}\left(1-\frac{d_{\mathrm{ij}t}}{r_{\mathrm{j}t}}\right) + \beta_{out}\left(1-\frac{d_{\mathrm{ij}t}}{r_{\mathrm{i}t}}\right), \end{aligned} \tag{6.10}$$

where S_t, H_t, $d_{\mathrm{ij}t}$ and $r_{\mathrm{i}t}$ are as defined in (6.9), and the parameters β_{in} and β_{out} are pre-specified constants which measure the global popularity and activity effects of the network, respectively. Note that this model deals with directed networks as well as undirected ones, while the DSNL model allows undirected edges only. In this dissertation, we consider undirected networks only.

2) A dynamic social network analysis

This section studies a college messaging dynamic network downloaded at https://snap.stanford.edu/data/. It's comprised of private messages sent on an online social network at the University of California, Irvine, where each node represents a user. Users could search the network for others and then initiate conversation based on profile information. An edge (u, v, t) means that user u sent a private message to user v at time t. To construct an undirected user-user interaction network, we set an edge between user u and user v for day t, if u has messaged v or v has messaged u on day t. The resulting dynamic network contains 672 users with a time span of 25 days. Figure 6.17 shows the dynamic network on six selected days, which indicates that the network changes along with time.

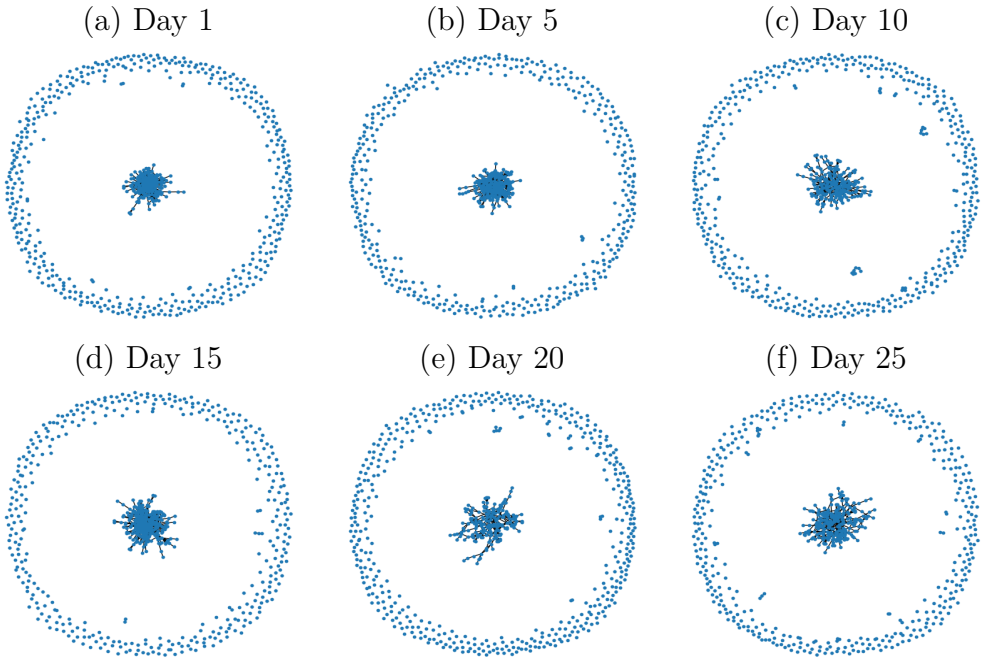


Figure 6.17. College Messaging networks on Days 1, 5, 10, 15, 20 and 25.

We modeled the college messaging network by the DSNL and DLD models and trained the models using Algorithm 10. For the DSNL model, we set $\rho = 0.001$, the ensemble size $m = 20$, $\sigma_1 = \sigma_2 = 0.1$, $d = 5$, $n_t = 250$, the iteration number $\mathcal{K} = 20$, and the learning rate $\epsilon_{t,k} = 0.4/max(10, k)^{0.6}$ for $k = 1, 2, \ldots, \mathcal{K}$ and $t = 1, 2, \ldots, T$. We applied Metropolis-Hastings algorithm for imputation, where a Gaussian random walk proposal with variance 0.01 was used. To impute the latent vector for a selected node, the Metropolis-Hastings algorithm was run for 50 iterations and the sample simulated at the last iteration was output as the imputed value. At each stage t, the state vector was estimated by averaging over the ensemble and over the last $\mathcal{K}/2$ iterations. For the DLD model, we set $\beta_{in} = \beta_{out} = 1.5$, and set other parameters to the same values as those used for the DSNL model.

For comparison, the MCMC-EnKF algorithm was applied to the two models, where the latent vector of each node was imputed in the same procedure as used for the DSNL and DLD models. Table 6.5 compares the AUC values, i.e., the areas under the ROC curves, produced by the two algorithms with the DSNL and DLD models. The comparison shows that the Extended LEnKF algorithm produced almost the same AUC values as the

MCMC-EnKF algorithm, but cost much less CPU time. The MCMC-EnKF algorithm is more costly as it consists of a single iteration at each stage and does not allow the use of mini-batch data. As a result, it needs to invert an $(dN_t) \times (dN_t)$-matrix for calculating the Kalman gain matrix at each stage t, which makes the algorithm unscalable for large-scale networks. In contrast, the Extended LEnKF algorithm is essentially a sequential stochastic gradient MCMC algorithm, which is scalable with respect to large-scale networks due to the subsampling scheme it employed in simulations.

Table 6.5. Comparison of the Extended LEnKF and MCMC-EnKF algorithms for the college messaging dynamic network, where "Avg-AUC" denotes the averaged AUC values over all 25 days and its standard deviation is given in the parentheses. The CPU time (in seconds) was recorded for a single run of each algorithm.

Algorithm	Model	Avg-AUC	CPU Time
Extended LEnKF	DSNL	0.8702(0.0177)	2014
	DLD	0.9388(0.0128)	1700
MCMC-EnKF	DSNL	0.8670(0.0186)	17201
	DLD	0.9385(0.0128)	13008

To show that the Extended LEnKF algorithm can be used for uncertainty quantification for dynamic networks, we plotted in Figure 6.18 and Figure 6.19 the box-plots of the link probabilities produced with the DSNL and DLD models, respectively. For comparison, the link probabilities produced by the MCMC-EnKF algorithm are also plotted. As implied by the plots, uncertainty of the link probability can be easily measured based on the samples produced by the Extended LEnKF algorithm. The MCMC-EnKF produced similar box plots to the Extended LEnKF for this example. However, as implied by Figure 6.16, they might be biased in terms of inference.

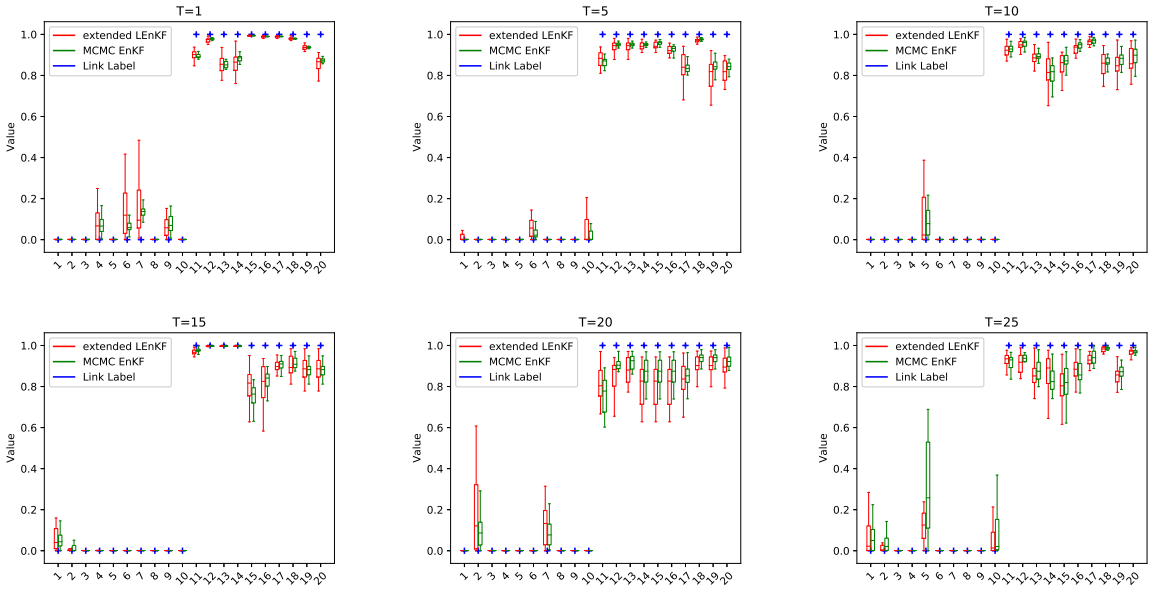


Figure 6.18. Box-plots of link probabilities produced with the DSNL model: fitted link probabilities for 10 pairs of nodes with edges and 10 pairs of nodes without edges are plotted for day 1, 5, 10, 15, 20 and 25.

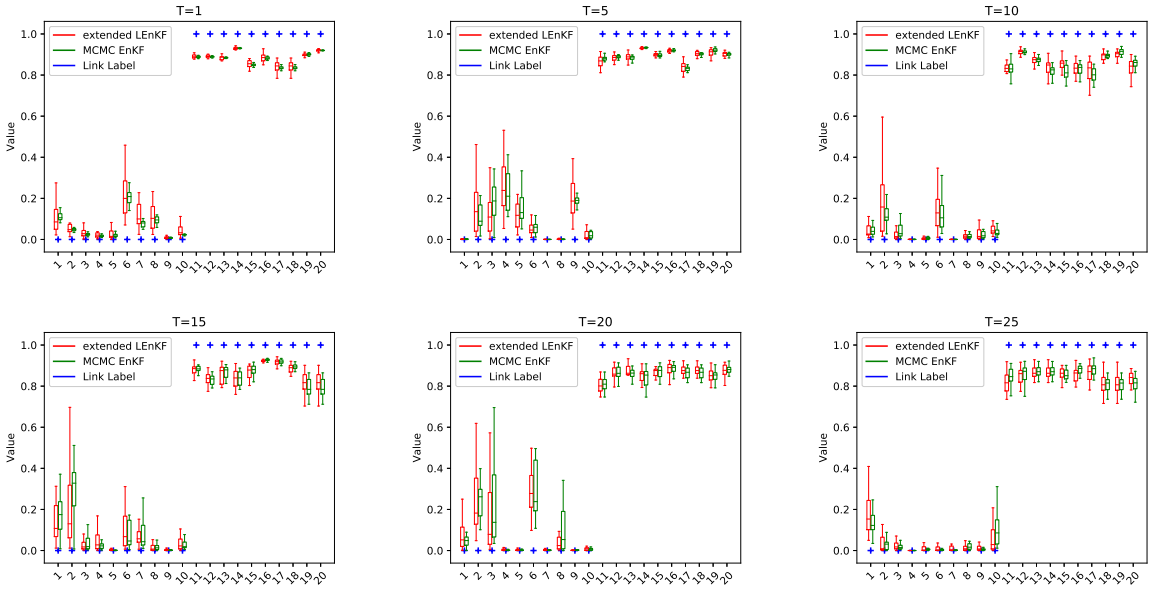


Figure 6.19. Box-plots of link probabilities produced with the DLD model: fitted link probabilities for 10 pairs of nodes with edges and 10 pairs of nodes without edges are plotted for day 1, 5, 10, 15, 20 and 25.

7. SUMMARY AND DISCUSSION

7.1 Summary

In the first part of the dissertation, we propose a new particle filtering algorithm, called LEnKF, by reformulating the EnKF under the framework of Langevin dynamics. The LEnKF converges to the right filtering distribution in Wasserstein distance and thus can be used for uncertainty quantification. Extensions of the LEnKF algorithm are explored in the second part of the dissertation. First, we develop the SA-LEnKF-Online algorithm as an effective and efficient method for simultaneously estimating the state and parameters for long series, large scale and high-dimensional dynamic systems. Second, we extend the LEnKF algorithm to non-Gaussian systems by introducing a latent Gaussian measurement variable to the state space model. Both two extensions inherit the scalability of the LEnKF with respect to the dimension and sample size.

7.2 Discussion

There are a number of open avenues for future investigation. First, we plan to use the proposed algorithms to enhance the safety of motion planning. As automated vehicles has increasingly influenced our mobility behavior, the topic of safe motion planning plays a pivotal role. Our algorithms can be conveniently used for uncertainty quantification in motion planning, which can significantly improve the reliability of decision making during movement, since the environment cannot be modelled perfectly. Second, our algorithms can also be applied to natural language processing (NLP) tasks. Uncertainty quantification is very useful at enhancing model performances in various NLP tasks, such as sentiment analysis, named entity recognition, language modeling using convolutional and recurrent neural network models, or other NLP-based subjects such as image captioning. Our empirical experiments have already proved that we could reformulate RNN models into a state-space models, and then our algorithms could be applied conveniently. Hence, we believe that our algorithms with uncertainty quantification can yield better decisions and potentially advances the development of trustworthy AI.

REFERENCES

[1] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *ICML*, 2011.

[2] N. Ding, Y. Fang, R. Babbush, C. Chen, R. D. Skeel, and H. Neven, "Bayesian sampling using stochastic gradient thermostats," in *NIPS*, 2014.

[3] S. Ahn, A. K. Balan, and M. Welling, "Bayesian posterior sampling via stochastic gradient fisher scoring," in *ICML*, 2012.

[4] T. Chen, E. B. Fox, and C. Guestrin, "Stochastic gradient hamiltonian monte carlo," in *ICML*, 2014.

[5] M. Betancourt, "The fundamental incompatibility of scalable Hamiltonian Monte Carlo and naive data subsampling," in *ICML*, 2015.

[6] C. Li, C. Chen, D. Carlson, and L. Carin, "Preconditioned stochastic gradient langevin dynamics for deep neural networks," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16, Phoenix, Arizona: AAAI Press, 2016, pp. 1788–1794. [Online]. Available: http://dl.acm.org/citation.cfm?id=3016100.3016149.

[7] Y.-A. Ma, T. Chen, and E. B. Fox, "A complete recipe for stochastic gradient mcmc," in *NIPS*, 2015.

[8] C. Nemeth and P. Fearnhead, "Stochastic gradient Markov chain Monte Carlo," *arXiv:1907.0698*
2019.

[9] S. L. Scott, A. W. Blocker, F. V. Bonassi, H. A. Chipman, E. I. George, and R. E. McCulloch, "Bayes and big data: The consensus monte carlo algorithm," *International Journal of Management Science and Engineering Management*, vol. 11, no. 2, pp. 78–88, 2016.

[10] C. Li, S. Srivastava, and D. Dunson, "Simple, scalable and accurate posterior interval estimation," *Biometrika*, vol. 104, no. 3, pp. 665–680, 2017.

[11] S. Srivastava, C. Li, and D. B. Dunson, "Scalable bayes via barycenter in wasserstein space," *Journal of Machine Learning Research*, vol. 19, pp. 1–35, 2018.

[12] J. Xue and F. Liang, "Double-parallel monte carlo for bayesian analysis of big data," *Statistics and Computing*, vol. 29, pp. 23–32, 2019.

[13] H. Chen, D. Seita, X. Pan, and J. Canny, "An efficient minibatch acceptance test for metropolis-hastings," *arXiv preprint arXiv:1610.06848*, 2016.

[14] A. Korattikara, Y. Chen, and M. Welling, "Austerity in mcmc land: Cutting the metropolis-hastings budget," in *International Conference on Machine Learning*, 2014, pp. 181–189.

[15] R. Bardenet, A. Doucet, and C. Holmes, "Towards scaling up markov chain monte carlo: An adaptive subsampling approach," in *International Conference on Machine Learning*, 2014, pp. 405–413.

[16] D. Maclaurin and R. P. Adams, "Firefly monte carlo: Exact mcmc with subsets of data," in *IJCAI*, 2014.

[17] R. Bardenet, A. Doucet, and C. C. Holmes, "On markov chain monte carlo methods for tall data," *Journal of Machine Learning Research*, vol. 18, 47:1–47:43, 2017.

[18] J. Bierkens, P. Fearnhead, and G. Roberts, "The zig-zag process and super-efficient Monte Carlo for Bayesian analysis of big data," *Annals of Statistics*, vol. 47, no. 3, pp. 1288–1320, 2019.

[19] A. Bouchard Coté, S. Vollmer, and A. Doucet, "The bouncy particle sampler: A non-reversible rejection-free Markov chain Monte Carlo method," *Journal of the American Statistical Association*, vol. 113, pp. 855–867, 2018.

[20] F. Liang, J. Kim, and Q. Song, "A bootstrap metropolis-hastings algorithm for bayesian analysis of big data," *Technometrics*, vol. 58, no. 3, pp. 304–318, 2016.

[21] E. Fong, S. Lyddon, and C. Holmes, "Scalable nonparametric sampling from multimodal posteriors with the posterior bootstrap," in *ICML*, 2019.

[22] R. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, pp. 35–45, 1960.

[23] J. K. Uhlmann, "Algorithms for multiple-target tracking," *American Scientist*, vol. 80, no. 2, pp. 128–141, 1992. [Online]. Available: http://www.jstor.org/stable/29774599.

[24] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *Signal Processing, Sensor Fusion, and Target Recognition VI*, I. Kadar, Ed., vol. 3068, SPIE, 1997, pp. 182–193.

[25] G. Evensen, "Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics," *J. Geophys. Res.*, vol. 99, pp. 10 143–10 162, 1994.

[26] K. Law, H. Tembine, and R. Tempone, "Deterministic mean-field ensemble kalman filtering," *SIAM J. Sci. Comput.*, vol. 38, no. 3, A1251–A1279, 2016.

[27] F. Le Gland, V. Monbet, and V.-D. Tran, "Large sample asymptotics for the ensemble kalman filter," *Research report RR-7014, INRIA*, 2009.

[28] E. Bergou, S. Gratton, and J. Mandel, "On the convergence of a non-linear ensemble kalman smoother," *Applied Numerical Mathematics*, vol. 137, pp. 151–168, 2019.

[29] E. Kwiatkowski and J. Mandel, "Convergence of the square root ensemble kalman filter in the large ensemble limit," *SIAM/ASA J. Uncertainty Quantification*, vol. 3, pp. 1–17, 2015.

[30] A. Doucet, N. de Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*. New York: Springer, 2001.

[31] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *IEE Proceedings F - Radar and Signal Processing*, vol. 140, no. 2, pp. 107–113, 1993.

[32] O. Cappé, A. Guillin, J. Martin, and C. Robert, "Population monte carlo," *Journal of Computational and Graphical Statistics*, vol. 13, no. 4, pp. 907–929, 2004.

[33] E. Lorenz, "Predictability—a problem partly solved," in *Seminar on Predictability*, T. Palmer and R. Hagedorn, Eds., Cambridge University Press, 1996, ch. 3, pp. 40–58.

[34] D. P. Dee and A. M. da Silva, "Maximum-Likelihood Estimation of Forecast and Observation Error Covariance Parameters. Part I: Methodology," *Monthly Weather Review*, vol. 127, no. 8, pp. 1822–1834, Aug. 1999.

[35] S. Gibson and B. Ninness, "Robust maximum-likelihood estimation of multivariable dynamic systems," *Automatica*, vol. 41, no. 10, pp. 1667–1682, 2005, ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2005.05.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109805001810.

[36] D. P. Dee, "On-line Estimation of Error Covariance Parameters for Atmospheric Data Assimilation," *Monthly Weather Review*, vol. 123, no. 4, pp. 1128–1145, Apr. 1995.

[37] J. R. Stroud and T. Bengtsson, "Sequential State and Variance Estimation within the Ensemble Kalman Filter," *Monthly Weather Review*, vol. 135, no. 9, pp. 3194–3208, Sep. 2007.

[38] C. Aicher, S. Putcha, C. Nemeth, P. Fearnhead, and E. Fox, "Stochastic gradient mcmc for nonlinear state space models," *ArXiv*, vol. abs/1901.10568, 2019.

[39] J. Anderson, "An ensemble adjustment filter for data assimilation," *Monthly Weather Review*, vol. 129, pp. 2884–2903, 2001.

[40] S.-J. Baek, B. Hunt, E. Kalney, E. Ott, and I. Szunyogh, "Local ensemble kalman filtering in the presence of model bias," *Tellus*, vol. 58A, pp. 293–306, 2006.

[41] S. Gillijns and B. De Moor, "Model error estimation in ensemble data assimilation," *Nonlinear Processes in Geophysics*, vol. 14, pp. 59–71, 2007.

[42] T. DelSole and X. Yang, "State and parameter estimation in stochastic dynamical models," *Physica D*, vol. 239, pp. 1781–1788, 2010.

[43] X. Yang and T. DelSole, "Using the ensemble kalman filter to estimate multiplicative model parameters," *Tellus*, vol. 61, pp. 601–609, 2009.

[44] H. Robbins and S. Monro, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.

[45] M. Katzfuss, J. R. Stroud, and C. K. Wikle, "Ensemble kalman methods for high-dimensional hierarchical dynamic space-time models," *Journal of the American Statistical Association*, vol. 115, no. 530, pp. 866–885, 2020.

[46] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, pp. 1087–1091, 1953.

[47] W. Hastings, "Monte carlo sampling methods using markov chain and their applications," *Biometrika*, vol. 57, pp. 97–109, 1970.

[48] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions and the bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 721–741, 1984.

[49] S. Aanonsen, G. Naevdal, D. Oliver, A. Reynolds, and B. Valles, "The ensemble kalman filter in reservoir engineering—a review," *SPE Journal*, vol. 14, no. 3, pp. 393–412, 2009.

[50] G. Evensen and P. Van Leeuween, "Assimilation of geosat altimeter data for the agulhas current using the ensemble kalman filter with a quasi-geostrophic model," *Monthly Weather Review*, vol. 124, pp. 85–96, 1996.

[51] P. Houtekamer and H. Mitchell, "A sequential ensemble kalman filter for atmospheric data assimilation," *Monthly Weather Review*, vol. 129, pp. 123–137, 2011.

[52] R. Shumway and D. Stoffer, *Time Series Analysis and Its Applications with R Examples*. New York: Springer, 2006.

[53] M. Iglesias, K. Law, and A. Stuart, "Ensemble kalman methods for inverse problems," *Inverse Problems*, vol. 29, no. 4, p. 045 001, 2013.

[54] O. Ernst, B. Sprungk, and H.-J. Starkloff, "Analysis of the ensemble and polynomial chaos kalman filters in bayesian inverse problems," *SIAM/ASA J. Uncertainty Quantification*, vol. 3, pp. 823–851, 2015.

[55] S. Patterson and Y. W. Teh, "Stochastic gradient riemannian langevin dynamics on the probability simplex," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2013, pp. 3102–3110. [Online]. Available: http://papers.nips.cc/paper/4883-stochastic-gradient-riemannian-langevin-dynamics-on-the-probability-simplex.pdf.

[56] M. Girolami and B. Calderhead, "Riemann manifold langevin and hamiltonian monte carlo methods (with discussion)," *Journal of the Royal Statistical Society, Series B*, vol. 73, no. 2, pp. 123–214, 2011.

[57] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 2933–2941. [Online]. Available: http://papers.nips.cc/paper/5486-identifying-and-attacking-the-saddle-point-problem-in-high-dimensional-non-convex-optimization.pdf.

[58] Q. Song, Y. Sun, M. Ye, and F. Liang, "Extended stochastic gradient Markov chain Monte Carlo for large-scale Bayesian variable selection," *Biometrika*, vol. 107, no. 4, pp. 997–1004, 2020.

[59] C. Chen, N. Ding, and L. Carin, "On the convergence of stochastic gradient mcmc algorithms with high-order integrators," in *Advances in Neural Information Processing Systems*, 2015, pp. 2278–2286.

[60] W. Teh, A. Thiery, and S. Vollmer, "Consistency and fluctuations for stochastic gradient langevin dynamics," *Journal of Machine Learning Research*, vol. 17, pp. 1–33, 2016.

[61] A. Saumard and J. Wellner, "Log-concavity and strong log-concavity: A review," *Statistics Surveys*, vol. 8, p. 45, 2014.

[62] H. J. Brascamp and E. H. Lieb, "On extensions of the brunn-minkowski and prékopa-leindler theorems, including inequalities for log concave functions, and with an application to the diffusion equation," in *Inequalities*, Springer, 2002, pp. 441–464.

[63] A. S. Dalalyan and A. G. Karagulyan, "User-friendly guarantees for the langevin monte carlo with inaccurate gradient," *Stochastic Processes and Their Applications*, vol. 129, no. 12, pp. 5278–5311, 2019.

[64] X. Cheng and P. L. Bartlett, "Convergence of langevin mcmc in kl-divergence," *PMLR 83*, no. 83, pp. 186–211, 2018.

[65] A. Durmus and E. Moulines, "Nonasymptotic convergence analysis for the unadjusted langevin algorithm," *The Annals of Applied Probability*, vol. 27, no. 3, pp. 1551–1587, 2017.

[66] E. George and R. McCulloch, "Variable selection via gibbs sampling," *Journal of the American Statistical Association*, vol. 88, pp. 881–889, 1993.

[67] F. Liang, Y. Cheng, Q. Song, J. Park, and P. Yang, "A resampling-based stochastic approximation method for analysis of large geostatistical data," *Journal of the American Statistical Association*, vol. 108, pp. 325–339, 2013.

[68] Y. Sun, Q. Song, and F. Liang, "Consistent sparse deep learning: Theory and computation," *Manuscript*, submitted, 2019.

[69] J. Saetrom and H. Omre, "Uncertainty quantification in the ensemble kalman filter," *Scandinavian Journal of Statistics*, vol. 40, no. 4, pp. 868–885, 2013. DOI: 10.1111/sjos.12039.

[70] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[71] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, May 2009, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2008.137.

[72] A. Graves, A.-r. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649, 2013.

[73] O. Cappé, E. Moulines, and T. Ryden, *Inference in Hidden Markov Models*. New York: Springer, 2005.

[74] A. Benveniste, M. Métivier, and P. Priouret, *Adaptive Algorithms and Stochastic Approximations*. Berlin: Springer, 1990.

[75] W. Deng, X. Zhang, F. Liang, and G. Lin, "An adaptive empirical bayesian method for sparse deep learning," in *Advances in Neural Information Processing Systems*, 2019.

[76] W. Deng, G. Lin, and F. Liang, "A contour stochastic gradient langevin dynamics algorithm for simulations of multi-modal distributions," in *NeurIPS*, 2020.

[77] S. Kim, Q. Song, and F. Liang, "Stochastic gradient langevin dynamics algorithms with adaptive drifts," *arXiv:2009.09535*, 2020.

[78] M. Raginsky, A. Rakhlin, and M. Telgarsky, "Non-convex learning via stochastic gradient Langevin dynamics: A nonasymptotic analysis," in *Proceedings of the 2017 Conference on Learning Theory*, S. Kale and O. Shamir, Eds., ser. Proceedings of Machine Learning Research, vol. 65, Amsterdam, Netherlands: PMLR, 2017, pp. 1674–1703.

[79] H. Chen and Y. Zhu, "Stochastic approximation procedures with randomly varying truncations," *Science in China Series A-Mathematics, Physics, Astronomy & Technological Science*, vol. 29, no. 9, pp. 914–926, 1986.

[80] C. Andrieu, E. Moulines, and P. Priouret, "Stability of stochastic approximation under verifiable conditions," *SIAM Journal on Control and Optimization*, vol. 44, no. 1, pp. 283–312, 2005.

[81] Y. W. Teh, A. H. Thiery, and S. J. Vollmer, "Consistency and fluctuations for stochastic gradient langevin dynamics," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 193–225, 2016, ISSN: 1532-4435.

[82] C. Li, C. Chen, D. Carlson, and L. Carin, "Preconditioned stochastic gradient langevin dynamics for deep neural networks," in *AAAI*, 2016.

[83] K. Bhatia, Y.-A. Ma, A. D. Dragan, P. L. Bartlett, and M. I. Jordan, "Bayesian robustness: A nonasymptotic viewpoint," *arXiv preprint arXiv:1907.11826*, 2019.

[84] M. Barbieri and J. Berger, "Optimal predictive model selection," *Annals of Statistics*, vol. 32, no. 3, pp. 870–897, 2004.

[85] B. Kim, K. Lee, L. Xue, and X. Niu, "A review of dynamic network models with latent variables," *Statistics Surveys*, vol. 12, pp. 105–135, 2018.

[86] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation learning for dynamic graphs: A survey," *Journal of Machine Learning Research*, vol. 21, no. 70, pp. 1–73, 2020. [Online]. Available: http://jmlr.org/papers/v21/19-447.html.

[87] J. Skarding, B. Gabrys, and K. Musial, "Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey," *IEEE Access*, vol. 9, pp. 79 143–79 168, 2021. DOI: 10.1109/ACCESS.2021.3082932.

[88] D. K. Sewell and Y. Chen, "Latent space models for dynamic networks," *Journal of the American Statistical Association*, vol. 110, no. 512, pp. 1646–1657, 2015.

[89] P. Sarkar and A. Moore, "Dynamic social network analysis using latent space models," *ACM SIGKDD Explorations Newletter*, vol. 7, no. 2, pp. 31–40, 2005.

VITA

Peiy Zhang was born in August 1993 in China. In 2015, she obtained a B.S. degree in Statistics from the School of Mathematical Sciences at Zhejiang University. Having received an Master degree in Statistics from Cornell University in May 2016, she joined Purdue University's department of Statistics in January 2017. At Purdue University, she earned a Joint Statistics and Computer Science M.S. Degree in May 2021, and Ph.D. degree in Statistics in August 2021. Peiyi's research interests include machine learning, online learning, advanced MCMC algorithms, and uncertainty quantification. After graduation, she would join Facebook, Inc. as a Research Data Scientist.