An Extended Langevinized Ensemble Kalman Filter for non-Gaussian Dynamic Systems

Peiyi Zhang, Tianning Dong and Faming Liang

Department of Statistics
Purdue University
West Lafayette, IN 47907
Email: fmliang@purdue.edu

SUMMARY

State estimation for large-scale non-Gaussian dynamic systems remains an unresolved issue, given nonscalability of the existing particle filter algorithms. To address this issue, this paper extends the Langevinized ensemble Kalman filter (LEnKF) algorithm to non-Gaussian dynamic systems by introducing a latent Gaussian measurement variable to the dynamic system. The extended LEnKF algorithm can converge to the right filtering distribution as the number of stages become large, while inheriting the scalability of the LEnKF algorithm with respect to the sample size and state dimension. The performance of the extended LEnKF algorithm is illustrated by dynamic network embedding and dynamic Poisson spatial models.

Keywords and phrases: Dynamic Network Embedding, Ensemble Kalman Filter, Sequential Monte Carlo, State Space Model

1 Introduction

State space models (SSMs) are ubiquitous in modeling dynamic systems in fields as diverse as biology, finance, and engineering. Consider a general form of non-Gaussian SSMs:

$$\begin{aligned} x_t &= g(x_{t-1}) + u_t, \quad u_t \sim \mathcal{N}_p(0, U_t), \\ \boldsymbol{z}_t &\sim f(\cdot|x_t), \end{aligned} \tag{1.1}$$

where t indexes stages for $t = 1, 2, \ldots, T$; $x_t \in \mathbb{R}^p$ and $\boldsymbol{z}_t \in \mathbb{R}^{N_t}$ are called, respectively, the state and measurement variables/vectors at stage t; p is the dimension of the state variable x_t; and N_t is the dimension of the measurement variable $\boldsymbol{z}_t$. For the problems where a single measurement is taken from each sample, N_t represents the sample size at stage t. In model (1.1), the first equation is called the state evolution equation and the state propagator $g(\cdot)$ can be highly nonlinear, and the second equation is called the measurement equation and the distribution $f(\cdot)$ can deviate 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 the SSM is to infer the so-called filtering distribution $\pi(x_t|\boldsymbol{z}_{1:t})$ such that downstream statistical inference can be conducted for the dynamic system. In the study, it is often assumed that the model (1.1) contains no unknown parameters. Otherwise, the state-augmentation (Anderson, 2001) scheme can be employed to estimate the states and parameters in a simultaneous manner.

The model (1.1) has been studied in the literature for over half a century. The simplest case of the model is that $g(\cdot)$ is linear, $f(\cdot)$ is Gaussian with the mean can be represented as a linear function of x_t. In this case, the filtering distribution is still Gaussian, and the Kalman filter (Kalman, 1960) provides a recursive formula for analytically updating the mean and variance of the filtering distribution. For the nonlinear case where both the state propagator and the mean function of $f(\cdot)$ can be nonlinear, extensions of the Kalman filter, such as extended Kalman filter (Uhlmann, 1992) and unscented Kalman Filter (Julier and Uhlmann, 1997), have been developed via appropriate linearization of the system. Rather than directly calculating the mean and variance of the filtering distribution, the ensemble Kalman filter (EnKF) (Evensen, 1994) proposed to approximate the filtering distribution using particles, while the linearization technique developed in the extended Kalman filter is still applied. For high-dimensional problems, i.e., p is large, the ensemble size is typically much smaller than p, which leads to dimension reduction and computational feasibility compared to the Kalman filter (Shumway and Stoffer, 2006). In particular, the storage for the covariance matrix of the filtering distribution can be much reduced, and the high-dimensional particles can be efficiently generated via a forecast-analysis procedure. Since it was proposed, the EnKF has gained enormous popularity in atmospheric and oceanic sciences. However, as shown by Law et al. (2016), 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 for linear systems in the large sample limit. Similar results can be found in Le Gland et al. (2009), Bergou et al. (2019) and Kwiatkowski and Mandel (2015).

Other than the EnKF, the particle filter (Gordon et al., 1993), also known as sequential importance sampler, has been used to infer the filtering distribution for the model (1.1), which works for the general case of nonlinear and/or non-Gaussian data. However, it 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 (Cappé et al., 2004) under these scenarios. Moreover, the particle filter is not scalable with respect to the sample size N_t due to its Metropolis sampling nature, where a likelihood function needs to be evaluated with all available data when a particle is generated at each stage.

Quite recently, Zhang et al. (2023) proposed a Langevinized EnKF (LEnKF) algorithm for inference of the filtering distribution for the model (1.1) with $f(\cdot)$ being assumed to be Gaussian. LEnKF works by reformulating the EnKF under the framework of Langevin dynamics. It 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 the sample size. Moreover, LEnKF overcomes the sample degeneracy issue suffered by the particle filter as well as the filtering distribution estimation issue suffered by the EnKF. Under the big data scenario, LEnKF is shown to converge to the right filtering distribution in Wasserstein distance as t becomes large.

Towards filtering distribution estimation for the non-Gaussian case of the model (1.1), Katzfuss et al. (2020) proposed to use Markov chain Monte Carlo (MCMC), e.g., the Metropolis-Hasting algorithm (Metropolis et al., 1953; Hastings, 1970) and Gibbs sam-

pler (Geman and Geman, 1984), 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 data set.

In this paper, we extend LEnKF to non-Gaussian systems by introducing a latent Gaussian measurement variable to the model (1.1). The proposed algorithm can converge to the right filtering distribution as the number of stages becomes large, while inheriting the scalability of LEnKF with respect to the state dimension and the sample size. The proposed algorithm is applied to dynamic networks for sampling from the filtering distribution of the node embedding vectors, which facilitates the downstream statistical inference for the dynamic networks.

The remaining part of this paper is organized as follows. Section 2 describes the proposed algorithm with justification for its validity. Sections 3 illustrates the performance of the proposed algorithm using some simulated data sets. Section 4 applies the proposed algorithm to inference of large-scale dynamic networks. Section 5 concludes the paper with a brief discussion.

2 Extended Langevinized Ensemble Kalman Filter

2.1 A Brief Review of the LEnKF Algorithm

Motivated by the observation that the model (1.1) forms a Bayesian inverse problem at each stage, where the state evolution equation implies a prior distribution of x_t as shown later, let's start with a Bayesian inverse problem for the linear regression:

$$y = Hx + \eta, \tag{2.1}$$

where $\eta \sim \mathcal{N}_N(0, \Gamma)$ 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 $\{\tilde{y}_1, \ldots, \tilde{y}_B\}$, where each block is of size n and has the positive definite covariance matrix V such that $\Gamma = \text{diag}[V, \cdots, V]$.

Let $\pi(x)$ denote the prior density function of x, which is assumed to be differentiable with respect to x. Let $\pi(x|y)$ denote the posterior distribution. LEnKF provides a uniform framework for dealing the inverse and data assimilation problems. It reformulates the model (2.1) as a state-space model through subsampling and Langevin diffusion:

$$\begin{aligned} x_t &= x_{t-1} + \epsilon_t \frac{n}{2N} \nabla \log \pi(x_{t-1}) + w_t, \\ y_t &= H_t x_t + v_t, \end{aligned} \tag{2.2}$$

where ϵ_t denotes the learning rate used at stage t, $w_t \sim \mathcal{N}_p(0, \frac{n}{N} Q_t)$ with $Q_t = \epsilon_t I_p$, y_t denotes a block randomly drawn from $\{\tilde{y}_1, \ldots, \tilde{y}_B\}$, $v_t \sim \mathcal{N}_n(0, V_t)$ with $V_t = V$, and H_t

is a submatrix of H extracted with the corresponding vector y_t. In this state-space model, at each stage t, the state x_t evolves according to an Euler-discretized Langevin equation of the prior distribution, and the measurement varies with subsampling. To simulate from the dynamic system (2.2), LEnKF works as described in Algorithm A1 in the Appendix.

For the linear inverse problem, as shown in Zhang et al. (2023), LEnKF is mathematically equivalent to, but practically much more efficient than, a parallel preconditioned stochastic gradient MCMC algorithm for which each chain (represented by a member of the ensemble) iterates in the following equation:

$$x_t^a = x_{t-1}^a + \frac{\epsilon_t}{2}\Sigma_t \widehat{\nabla} \log \pi(x_{t-1}^a|y_t) + e_t, \tag{2.3}$$

where x_t^a denotes a generic member of the ensemble produced by Algorithm A1 in the analysis step of stage t, $\Sigma_t = \frac{n}{N}(I - K_t H_t)$ is the preconditioning matrix, e_t is a zero mean Gaussian random error with covariance $\mathrm{Var}(e_t) = \epsilon_t \Sigma_t$, and $\widehat{\nabla} \log \pi(x_{t-1}^a|y_t) = \frac{N}{n} H_t^T V_t^{-1}(y_t - H_t x_{t-1}^a) + \nabla \log \pi(x_{t-1}^a)$ denotes an unbiased estimate of $\nabla \log \pi(x_{t-1}^a|y_t)$.

LEnKF makes use of both techniques, subsampling and the forecast-analysis procedure. The former makes it scalable with respect to the sample size N, and the latter makes it scalable with respect to the dimension p. As shown in Zhang et al. (2023), for a high-dimensional problem with $n \ll p$, the computational complexity of LEnKF can be much lower than directly simulating from (2.3). The latter requires an LU-decomposition of Σ_t, which has a computational complexity of $O(p^3)$. While LEnKF gets around this issue with the forecast-analysis procedure, making it scalable with respect to the dimension of the state variable. By the analysis of Zhang et al. (2023), the computational complexity of LEnKF is actually the same as that of the parallel stochastic gradient Langevin dynamics (SGLD) algorithm Welling and Teh (2011), which consists of m chains and each chain evolves via the iteration

$$x_t^i = x_{t-1}^i + \frac{\epsilon_t}{2}\widehat{\nabla}_x \log \pi(x_{t-1}^i|y) + \tilde{e}_t, \ i = 1, 2, \ldots, m,$$

where $\tilde{e}_t \sim \mathcal{N}_p(0, \epsilon_t I_p)$, and all chains are updated based on the same mini-batch data y_t at each iteration t. However, as implied by Theorem 1 of Li et al. (2016), LEnKF can converge much faster than the parallel SGLD algorithm Welling and Teh (2011), since all eigenvalues of the preconditioning matrix Σ_t can be much less than 1 by noting that $\Sigma_t = \frac{n}{N}(I - K_t H_t) = \frac{n}{N}(I - \epsilon_t H_t^T(\epsilon_t H_t H_t^T + 2V_t)^{-1} H_t)$ under the assumption V_t is positive definite.

Zhang et al. (2023) has extended Algorithm A1 to the Gaussian dynamic system

$$\begin{aligned} x_t &= g(x_{t-1}) + u_t, \quad u_t \sim \mathcal{N}_p(0, U_t), \\ y_t &= H_t x_t + \eta_t, \quad \eta_t \sim \mathcal{N}_{n_t}(0, \Gamma_t), \end{aligned} \tag{2.4}$$

based on the Bayesian formula

$$\pi(x_t|\boldsymbol{y}_{1:t}) = \frac{f(y_t|x_t)\pi(x_t|\boldsymbol{y}_{1:t-1})}{\int f(y_t|x_t)\pi(x_t|\boldsymbol{y}_{1:t-1})dx_t}, \tag{2.5}$$

where $y_t \in \mathbb{R}^{N_t}$ denotes the samples at stage t, $\boldsymbol{y}_{1:t} = \{y_1, y_2, \ldots, y_t\}$ denotes the collection of the observations up to stage t. To ensure the particles simulated at stage t to converge to the filtering distribution $\pi(x_t|\boldsymbol{y}_{1:t})$, they employed the predictive distribution $\pi(x_t|\boldsymbol{y}_{1:t-1})$ as the prior according to the Bayesian formula. To estimate the gradient $\nabla \log \pi(x_t|\boldsymbol{y}_{1:t-1})$, they employed an importance resampling procedure, see Zhang et al. (2023) for the detail. To make the algorithm scalable with respect to big data, they employed a mini-batch strategy under the assumption that 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 $y_{t,k}$ is a block of n_t observations randomly drawn from $y_t = \{\tilde{y}_{t,1}, \ldots, \tilde{y}_{t,B_t}\}$, $v_{t,k} \sim \mathcal{N}_{n_t}(0, V_t)$ for all k, and $v_{t,k}$'s are mutually independent. Finally, they showed that the state x_t of the dynamic system (2.4) at stage t can be simulated by applying Algorithm A1 to simulate from the following dynamic system

$$\begin{aligned} x_{t,k} &= x_{t,k-1} - \epsilon_t \frac{n_t}{2N_t} U_t^{-1}(x_{t,k-1} - g(\tilde{x}_{t-1,k-1})) + w_{t,k}, \\ y_{t,k} &= H_{t,k}x_{t,k} + v_{t,k}, \end{aligned} \tag{2.6}$$

for $k = 1, 2, \ldots$, where $x_{t,0} = g(x_{t-1}) + u_t$; $\tilde{x}_{t-1,k-1}$ denotes a sample drawn from $\pi(x_{t-1}|\ x_{t,k-1},\ \boldsymbol{y}_{1:t-1})$ at iteration k of stage t through the importance resampling procedure; $w_{t,k} \sim \mathcal{N}_p(0, \frac{n_t}{N_t}Q_{t,k})$ with $Q_{t,k} = \epsilon_{t,k}I_p$, and p is the dimension of x_t. The gradient term $-U_t^{-1}(x_{t,k-1} - g(\tilde{x}_{t-1,k-1}))$ forms an unbiased estimator of $\nabla \log \pi(x_{t,k-1}|\boldsymbol{y}_{1:t-1})$ under the assumption that the samples in $\mathcal{X}_{t-1}$ follows from the distribution $\pi(x_{t-1}|y_{1:t-1})$ and the sample size $|\mathcal{X}_{t-1}|$ is sufficiently large. This assumption has often been used in theoretical analysis of sequential Monte Carlo. Possible deviations from this assumption has been taken into account in the theoretical study of Zhang et al. (2023). Refer to Zhang et al. (2023) for the detail.

Zhang et al. (2023) argued that LEnKF overcomes the sample degeneracy issue (Cappé et al., 2004) suffered by the particle filter, as the importance resampling procedure employed in LEnKF aims to draw a sample for the prior $\pi(x_t|\boldsymbol{y}_{1:t-1})$ at each stage t, while that of sequential importance sampling aims to draw a sample for the posterior $\pi(x_t|\boldsymbol{y}_{1:t})$. In consequence, LEnKF is less bothered by the sample degeneracy issue.

2.2 Extended LEnKF Algorithms for non-Gaussian Data

In practice, we often encounter dynamic systems where the measurement variable follows a non-Gaussian distribution, e.g., multinomial or Poisson. 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.

2.2.1 Inverse Problems

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

$$z \sim \psi(\cdot|y), \quad y = Hx + \eta, \quad \eta \sim \mathcal{N}_N(0, \Gamma), \tag{2.7}$$

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 $\pi(y|x,z) \propto \psi(z|y)\phi(y|x)$, where $\phi(\cdot)$ denotes a Gaussian density function.

To accommodate the case that N is extremely large, we assume that z can be partitioned into $B = N/n$ independent and identically distributed blocks $\{\tilde{z}_1, \ldots, \tilde{z}_B\}$, where each block is of size n and the corresponding latent variables have the positive definite covariance matrix V such that $\Gamma = \text{diag}[V, \cdots, V]$. To adapt LEnKF to simulating from the posterior distribution $\pi(x|z)$, we only need to add an imputation step in Algorithm A1. The extended algorithm is described in Algorithm 1.

Algorithm 1: 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**

(i) Subsampling: Draw a mini-batch sample, denoted by (z_t, H_t), of size n from the full data set of size N.

Set $Q_t = \epsilon_t I_p$, $R_t = 2V$, 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**

(ii) Forecast: Draw $w_t^i \sim \mathcal{N}_p(0, \frac{n}{N} Q_t)$ and calculate

$$x_t^{f,i} = x_{t-1}^{a,i} + \epsilon_t \frac{n}{2N} \nabla \log \pi(x_{t-1}^{a,i}) + w_t^i. \tag{2.8}$$

(iii) Imputation: Draw $y_t^i \sim \pi(y|x_t^{f,i}, z_t) \propto \psi(z_t|y)\phi(y|x_t^{f,i})$, where $y_t^i|x_t^{f,i} \sim \mathcal{N}_n(H_t x_t^{f,i}, V)$.

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

$$x_t^{a,i} = x_t^{f,i} + K_t(y_t^i - H_t x_t^{f,i} - v_t^i) \triangleq x_t^{f,i} + K_t(y_t^i - y_t^{f,i}). \tag{2.9}$$

end

end

2.2.2 Data Assimilation

For non-Gaussian data assimilation problems, the corresponding state space model is given by

$$\begin{aligned} x_t &= g(x_{t-1}) + u_t, \quad u_t \sim \mathcal{N}_p(0, U_t), \\ y_t &= H_t x_t + \eta_t, \quad \eta_t \sim \mathcal{N}_{N_t}(0, \Gamma_t), \\ z_t &\sim \psi(\cdot|y_t), \end{aligned} \tag{2.10}$$

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 $\pi(y_t|x_t, z_t) \propto \psi(z_t|y_t)\phi(y_t|x_t)$, and $\phi(\cdot)$ denotes a Gaussian density function. Then the data assimilation LEnKF algorithm of Zhang et al. (2023) can be extended to the model (2.10) by including an imputation step for the latent variable y_t at each stage t. To accommodate the case that N_t is extremely large at each stage t, we assume that z_t can be partitioned into $B_t = N_t/n_t$ independent and identically distributed blocks $\{\tilde{z}_{t,1}, \ldots, \tilde{z}_{t,B_t}\}$, where each block is of size n_t and the corresponding latent variable has the covariance matrix V_t such that $\Gamma_t = \text{diag}[V_t, \cdots, V_t]$. The resulting algorithm is described in Algorithm 2.

2.3 Convergence of the Extended LEnKF Algorithms

To justify the convergence of the extended LEnKF algorithms, we introduce the following identity established in Song et al. (2020):

$$\nabla_\theta \log \pi(\theta \mid D) = \int \nabla_\theta \log \pi(\theta \mid \vartheta, D)\pi(\vartheta \mid \theta, D)d\vartheta, \tag{2.14}$$

where D denotes data, and θ and ϑ denote two parameters of the posterior $\pi(\theta, \vartheta|D)$. With this identity, it is easy to show that

$$\nabla_x \log \pi(x|z_t) = \int \nabla_x \log \pi(x|y, z_t)\pi(y|x, z_t)dy = \int \nabla_x \log \pi(x|y)\pi(y|x, z_t)dy, \tag{2.15}$$

where the last equality holds as $\pi(x|y, z) = \pi(x|y)$ given the hierarchical structure (2.7). That is, $\nabla_x \log \pi(x|y)$ forms an unbiased estimator of $\nabla_x \log \pi(x|z_t)$, provided that $y \sim \pi(y|x, z_t)$. Further, by (2.3), we can show that Algorithm 1 leads to a preconditioned SGLD algorithm for simulating from the posterior distribution $\pi(x|z)$ even when z is not Gaussian. In summary, we have the following Lemma concerning the convergence of Algorithm 1.

Lemma 2.1. *(Convergence of the extended LEnKF for non-Gaussian inverse problems) Let x_t^a denote a generic member of the ensemble produced by Algorithm 1 in the analysis step of stage t. If V is positive definite, $\log \pi(x)$ is differentiable with respect to x, and the learning rate $\epsilon_t = O(t^{-\varpi})$ for some $0 < \varpi < 1$, then $\lim_{t\to\infty} W_2(\tilde{\pi}_t, \pi_*) = 0$, where $\tilde{\pi}_t$ denotes the empirical distribution of x_t^a, $\pi_* = \pi(x|z)$ denotes the target posterior distribution, and $W_2(\cdot, \cdot)$ denotes the second-order Wasserstein distance.*

The proof of Lemma 2.1 directly follows from (2.15) and the proof of Theorem 1 in Zhang et al. (2023) and is thus omitted. When an exact sampler for $\pi(y|x, z_t)$ is not available, the imputation step can be done by a short run of the Metropolis-Hastings algorithm. In this case, (2.9) can be replaced by

$$x_t^{a,i} = x_t^{f,i} + K_t\big(\bar{y}_t^i - H_t x_t^{f,i} - v_t^i\big), \tag{2.16}$$

where $\bar{y}_t^i = \frac{1}{r}\sum_{j=1}^r y_t^{i,j}$ and $y_t^{i,1}, \ldots, y_t^{i,r}$ are the samples simulated in a short Metropolis-Hastings run. The validity of the resulting algorithm can be justified by noting that $\pi(y|x)$

Algorithm 2: 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 $\pi(x_1)$, where m denotes the ensemble size. Set $\mathcal{X}_t = \emptyset$ for $t = 1, 2, \ldots, T$. Set k_0 as the common burnin period for each stage t.

for *t=1,2,...,T* **do**

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

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

$$x_{t,k}^{f,i} = x_{t,k-1}^{a,i} - \epsilon_{t,k} \frac{n_t}{2N_t} \nabla \log \pi(x_{t,k-1}^{a,i}) + w_{t,k}^i, \tag{2.11}$$

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

$$x_{t,k}^{f,i} = x_{t,k-1}^{a,i} - \epsilon_{t,k} \frac{n_t}{2N_t} U_t^{-1} (x_{t,k-1}^{a,i} - g(\tilde{x}_{t-1,k-1}^i)) + w_{t,k}^i. \tag{2.12}$$

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

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

$$x_{t,k}^{a,i} = x_{t,k}^{f,i} + K_{t,k}(y_{t,k}^i - H_{t,k} x_{t,k}^{f,i} - v_{t,k}^i) \triangleq x_{t,k}^{f,i} + K_{t,k}(y_{t,k}^i - y_{t,k}^{f,i}). \tag{2.13}$$

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

    end

  end

end

is Gaussian, which implies that $\nabla_x \log \pi(x|y)$ is a linear function of y. As a result, (2.16) leads to an asymptotically unbiased estimator of $\nabla_x \log \pi(x|z)$ given by $N/nH_t^T V_t^{-1}(\bar{y}_t - H_t x_{t-1}^a) + \nabla_x \log \pi(x_{t-1}^a)$, which is similar to the one given in (2.3). 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 S1 of Zhang et al. (2023), 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 $\limsup_{t\to\infty} W_2(\tilde{\pi}_t, \pi_*) = O(1/r)$ in this case. Similar results can be found in Song et al. (2020), Dalalyan and Karagulyan (2017) and Bhatia et al. (2019), where the convergence of the SGLD algorithm Welling and Teh (2011) was established with an inaccurate gradient. In general, the averaging estimator reduces the variation of the stochastic gradient and improves the convergence of the stochastic gradient MCMC algorithm, see Nemeth and Fearnhead (2019) for more discussions on this issue. Alternative to the averaging estimator, the last sample generated in a short run 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 2, we have the following lemma, whose proof directly follows from (2.15) and the proof of Theorem 2 in Zhang et al. (2023) and is thus omitted.

Lemma 2.2. *(Convergence of the 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 2 of Zhang et al. (2023). If $\epsilon_{t,k} \propto \frac{1}{n_t^2 \log \mathcal{K}} k^{-\varpi}$ for some $\varpi \in (0,1)$ and any $k \in \{1, 2, \ldots, \mathcal{K}\}$, then uniformly with dominating probability, for any $t \in \{1, 2, \ldots, T\}$, $x_{t,\mathcal{K}}^{a,i}$ follows a probability law $\tilde{\pi}_t$ and $\lim_{\mathcal{K}\to\infty} W_2(\tilde{\pi}_t, \pi_t) = 0$, where $\pi_t = \pi(x_t|z_{1:t})$ denotes the filtering distribution at stage t, and $W_2(\cdot,\cdot)$ denotes 2-Wasserstein distance between two distributions.*

3 Illustrative Examples

This section contains two examples, which are for Poisson regression and dynamic Poisson spatial model, respectively. The first example illustrates the application of Algorithm 1 for inverse problems, and the second example illustrates the application of Algorithm 2 for data assimilation problems. In the Appendix, we give an example on nonlinear inverse problems, which illustrates how the proposed algorithm can be extended to more complicated dynamic systems for which the expectation function of the Gaussian latent variable, i.e., $E(y_t|x_t)$, is a nonlinear function of x_t.

3.1 Poisson Regression

Consider a Poisson regression reformulated in the following equations:

$$\begin{aligned} y_i &= \boldsymbol{x}_i^T \boldsymbol{\beta} + \sigma_0 \epsilon_i, \\ z_i &\sim \mathcal{P}ois(\exp(y_i)), \end{aligned} \tag{3.1}$$

for $i = 1, 2, \ldots, N$, where $\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,p})^T$ is a p-dimensional vector of explanatory variables, $\boldsymbol{\beta}$ is a p-dimensional vector of regression coefficients, σ_0 is the standard deviation, and ϵ_i's are iid standard Gaussian random variables. With a slight abuse of notation, we use x_k, for $k = 1, 2, \ldots, p$, to denote the kth covariate of the regression.

We generated 10 data sets from (3.1) 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 $\beta_j = 0$ for $j = 9, \cdots, p$. We generated each vector $\boldsymbol{x}_i$ from a multivariate Gaussian distribution such that each of its component has a standard Gaussian marginal distribution and a correlation coefficient of 0.5 with any other component.

To conduct Bayesian analysis, we let each component of $\boldsymbol{\beta}$ be subject to a mixture Gaussian prior distribution

$$\alpha_1 \mathcal{N}(0, \tau_1^2) + (1 - \alpha_1)\mathcal{N}(0, \tau_0^2), \tag{3.2}$$

where $\alpha_1 = 1/p = 0.0005$, $\tau_1^2 = 1$ and $\tau_0^2 = 0.001$, and assume that all the components are *priori* independent. Algorithm 1 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 Metropolis-Hastings sampler (Metropolis et al., 1953; Hastings, 1970) 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 on a personal computer with RAM16GB and 2.9GHz Intel Core i7.

Figure 1 summarizes the variable selection results for one data set. The results for other data sets are similar. Figure 1(a) displays the sample paths of $\beta_1, \beta_2, \ldots, \beta_{2000}$ along with iterations, which indicate that they all have converged within 500 iterations. Specifically, after 500 iterations: the sample paths of $\beta_1, \ldots, \beta_4$ approach to the horizontal line at 1, the sample paths of $\beta_5, \ldots, \beta_8$ approach to the horizontal line at -1, and the sample paths of $\beta_9, \ldots, \beta_{2000}$ overlap around the horizontal line at 0. The plot also shows that the estimates of $\beta_1, \beta_2, \ldots, \beta_8$ are slightly biased, which might be due to the shrinkage prior we used. Figure 1(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 Barbieri and Berger (2004), the true model can be correctly identified. In summary, this example shows that the extended LEnKF can be applied to non-Gaussian inverse problems.

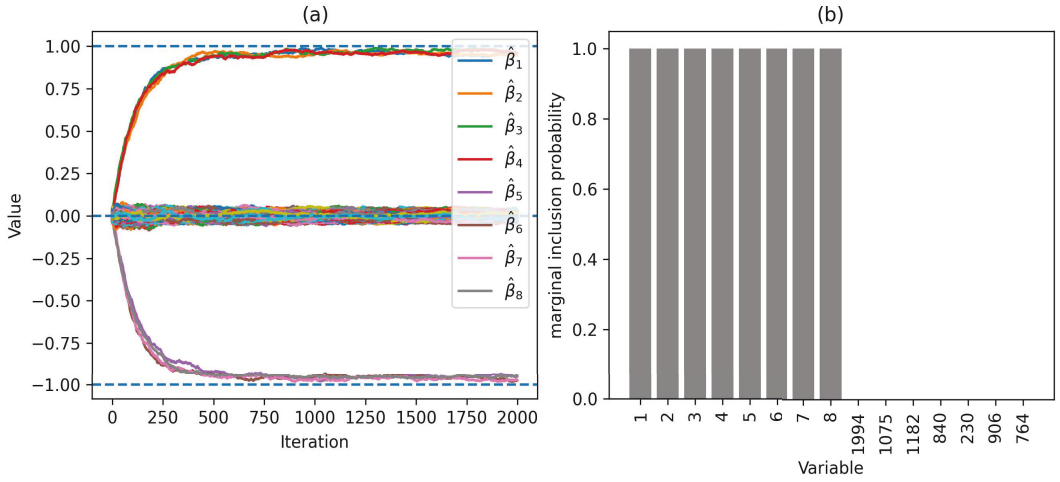


Figure 1: Langevinized EnKF for large-scale generalized linear regression: (a) Sample paths of $\beta_1, \beta_2, \ldots, \beta_{2000}$ across iterations, where $\hat{\beta}_i$, for each $i \in \{1, 2, \ldots, 2000\}$, was calculated by averaging over the ensemble at each iteration; (b) marginal inclusion probabilities of all covariates $x_1, \ldots, x_{2000}$, where the covariates are shown in the rank of marginal inclusion probabilities.

For this example, we set the prior hyperparameter α_1 at the level of $1/p$, following the suggestion from Narisetty and He (2014). This setting is robust for high-dimensional Bayesian variable selection, especially in the big data scenario of this example where the total sample size N is much larger than p.

3.2 A Nonlinear Non-Gaussian Example

Consider a nonlinear logistic regression reformulated in the following equations:

$$\begin{aligned} y_i &= 0x_{i,0} + \frac{10x_{i,1}^2}{1+x_{i,2}^2} + 5\sin(x_{i,3}x_{i,4}) + x_{i,5} + 0x_{i,6} + \cdots + 0x_{i,99} + \epsilon_i, \\ z_i &\sim \psi\left(\cdot|y_i\right) = \mathcal{B}ernoulli(1/1 + \exp(-y_i)), \end{aligned} \tag{3.3}$$

where $i \in \{1, 2, \cdots, N\}$ indexes the observations, $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, and $x_{i,1}$, $x_{i,2}$, …, $x_{i,99}$ 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 half of z_i's are 1 and the other half are 0.

Suppose that the state propagator in (3.3) 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 (Zhang et al., 2023), we can rewrite the nonlinear logistic regression model as

$$\begin{aligned} \boldsymbol{y}_t^{(1)} &= g(\boldsymbol{x}_t, \boldsymbol{\beta}) + \boldsymbol{\varepsilon}_t^{(1)}, \quad \boldsymbol{\theta}_t^T = (\boldsymbol{\beta}^T, (\boldsymbol{y}_t^{(1)})^T), \quad \boldsymbol{\varepsilon}_t^{(1)} \sim \mathcal{N}_n(0, \alpha\sigma^2 I_n), && (3.4) \\ \boldsymbol{y}_t &= H_t\boldsymbol{\theta}_t + \boldsymbol{\varepsilon}_t^{(2)} = \boldsymbol{y}_t^{(1)} + \boldsymbol{\varepsilon}_t^{(2)}, \quad \boldsymbol{\varepsilon}_t^{(2)} \sim \mathcal{N}_n(0, (1-\alpha)\sigma^2 I_n), && (3.5) \\ \boldsymbol{z}_t &\sim \psi(\cdot|\boldsymbol{y}_t), && (3.6) \end{aligned}$$

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 $\pi(\boldsymbol{\theta}_t) = \pi(\boldsymbol{\beta})\pi(\boldsymbol{y}_t^{(1)}|\boldsymbol{\beta})$, and $\pi(\boldsymbol{\beta})$ denotes the prior density function of $\boldsymbol{\beta}$. In this paper, we assume that the components of $\boldsymbol{\beta}$ are *a priori* independent and each follows a mixture Gaussian distribution:

$$\alpha_1\mathcal{N}(0, \tau_1^2) + (1-\alpha_1)\mathcal{N}(0, \tau_0^2), \tag{3.7}$$

where we set $\alpha_1 = 0.01$, $\tau_1^2 = 1$ and $\tau_0^2 = 0.05$ for this example.

Algorithm 1 was applied to solve the linear inverse problem formed by (3.5)-(3.6), 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 $k = 1, \cdots, \mathcal{K}$.

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

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

where $\psi(\cdot)$ is a Bernoulli distribution, and $\boldsymbol{y}|\boldsymbol{\beta}_{t-1,\mathcal{K}}^{a,i} \sim \mathcal{N}_n(g(\boldsymbol{x}_t, \boldsymbol{\beta}_{t-1,\mathcal{K}}^{a,i}), \alpha\sigma^2 I_n)$.

In the imputation step of iteration t, we drew $\boldsymbol{y}_{t,k}^i \sim \pi(\boldsymbol{y}|\boldsymbol{\theta}_{t,k}^{f,i}, \boldsymbol{z}_t) \propto \psi(\boldsymbol{z}_t|\boldsymbol{y})\pi(\boldsymbol{y}|\boldsymbol{\theta}_{t,k}^{f,i})$, where $y_{t,k}^i|\boldsymbol{\theta}_{t,k}^{f,i} \sim \mathcal{N}_n(H_t\boldsymbol{\theta}_{t,k}^{f,i}, (1-\alpha)\sigma^2 I_n)$, and $\boldsymbol{z}_{t,l}|\boldsymbol{y} \sim \mathcal{B}ernoulli(\exp(y_l))$ for $l = 1, \cdots, n$.

Figure 2 summarizes the results for one dataset. The results for other datasets are similar. Figure 2(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 2(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 2(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 $Y = 0$.

Table 1 summarizes the fitting and prediction results of the extended LEnKF over 10 simulated data sets in five metrics, namely, accuracy, precision, recall, F1 score and AUC value (i.e., the area under the ROC curve), for which the average values over 10 data sets were reported with the standard deviation given in the parentheses. Here the ROC curve refers to the receiver operating characteristic curve, which graphs the true positive rate (TPR) against the false positive rate (FPR) at various threshold values. The ROC curve

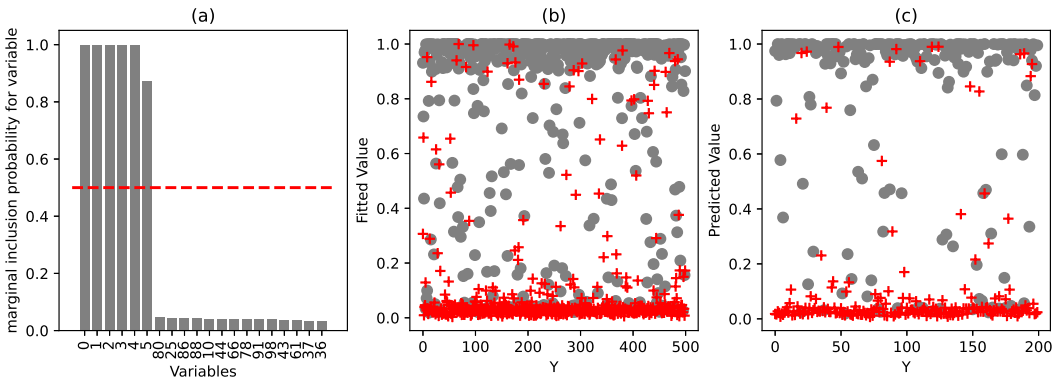


Figure 2: Extended Langevinized EnKF for a nonlinear classification example: (a) marginal inclusion probabilities of the variables, where the variables are shown in the rank of marginal inclusion probabilities; and (b) fitted value of Y (grey dot for true Y = 1, red cross sign for true Y = 0, randomly 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).

Table 1: Fitting and prediction results produced by the extended LEnKF for 10 simulated data sets, where the averages of the respective quantities, along with their standard deviations (given in the parentheses), are reported. The average CPU time for each run is 2630 seconds on a computer with Apple M1 chip and 64GB memory.

	Accuracy	Precision	Recall	F1 score	AUC
Fitting	0.8479 (0.0054)	0.8858 (0.0096)	0.8316 (0.0160)	0.8563 (0.0061)	0.9278 (0.0038)
Prediction	0.8659 (0.0035)	0.8843 (0.0116)	0.8582 (0.0120)	0.8704 (0.0024)	0.9458 (0.0038)

has often been used to assess the performance of a classification model, with the AUC value serving as a global measure for its classification ability. Refer to the Appendix for the definitions of the above metrics, and refer to Fawcett (2006) for more discussions about the use of ROC curves.

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.

3.3 Dynamic Poisson Spatial Model

We illustrate the performance of Algorithm 2 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:

$$\begin{aligned} \boldsymbol{\beta}_t|\boldsymbol{\beta}_{t-1} &\sim \mathcal{N}_p(\mathbf{M}(\boldsymbol{\gamma})\boldsymbol{\beta}_{t-1}, \mathbf{Q}(\tau^2, \lambda)), \\ \boldsymbol{y}_t|\boldsymbol{\beta}_t &\sim \mathcal{N}_{n_t}(\mathbf{H}_t\boldsymbol{\beta}_t, \sigma^2\mathbf{I}_{n_t}), \\ z_{t,l}|\boldsymbol{y}_t &\sim \mathcal{P}ois(\exp(y_{t,l})), \quad l = 1, \ldots, n_t, \end{aligned} \tag{3.8}$$

where $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \gamma_3)$, $\mathbf{M}(\boldsymbol{\gamma})$ is a tri-diagonal matrix with γ_1 on the main diagonal, γ_2 on the first upper sub-diagonal, and γ_3 on the first lower sub-diagonal. The state evolution error was assumed to exhibit spatial dependence in a Matern covariance matrix such that the (i, j)-th element of $\mathbf{Q}$ is given by $\mathbf{Q}_{i,j} = \tau^2 \,\mathrm{Mat}\left(|i - j|/\lambda\right)$, where $\mathrm{Mat}(d) = (1 + \sqrt{3}d)\exp(-\sqrt{3}d)$.

Ten data sets were simulated from the model (3.8). 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 $\boldsymbol{\beta}_0 \sim \mathcal{N}_p(\boldsymbol{\mu_0}, \boldsymbol{\Sigma}_0)$, where $\boldsymbol{\mu}_0$ is a constant vector of -2, and $(\boldsymbol{\Sigma}_0)_{i,j} = 0.2\,\mathrm{Mat}(|i - j|/5)$. Figure 3 shows the state values at 300 locations for $t = 1, 2, \ldots, 80$, whose chaotic behavior implies the challenge of the problem.

Algorithm 2 was applied to the data sets to re-estimate the states $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_T$. 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 $\mathcal{K}/2$ iterations, and the accuracy of the estimate was measured by the root mean-squared error (RMSE):

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

where $\hat{\boldsymbol{\beta}}_t$ denotes the estimate of $\boldsymbol{\beta}_t$. For comparison, the MCMC-EnKF algorithm Katzfuss et al. (2020) (see Algorithm A2 in the Appendix for the detail) was applied to this example,

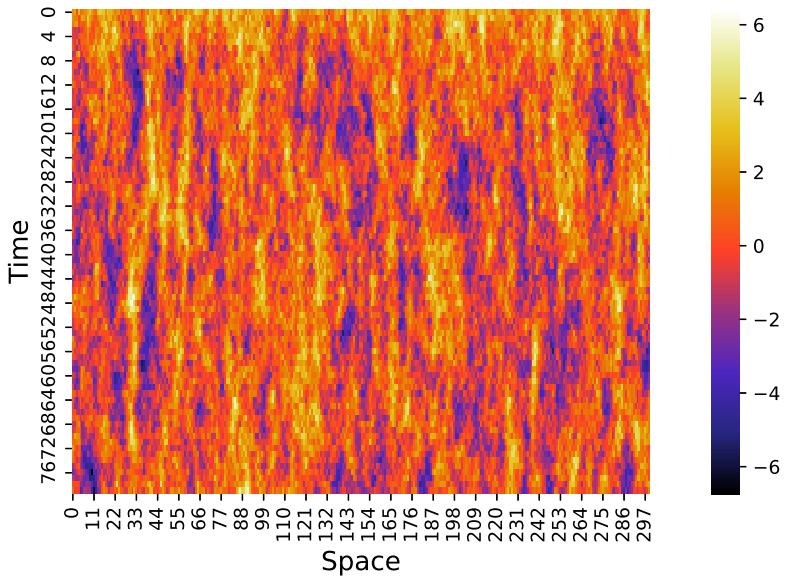


Figure 3: State values for $t = 1, 2, \ldots, T$

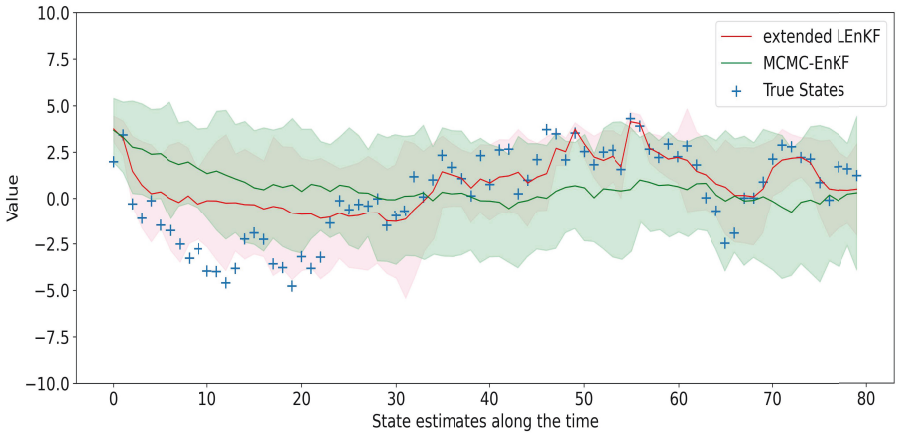
where the imputation was done as for the extended LEnKF, but the states were estimated using the EnKF. The ensemble size was set to $m = 20$ as well.

Figure 4 compares the estimates of two randomly selected components of $\boldsymbol{\beta}_t$ produced by LEnKF and 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 4(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 $\boldsymbol{\beta}_t$, while MCMC-EnKF cannot.

Figure 5 (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 5(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 5(c) shows that LEnKF produced smaller values of RMSE_t's than MCMC-EnKF.

Table 2 summarizes the numerical results of 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 LEnKF. In the table, we used $\text{MRMSE}(\boldsymbol{\beta})$ to denote the mean of the root-mean-squared-error of the esti-

(a)



(b)

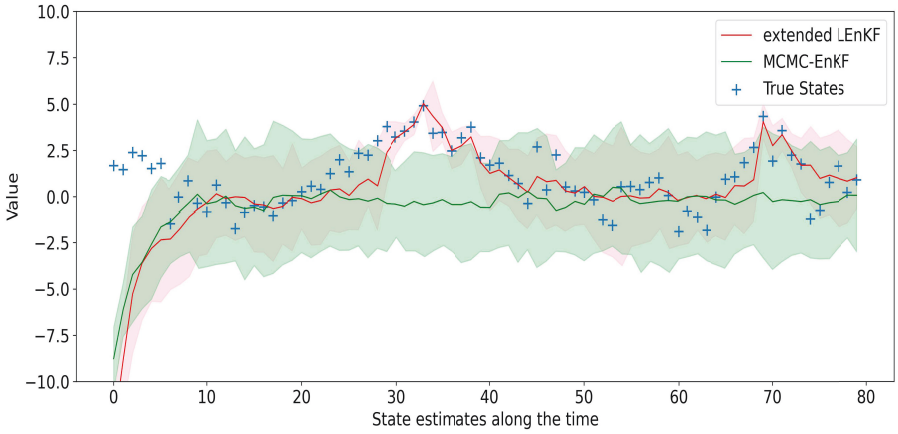


Figure 4: State estimates produced by LEnKF and MCMC-EnKF for a simulated cloud-motion data set along with stages $t = 1, 2, \ldots, 80$: each plot corresponds to one randomly selected component of $\boldsymbol{\beta}_t$, where the true state values are represented by '+', the estimates by LEnKF are represented by red lines, the estimates by MCMC-EnKF are represented by green lines, and their 95% confidence intervals are represented by shaded bands.

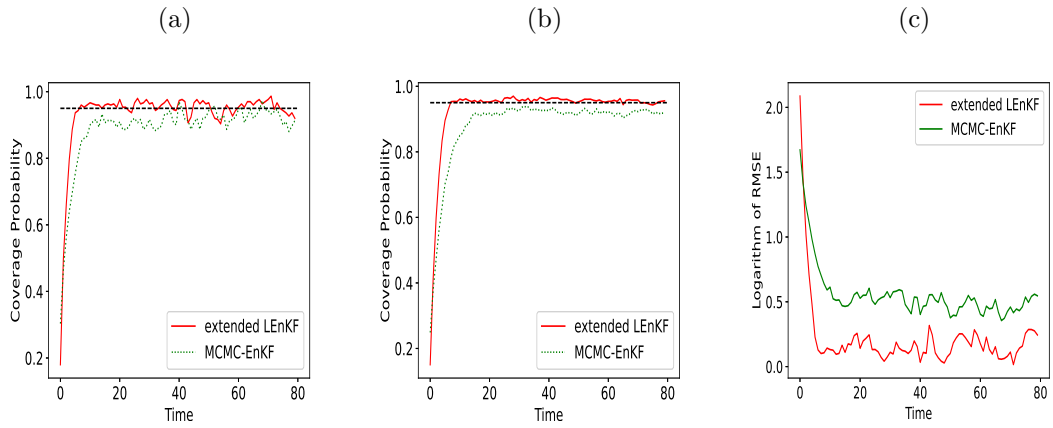


Figure 5: Coverage rates of the 95% confidence intervals produced by the extended LEnKF and MCMC-EnKF for the stages $t = 1, 2, \ldots, 80$: (a) coverage rates with one data set; (b) coverage rates averaged over 10 data sets; (c) $\log(\text{RMSE}_t)$ along with stage t.

mates for the states $\boldsymbol{\beta}_{30}, \boldsymbol{\beta}_{31}, \ldots, \boldsymbol{\beta}_T$ calculated on one data set, and used ave-MRMSE($\boldsymbol{\beta}$) to denote the average of MRMSE($\boldsymbol{\beta}$) over 10 data sets. Similarly, we used MCP to denote the coverage probability averaged over states $\boldsymbol{\beta}_{30}, \boldsymbol{\beta}_{31}, \ldots, \boldsymbol{\beta}_T$ 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, the extended LEnKF works well with both $k_0 = \mathcal{K}/2$ and $k_0 = \mathcal{K} - 1$, and it significantly outperforms MCMC-EnKF. For this example, MCMC-EnKF is faster than the extended LEnKF as it performs only one iteration at each stage, while the extended LEnKF involves $\mathcal{K}$ iterations without subsampling being performed. Note that each iteration of the extended LEnKF requires the inversion of an $n_t \times n_t$ matrix, while MCMC-EnKF deals with an $N_t \times N_t$ matrix. Since matrix inversion has a computational complexity of

Table 2: Comparison of LEnKF and MCMC-EnKF in state estimation for the dynamic spatial model, where the averages of the respective quantities over 10 data sets, along with their standard deviations (given in the parentheses), were reported. The average CPU time (in seconds) was recorded on a personal computer with RAM16GB and 2.9GHz Intel Core i7 for a single run of the algorithm.

	k_0	Ave-MRMSE($\boldsymbol{\beta}$)	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)

cubic order in relation to its size, the extended LEnKF can easily outperform MCMC-EnKF in terms of computation when subsampling is applied.

4 Dynamic Network Embedding

Dynamic networks have been studied in a variety of fields, such as social network analysis, recommendation systems and epidemiology. Embedding (or latent representation learning) has been recognized as one of the most promising approaches for dynamic network analysis. Its 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 for dynamic network embedding can be roughly grouped into two categories, namely, latent space model-based methods and dynamic graph neural network-based methods. Refer to Kim et al. (2018), Kazemi et al. (2020) and Skarding et al. (2021) 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 embedding vector. An exception is Sewell and Chen (2015), where the embedding vector 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 embedding vector for dynamic networks. More precisely, we employ the extended LEnKF algorithm to sample from the filtering distributions of the embedding vector, 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.

4.1 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 $\boldsymbol{x}_{it} = (x_{d(i-1)+1,t}, \ldots, x_{di,t})$ represents the 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 embedding vector at time $t+1$ only depends on the embedding vector at time t, which is the standard Markov assumption.

Dynamic Social Network Latent (DSNL) Model The first latent space model we considered in this paper is the DSNL model developed in Sarkar and Moore (2005). In the

form of state space models with the subsampling scheme, the DSNL model is given by

$$\begin{aligned}
&\boldsymbol{x}_t|\boldsymbol{x}_{t-1} \sim \mathcal{N}_{dN_t}(\boldsymbol{x}_{t-1}, \sigma_1^2 I_{dN_t}),\\
&\boldsymbol{y}_t|\boldsymbol{x}_t \sim \mathcal{N}_{dn_t}(H_t\boldsymbol{x}_t, \sigma_2^2 I_{dn_t}),\\
&P(\boldsymbol{z}_t|\boldsymbol{y}_t) = \prod_{i\sim j, i,j\in S_t} p_{ijt} \prod_{i\not\sim j, i,j\in S_t} (1-p_{ijt}),
\end{aligned} \tag{4.1}$$

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_{ijt} is the probability of a link at stage t. Specifically, p_{ijt} is defined by

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

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

Dynamic Latent Distance (DLD) Model Another latent space model we considered in the paper is the DLD model developed in Sewell and Chen (2015). In the form of state space models with the subsampling scheme, the DLD model is given by

$$\begin{aligned}
\boldsymbol{x}_t|\boldsymbol{x}_{t-1} &\sim \mathcal{N}_{dN_t}(\boldsymbol{x}_{t-1}, \sigma_1^2 I_{dN_t}),\\
\boldsymbol{y}_t|\boldsymbol{x}_t &\sim \mathcal{N}_{dn_t}(H_t\boldsymbol{x}_t, \sigma_2^2 I_{dn_t}),\\
P(\boldsymbol{z}_t|\boldsymbol{y}_t) &= \prod_{i\neq j, i,j\in S_t} \frac{\exp(z_{ijt}\eta_{ijt})}{1+\exp(\eta_{ijt})},\\
\eta_{ijt} &:= \log\left(\frac{\mathbb{P}(z_{ijt}=1|\boldsymbol{y}_t)}{\mathbb{P}(z_{ijt}=0|\boldsymbol{y}_t)}\right) = \beta_{in}\left(1-\frac{d_{ijt}}{r_{jt}}\right) + \beta_{out}\left(1-\frac{d_{ijt}}{r_{it}}\right),
\end{aligned} \tag{4.2}$$

where S_t, H_t, d_{ijt}, r_{it} and r_{jt} are as defined in (4.1); 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 paper, we consider undirected networks only.

4.2 A Dynamic Social Network

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 shows the dynamic network on six selected days. Our study delves into modeling the dynamics of College Messaging networks, scrutinizing their structural variations over time..

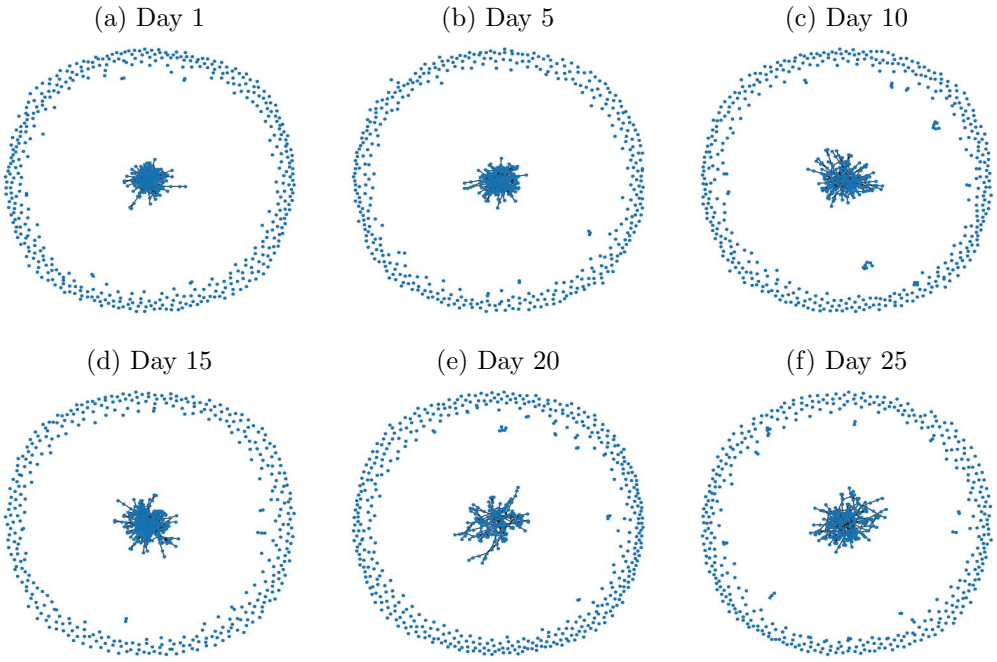


Figure 6: Dynamics of College Messaging networks on six selected days — Days 1, 5, 10, 15, 20, and 25, scrutinizing their structural variations over time.

We modeled the college messaging network by the DSNL and DLD models and trained the models using Algorithm 2. For the DSNL model, we set the constant noise parameter $\rho = 0.001$, the ensemble size $m = 20$, the standard deviations $\sigma_1 = \sigma_2 = 0.1$, the latent space dimension $d = 5$, the sample size $n_t = 250$, 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 the Metropolis-Hastings algorithm for imputation, where a Gaussian random walk

Table 3: 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 on a personal computer with RAM16GB and 2.9GHz Intel Core i7 for a single run of each algorithm.

Algorithm	Model	Avg-AUC	CPU Time(seconds)
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

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. Recall that β_{in} and β_{out} serve as measurements for the global popularity and activity effects of the network, respectively.

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 3 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.

To show that the extended LEnKF algorithm can be used for uncertainty quantification for dynamic networks, we plotted in Figure 7 and Figure 8 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 5, they might be biased in terms of inference for other examples. Moreover, as implied by Table 3,

MCMC-EnKF is much more costly than the extended LEnKF.

5 Conclusion

This paper has extended the LEnKF algorithm to non-Gaussian systems by introducing a latent Gaussian measurement variable to the state space model. The proposed algorithm converges to the right filtering distribution as the number of stages becomes large, while inheriting the scalability of LEnKF with respect to the state dimension and sample size. The proposed algorithm can be applied to sample from the filtering distribution of the embedding vector and thus downstream statistical inference for the dynamic networks.

The extended LEnKF can be further extended in many different ways. In methodology development, it can be easily combined with the state-augmentation method (Anderson, 2001) to simultaneously estimate the states and parameters of the dynamic system. With this combined method, for example, the parameters β_{in} and β_{out} of the DLD model can be adaptively determined instead of being specified as constants. In applications, LEnKF can be easily applied to learn the latent representation for a static network by treating it as an inverse problem. A further comparison of this method with the existing methods, see Kazemi et al. (2020) for a survey, is of interest. Other than the latent space models, the extended LEnKF algorithm can also be used with the dynamic graph neural networks to learn embedding vectors for dynamic networks, where, as shown in Zhang et al. (2023), the recurrent neural network or long-short-term-memory (LSTM) network can be easily expressed as a state space model.

Acknowledgments

Liang's research is support in part by the NSF grants DMS-2015498 and DMS-2210819, and the NIH grant R01-GM126089. The authors thank the editor, associate editor, and two referees for their constructive comments, which have led to significant improvement of this paper.

Appendix

A Metrics for binary classifiers

Consider an experiment with P positive instances and N negative instances under certain conditions. The four possible outcomes of a binary classifier can be represented in Table A1.

The metrics used in the paper are defined as follows:

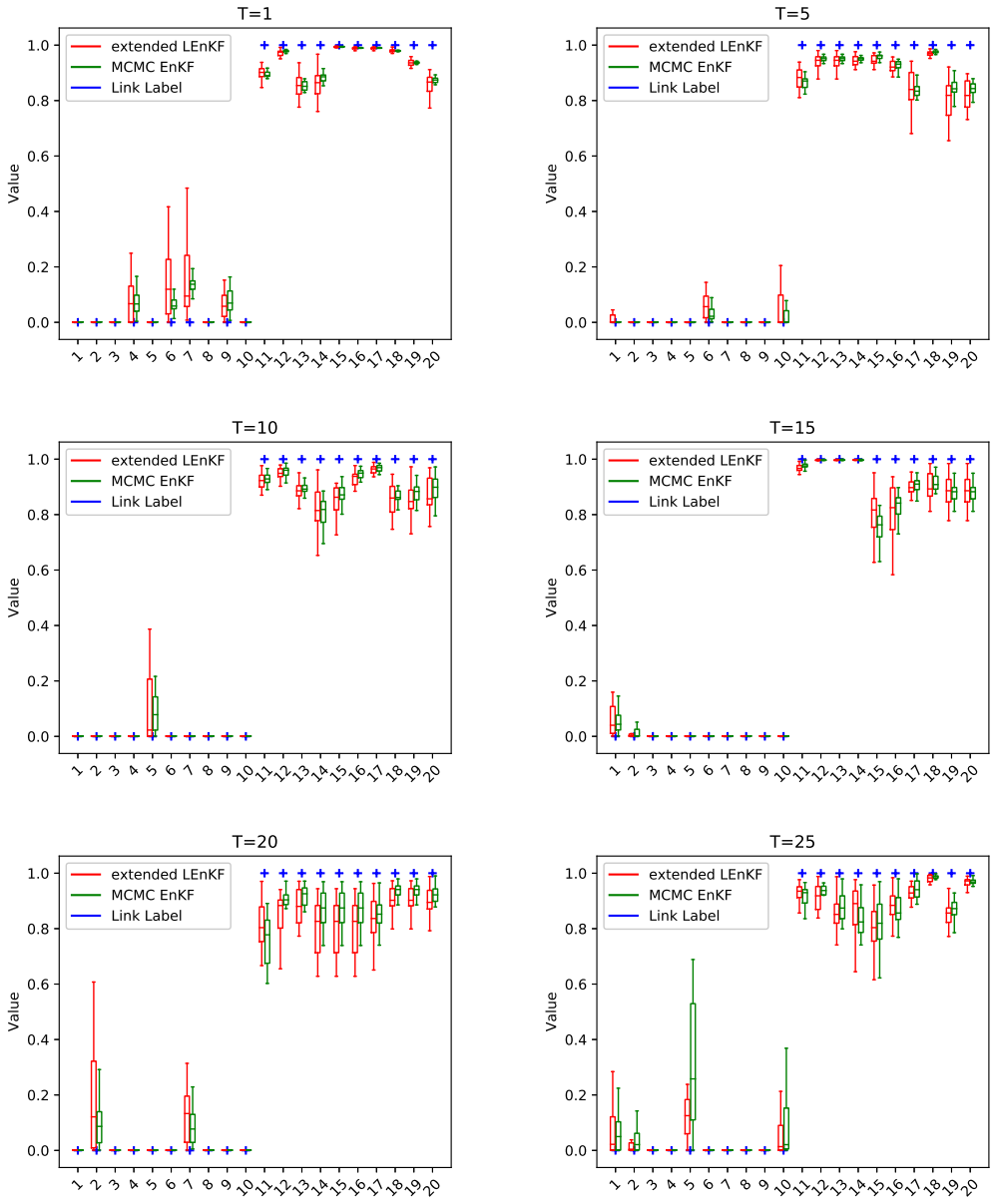


Figure 7: Box-plots of link probabilities produced with the DSNL model: fitted link probabilities for 10 pairs of nodes with edges and 10 pairs of nodes without edges are plotted for day 1, 5, 10, 15, 20 and 25.

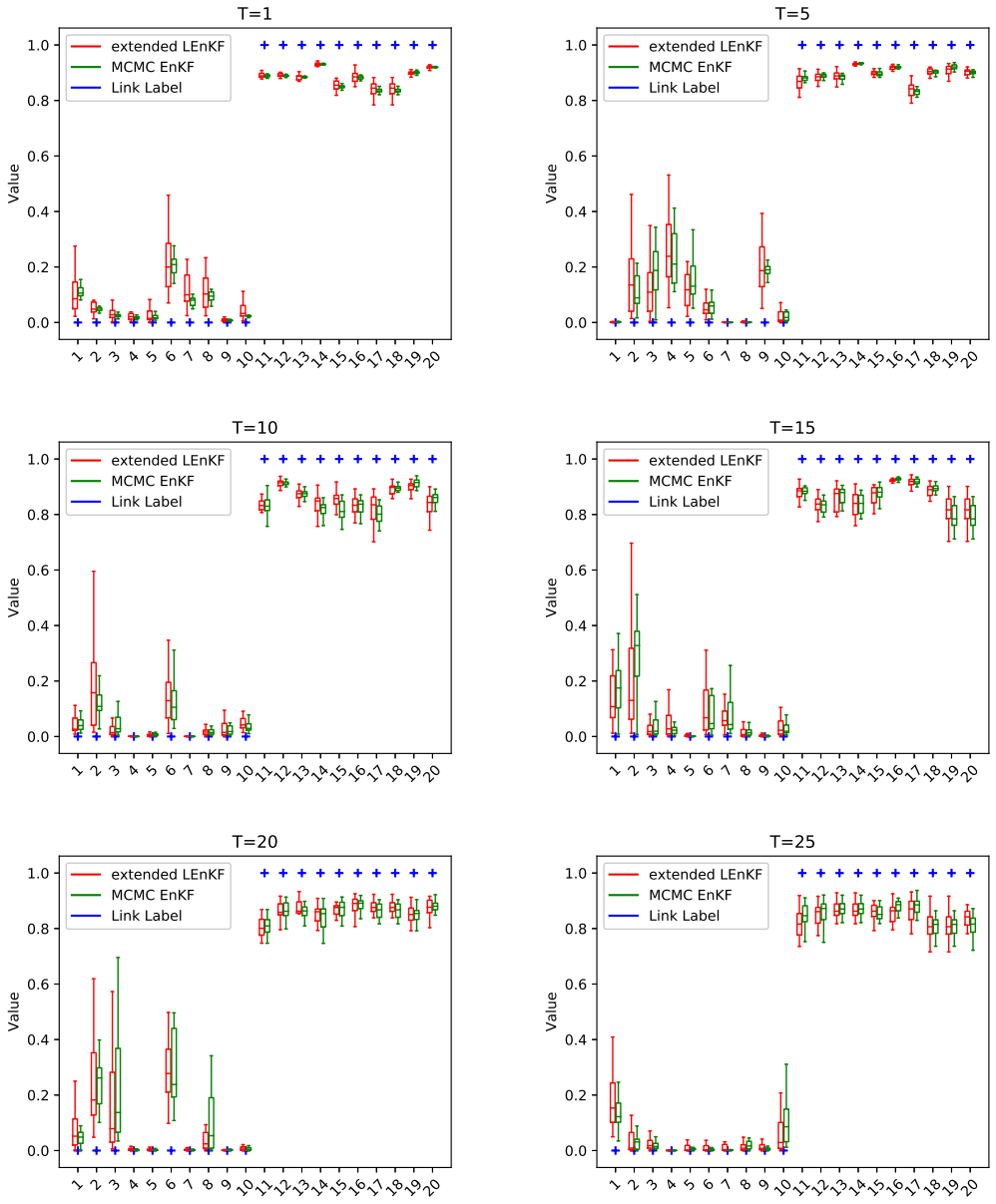


Figure 8: Box-plots of link probabilities produced with the DLD model: fitted link probabilities for 10 pairs of nodes with edges and 10 pairs of nodes without edges are plotted for day 1, 5, 10, 15, 20 and 25.

Table A1: Outcomes of a classifier

		Classified Values	
		P	N
True values	P	True Positive (TP)	False Negative (FN)
	N	False Positive (FP)	True Negative (TN)

- Accuracy=$\frac{TP+TN}{TP+TN+FP+FN}$;
- Precision=$\frac{TP}{TP+FP}$;
- Recall=$\frac{TP}{TP+FN}$, which is also known as sensitivity or true positive rate (TPR);
- Specificity=$\frac{TN}{TN+FP}$, which is also known as true negative rate (TNR);
- False positive rate (FPR)=1−Specificity;
- F1 score=$2\frac{\text{presision}\times\text{recall}}{\text{precision}+\text{recall}}$.

B Two Existing Algorithms

This section presents two existing algorithms, namely, the LEnKF algorithm (Zhang et al., 2023) for Gaussian linear inverse problems and the MCMC-EnKF algorithm (Katzfuss et al., 2020) for data assimilation problems, which are described in Algorithm A1 and Algorithm A2, respectively.

Algorithm A1: LEnKF for Linear Inverse Problems (Zhang et al., 2023)

(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 $w_t^i \sim \mathcal{N}_p(0, \frac{n}{N} Q_t)$ and calculate

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

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

$$x_t^{a,i} = x_t^{f,i} + K_t(y_t - H_t x_t^{f,i} - v_t^i) \triangleq x_t^{f,i} + K_t(y_t - y_t^{f,i}). \tag{B.2}$$

end

end

Algorithm A2: MCMC-EnKF for Data Assimilation (Katzfuss et al., 2020)

(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 $\pi(\boldsymbol{x}_0)$, where m denotes the ensemble size.

for *t=1,2,...,T* **do**

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

$$\boldsymbol{x}_t^{f,i} = g(\boldsymbol{x}_{t-1}^{a,i}) + w_t^i, \tag{B.3}$$

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 $\boldsymbol{y}_t^i \sim \pi(\boldsymbol{y}|\boldsymbol{x}_t^{f,i}, \boldsymbol{z}_t) \propto \rho(\boldsymbol{z}_t|\boldsymbol{y}) f(\boldsymbol{y}|\boldsymbol{x}_t^{f,i})$, where $\boldsymbol{y}_t^i|\boldsymbol{x}_t^{f,i} \sim \mathcal{N}_{N_t}(H_t \boldsymbol{x}_t^{f,i}, V_t)$, and H_t is an $N_t \times p$-dimensional matrix.

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

$$\boldsymbol{x}_t^{a,i} = \boldsymbol{x}_t^{f,i} + \hat{K}_t(\boldsymbol{y}_t^i - H_t \boldsymbol{x}_t^{f,i} - v_t^i) \triangleq \boldsymbol{x}_t^{f,i} + \hat{K}_t(\boldsymbol{y}_t^i - \boldsymbol{y}_t^{f,i}), \tag{B.4}$$

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 $\boldsymbol{x}_t$

end

References

Anderson, J. (2001), "An ensemble adjustment filter for data assimilation," *Monthly Weather Review*, 129, 2884–2903.

Barbieri, M. and Berger, J. (2004), "Optimal predictive model selection," *Annals of Statistics*, 32, 870–897.

Bergou, E., Gratton, S., and Mandel, J. (2019), "On the convergence of a non-linear ensemble Kalman smoother," *Applied Numerical Mathematics*, 137, 151–168.

Bhatia, K., Ma, Y.-A., Dragan, A. D., Bartlett, P. L., and Jordan, M. I. (2019), "Bayesian Robustness: A Nonasymptotic Viewpoint," *arXiv preprint arXiv:1907.11826*.

Cappé, O., Guillin, A., Martin, J., and Robert, C. (2004), "Population Monte Carlo," *Journal of Computational and Graphical Statistics*, 13, 907–929.

Dalalyan, A. S. and Karagulyan, A. G. (2017), "User-friendly guarantees for the Langevin Monte Carlo with inaccurate gradient," *CoRR*, abs/1710.00095.

Evensen, G. (1994), "Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics," *J. Geophys. Res.*, 99, 10143–10162.

Fawcett, T. (2006), "An introduction to ROC analysis," *Pattern Recognit. Lett.*, 27, 861–874.

Geman, S. and Geman, D. (1984), "Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6, 721–741.

Gordon, N., Salmond, D., and Smith, A. (1993), "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proceedings F - Radar and Signal Processing*, 140, 107–113.

Hastings, W. (1970), "Monte Carlo sampling methods using Markov chain and their applications," *Biometrika*, 57, 97–109.

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

Kalman, R. (1960), "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, 82, 35–45.

Katzfuss, M., Stroud, J. R., and Wikle, C. K. (2020), "Ensemble Kalman Methods for High-Dimensional Hierarchical Dynamic Space-Time Models," *Journal of the American Statistical Association*, 115, 866–885.

Kazemi, S. M., Goel, R., Jain, K., Kobyzev, I., Sethi, A., Forsyth, P., and Poupart, P. (2020), "Representation Learning for Dynamic Graphs: A Survey," *Journal of Machine Learning Research*, 21, 1–73.

Kim, B., Lee, K., Xue, L., and Niu, X. (2018), "A Review of Dynamic Network Models with Latent Variables," *Statistics Surveys*, 12, 105–135.

Kwiatkowski, E. and Mandel, J. (2015), "Convergence of the square root ensemble Kalman filter in the large ensemble limit," *SIAM/ASA J. Uncertainty Quantification*, 3, 1–17.

Law, K., Tembine, H., and Tempone, R. (2016), "Deterministic Mean-Field Ensemble Kalman Filtering," *SIAM J. Sci. Comput.*, 38, A1251–A1279.

Le Gland, F., Monbet, V., and Tran, V.-D. (2009), "Large sample asymptotics for the ensemble Kalman filter," *Research report RR-7014, INRIA*.

Li, C., Chen, C., Carlson, D. E., and Carin, L. (2016), "Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks," in *AAAI*, pp. 1788–1794.

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

Narisetty, N. N. and He, X. (2014), "Bayesian variable selection with shrinking and diffusing priors," *Annals of Statistics*, 42, 789–817.

Nemeth, C. and Fearnhead, P. (2019), "Stochastic Gradient Markov Chain Monte Carlo," *arXiv:1907.06986*.

Sarkar, P. and Moore, A. (2005), "Dynamic social network analysis using latent space models," *ACM SIGKDD Explorations Newletter*, 7, 31–40.

Sewell, D. K. and Chen, Y. (2015), "Latent space models for dynamic networks," *Journal of the American Statistical Association*, 110, 1646–1657.

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

Skarding, J., Gabrys, B., and Musial, K. (2021), "Foundations and Modeling of Dynamic Networks Using Dynamic Graph Neural Networks: A Survey," *IEEE Access*, 9, 79143–79168.

Song, Q., Sun, Y., Ye, M., and Liang, F. (2020), "Extended stochastic gradient Markov chain Monte Carlo for large-scale Bayesian variable selection," *Biometrika*, 107, 997–1004.

Uhlmann, J. K. (1992), "Algorithms for Multiple-Target Tracking," *American Scientist*, 80, 128–141.

Welling, M. and Teh, Y. W. (2011), "Bayesian Learning via Stochastic Gradient Langevin Dynamics," in *ICML*, pp. 681–688.

Zhang, P., Song, Q., and Liang, F. (2023), "A Langevinized Ensemble Kalman Filter for Large-Scale Dynamic Learning," *Statistica Sinica*, in press: doi:10.5705/ss.202022.0172.