Particle Flow Filter and Differentiable Particle Filter

Wenhao Xu [1],

Abstract

This report investigates advanced state estimation techniques, progressing from classical linear filters to modern particle flow methods. I review the theoretical foundations of the Kalman Filter, Extended/Unscented Kalman Filters, and Particle Filters. I then focus on the Particle Flow Particle Filter (PF-PF) and kernel-embedded Particle Flow Filter (PFF) to address the particle degeneracy problem in nonlinear and high-dimensional systems. Comprehensive experiments on Linear Gaussian (LG) Models, Stochastic Volatility (SV) models, Multi-Target Acoustic Tracking (MTAT) Models and high-dimensional chaotic systems (Lorenz 96) demonstrate the efficacy and numerical characteristics of these algorithms.

1 Introduction and Literature Review

State estimation in dynamic systems is a fundamental problem in engineering and statistics. The objective is to infer the latent state x_t given a sequence of noisy observations $y_{1:t}$.

1.1 Classical Filtering Methods

For Linear Gaussian State Space Models (LGSSM), the Kalman Filter (KF) provides the optimal analytical solution. However, practical implementations must address numerical stability. As noted in Bucy and Joseph (2005), the standard covariance update can lose positive definiteness due to round-off errors. The Joseph stabilized form is often preferred to maintain the symmetry and positive definiteness of the covariance matrix.

For nonlinear systems, approximations are required. The Extended Kalman Filter (EKF) linearizes the system via Taylor expansion, while the Unscented Kalman Filter (UKF) uses a deterministic sampling approach (sigma points) to capture higher-order moments. However, these Gaussian approximations fail when the posterior is highly non-Gaussian or multi-modal.

The Particle Filter (PF) approximates the posterior using a set of weighted samples. While asymptotically optimal, PFs suffer from the curse of dimensionality and weight degeneracy, where the effective sample size drops to zero, necessitating frequent resampling (Johansen 2009).

1.2 Challenges in Stochastic Volatility Models

A specific challenge addressed in this report is the Stochastic Volatility (SV) model. As defined in (Johansen 2009), the observation noise is multiplicative ($y_n = \beta \exp(x_n/2) w_n$). Standard additive-noise filters fail here. Harvey et al. (1994) propose a Quasi-Maximum Likelihood (QML) approach by log-transforming the squared observations to linearize the system ($\log y_n^2 = h(x_n) + \text{noise}$).

[1] Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong, China. Email: `whxu@se.cuhk.edu.hk`.

However, Jacquier et al. (2002) argue that this transformation introduces $\log \chi^2$ distributed noise, which is highly skewed, rendering Gaussian approximations inefficient. This necessitates the use of Particle Filters or advanced flow-based methods.

1.3 Particle Flow and Kernel Methods

To overcome particle degeneracy without aggressive resampling, Particle Flow Filters (PFF) transport particles from the prior to the posterior distribution via a continuous flow governed by a partial differential equation (log-homotopy). Daum et al. (2010) introduce the Exact Daum-Huang (EDH) flow, and Daum and Huang (2011) introduce the Localized EDH (LEDH) flow, which approximate the flow using local linearizations. However, the particle flow ODE cannot be solved analytically in most situations because the partial differential equation doesn't admit a unique solution. Li and Coates (2017) propose the invertible Particle Flow Particle Filter (PF-PF) framework to deal with this problem, and propose the PF-PF LEDH Algorithm and the PF-PF EDH Algorithm.

Hu and Van Leeuwen (2021) propose the matrix-valued Kernel-Embedded Particle Flow Filter, which computes the flow in a Reproducing Kernel Hilbert Space (RKHS). This non-parametric approach avoids explicit linearization and leverages the gradient of the log-likelihood directly, offering robustness in chaotic systems like Lorenz 96 and mitigating particle collapse.

1.3.1 Tasks of the Report

In this report, I implement and compare the methods introduced in the previous section. I begin with the Kalman Filter to establish a baseline for numerical stability. I then move to the SV model to demonstrate the limitations of EKF/UKF and the necessity of PFs. Finally, I implement PF-PF and Kernel-PFF to solve the degeneracy problem in complex, high-dimensional tracking tasks, validating the theoretical advantages proposed by Li and Coates (2017) and Hu and Van Leeuwen (2021).

2 Software Implementation and Verification

To ensure the reliability and mathematical correctness of the algorithms implemented in this report, a comprehensive unit testing suite is designed using the Python unittest framework and TensorFlow.

2.1 Testing Plan

The testing strategy is divided into two main categories:

1. Model Dynamics Verification: Verifying that the underlying state-space models (LGSSM, SV, MTAT, Lorenz 96) generate data with correct shapes, types, and statistical properties, and that likelihood computations are numerically valid.

2. Filter Algorithm Verification: Validating the initialization, prediction, and update steps of the filters (KF, EKF, UKF, PF, PF-PF, kernel PFF). This includes checking for numeri-

cal stability (e.g., absence of NaN), tensor shape consistency, and adherence to theoretical properties (e.g., covariance symmetry).

2.2 Detailed Verification: Stochastic Volatility and Particle Flow

As a representative example of my rigorous verification process, I detail the testing of the Stochastic Volatility (SV) model and the Particle Flow Filter (PFF).

2.2.1 Model Verification

I verify the SVModel class to ensure it correctly represents the system dynamics defined in equation (2).

- Data Generation: Tests confirm that generate_data() produces tensors of shape $(T,)$ without NaN values.
- Likelihood and Gradients: A critical requirement for particle flow is the computation of $\nabla_x \log p(y|x)$. I utilize tf.GradientTape in my tests (test_log_likelihood_gradients) to confirm that the model yields finite, non-zero gradients for the flow equation.

2.2.2 Filter Verification

The SVParticleFlowFilter class is tested under both EDH/LEDH and Kernel methods:

- Initialization: I verify that particles are initialized with the theoretical stationary variance $\sigma^2/(1-\alpha^2)$. The test test_init_particles confirms the sample variance matches the theoretical value within a 25% tolerance.
- Covariance Computation: The helper function for computing particle covariance is validated against a known dataset (test_compute_covariance).
- Kernel Properties: For the Kernel-PFF, I verify that the RBF kernel matrix is constructed correctly (diagonal elements equal to 1.0) and that gradients are computable (test_rbf_kernel).
- Numerical Stability: I simulate update steps for both EDH/LEDH and Kernel methods (test_update_edh_integration, test_update_kernel_integration). These tests pass, confirming that the differential equation solvers execute without divergence or numerical overflow.
- Trajectory Loop: A full predict-update loop simulation (test_full_trajectory_loop) confirms that the filter maintains valid state estimates over sequential time steps.

2.3 Testing Results Summary

All defined tests across all experiments pass successfully.

- For the Linear Gaussian State Space Model, tests confirm the symmetry of the Joseph-form covariance update.
- For the Multi-Target Acoustic Tracking Model, tests validate the Jacobian matrices against numerical approximations.
- For the Lorenz 96, tests confirm the distinction between scalar and matrix-valued kernels.

The successful execution of this suite provides strong evidence that the experimental results presented in Section 3 are based on a correct software implementation.

3 Experiments and Results

3.1 Linear-Gaussian State Space Models (LGSSM)

In this part, I will realize Kalman filter in LGSSM with tensorflow. I use the LGSSM defined in Example 2 in Johansen (2009).

3.1.1 Experiment Setup

The system dynamics and observation model are governed by the following equations:

$$\begin{aligned} x_n &= Ax_{n-1} + Bv_n, \\ y_n &= Cx_n + Dw_n, \end{aligned} \tag{1}$$

where $x_n \in \mathbb{R}^4$ represents the latent state and $y_n \in \mathbb{R}^2$ represents the observation at time n. The noise terms v_n and w_n are assumed to be independent standard Gaussian random vectors, i.e., $v_n \sim \mathcal{N}(0, I_4)$ and $w_n \sim \mathcal{N}(0, I_2)$, where $I_k \in \mathbb{R}^{k\times k}$ is the identity matrix.

The parameters of the model are initialized as follows to simulate a slowly evolving process with precise measurements. A is defined as an identity matrix perturbed by small random noise: $A = I_4+E,\ E_{ij} \sim \mathcal{N}(0, 0.01^2)$. The process noise B is scaled by a factor of 0.3, such that $B = 0.3I_4$. The mapping $C \in \mathbb{R}^{2\times 4}$ from the state space to the observation space is random. The elements of C are drawn from a standard normal distribution $\mathcal{N}(0, 1)$. The measurement noise is set to be very low relative to the process noise, with $D = 10^{-3}I_2$.

The simulation is initialized with a diffuse prior to represent high uncertainty about the starting state. The initial state distribution is set to $x_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$, where $\mu_0 = \mathbf{0}$ and $\Sigma_0 = 1000I_4$. The experiment is conducted over a horizon of $T = 100$ time steps.

3.1.2 Results

In this experiment, I compare the performance of the standard Kalman filter covariance update against the Joseph stabilized form, using the tfp.distributions.LinearGaussianStateSpaceModel (TFP) as a validated benchmark. The evaluation focuses on estimation accuracy, numerical stability, and the preservation of covariance matrix properties.

As illustrated in Figure 1, all three implementations—Standard, Joseph, and TFP—exhibit nearly identical trajectories in tracking the true state. The error curves overlap completely, indicating that under this specific linear Gaussian setup, the choice of covariance update formula does not significantly alter the immediate posterior mean estimates.

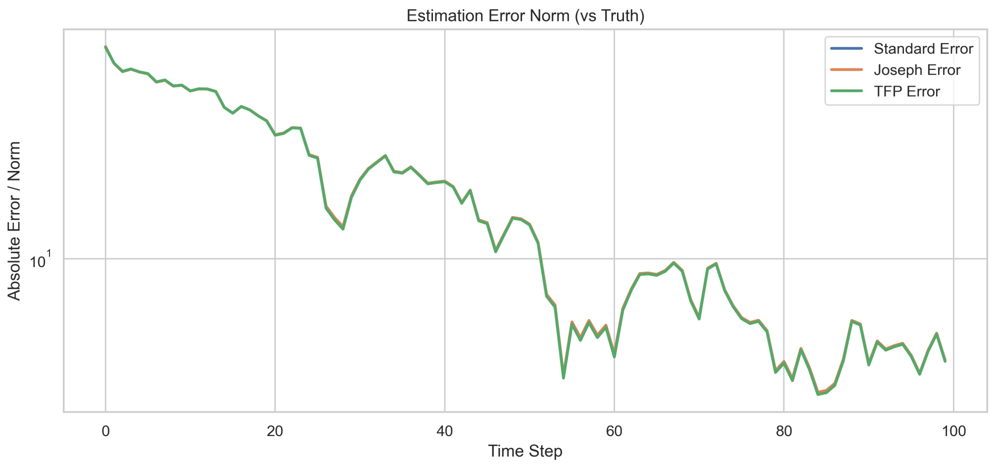


Figure 1: This figure shows the absolute error between the estimated trajectories and the true trajectory under the three different methods.

While the tracking accuracy is similar, significant differences arise in the numerical properties of the covariance matrices. The standard update formula, $P_t = (I - K_tC)P_{t|t-1}$, is computationally cheaper but susceptible to numerical errors such as loss of symmetry.

I calculate the 2-norm of the difference between the estimated covariance matrix and its transpose to verify symmetry. The results highlight a clear degradation in the symmetry of the estimated covariance matrix of the standard method. The estimated covariance matrices of the standard method exhibit a maximum asymmetry of 0.227, which is substantial for a covariance matrix. In contrast, the Joseph form, which structurally guarantees symmetry via the update $P_t = (I - K_tC)P_{t|t-1}(I - K_tC)^T + K_tDD^TK_t^T$, reduces the maximum asymmetry significantly (8.15×10^{-3}), performing even better than the TFP benchmark (1.13×10^{-2}).

The most critical distinction between the methods is observed in the preservation of positive definiteness. The experimental setup involves a high initial covariance and random matrices, leading to an ill-conditioned problem as evidenced by Figure 2. The standard filter suffers from severe numerical instability, reaching a maximum condition number of 1.86×10^{10}, which is significantly higher than both the Joseph form (5.45×10^9) and TFP (2.70×10^9). Remarkably, the Joseph update yielded a mean condition number of 1.60×10^8, slightly outperforming the TFP benchmark (1.68×10^8). This result underscores the superior numerical stability of the Joseph form, demonstrating that it is highly effective at mitigating ill-conditioning, performing on par with, and in this specific metric even surpassing, the optimized library implementation. The standard filter fails to maintain a valid covariance matrix in a significant portion of the experiment, achieving a positive definite ratio of only 65%. This implies that in 35% of the time steps, the covariance matrix had at least one non-positive eigenvalue, which can lead to filter divergence in longer simulations. The Joseph update is the most robust, maintaining positive definiteness in 80.00% of the steps, slightly outperforming even the TFP benchmark (77.00%) in this specific realization.

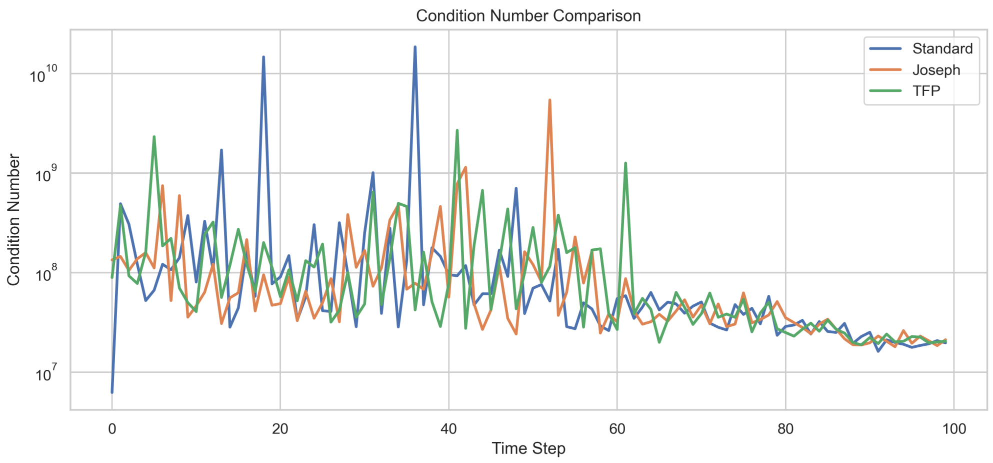


Figure 2: This figure compares the condition number of the estimated covariance matrices of the three methods.

Figure 3 shows that the numerical divergence is highest at the beginning of the simulation (likely due to the large initial covariance $\Sigma_0 = 1000I$) and decays over time. The Frobenius norm of the covariance difference is identical for both methods, suggesting that while the Joseph form corrects the structure (symmetry/definiteness) of the matrix, the element-wise values remain close to the standard calculation.

The experiment demonstrates that while the standard and Joseph forms provide equivalent state estimates in ideal conditions, the Joseph stabilized update is essential for numerical robustness. The standard filter's inability to maintain positive definiteness in 35% of the steps renders it unreliable for this specific ill-conditioned system. The Joseph form successfully mitigates numerical round-off errors, preserving the symmetry and positive definiteness required for a valid Kalman filter implementation.

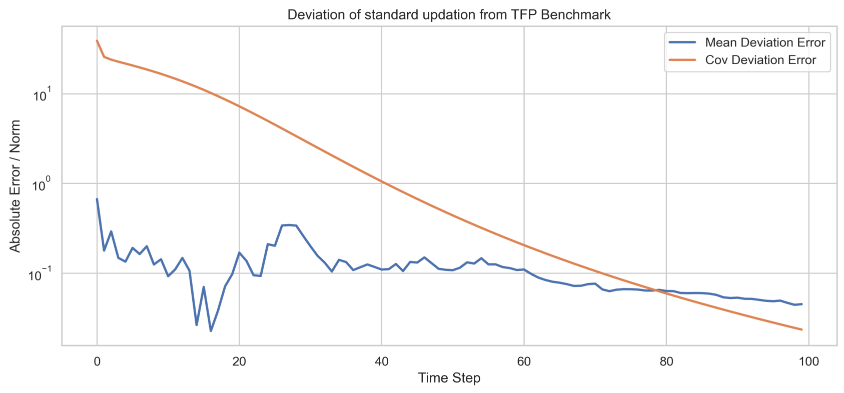
3.2 Nonlinear/Non-Gaussian State Space Models

In this section, I will deal with a nonlinear and non-Gaussian state space model with the extended Kalman filter (EKF), the unsented Kalman filter (UKF) and the standard particle filter (PF).

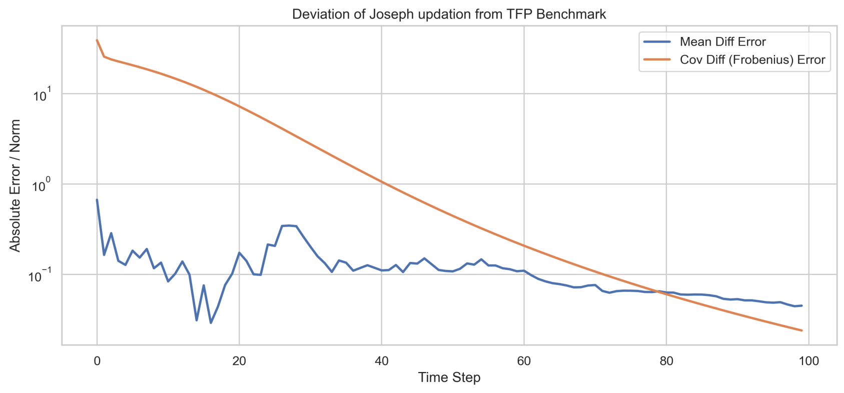
3.2.1 Experiment Setup

To evaluate the performance of the implemented filters on a nonlinear and non-Gaussian system, I utilize the stochastic volatility (SV) model. I adopt the specific formulation and parameter set described in Example 4 in Johansen (2009). The discrete-time evolution is governed by the following equations:

$$\begin{aligned} x_n &= \alpha x_{n-1} + \sigma v_n, \\ y_n &= \beta \exp(\frac{x_n}{2}) w_n, \end{aligned} \tag{2}$$



(a) Standard update



(b) Joseph update

Figure 3: The two figures shows the performance of the standard filter and Joseph filter relative to the TFP benchmark.

where x_n represents the latent state and y_n represents the observation at time n. $v_n \sim \mathcal{N}(0,1)$ and $w_n \sim \mathcal{N}(0,1)$ are mutually independent standard Gaussian noise terms. The non-linearity arises in the observation equation, where the state x_t enters through an exponential function, acting as a multiplicative variance factor. I utilize the exact parameter values provided in Johansen (2009), where $\alpha = 0.91$, $\sigma = 1$, $\beta = 0.5$. The initial state x_0 is drawn from the distribution $x_0 \sim \mathcal{N}(0, \frac{\sigma^2}{1-\alpha^2})$. Based on the parameters above, the initial variance is approximately $\frac{1^2}{1-0.91^2} \approx 5.817$. The simulation generates a trajectory of length $T = 100$.

As I have discussed in Section 1.2, the canonical stochastic volatility model presents a significant challenge for standard filtering algorithms due to its observation equation, and Harvey et al. (1994) square the returns and take the logarithm to deal with this challenge. In my experiment, I will try both the standard and the transformed stochastic volatility model with EKF and UKF and compare their performance.

3.2.2 Results

In this section, I evaluate the performance of EKF, UKF and PF applied to the stochastic volatility model. I specifically investigate the impact of the observation equation formulation by comparing raw filters (operating on $y_n^2 = \beta^2 e^{x_n} \epsilon_n^2$) against transformed filters (operating on $\log(y_n^2) = c + x_n + \xi_n$).

I implement both EKF and UKF on the raw and transformed models. The results highlight the severe limitations of Gaussian filters when dealing with highly non-linear, multiplicative noise. As shown in Figure 4 (dashed yellow and teal lines), both EKF (Raw) and UKF (Raw) fail to track the true state accurately and exhibit significant divergence. The raw observation function $h(x_n) = \beta^2 \exp(x_n)$ has a Jacobian $H_n = \beta^2 \exp(x_n)$. Because the gradient depends exponentially on the current state estimate, small estimation errors are amplified. If the filter overestimates x_n, the Kalman gain becomes excessively large, causing over-correction and instability. The UKF (Raw) also performs poorly. The UKF assumes that the posterior distribution can be approximated by a Gaussian captured by a set of symmetric sigma points. However, the distribution of the squared returns y_n^2 is highly skewed and heavy-tailed. The symmetric sigma points fail to capture this asymmetry, leading to biased mean and covariance estimates. Conversely, the EKF and UKF applied to the log-transformed data (pink and green lines) track the true state effectively. The transformation $\log(y_n^2)$ renders the measurement equation linear with respect to the state x_n. Consequently, the Jacobian is constant ($H = 1$), and the EKF becomes mathematically equivalent to a standard Kalman Filter with non-Gaussian noise. As observed in the results, the EKF and UKF achieve identical RMSE (0.9693). This is expected because, for a strictly linear observation equation, the UKF sigma points propagate the mean and covariance exactly as the linear equations of the EKF do.

The PF is implemented using 1000 particles and systematic resampling. It achieves the RMSE (0.9742), which is close to the results of the EKF and UKF. Unlike the EKF/UKF which approximate the $\log(\chi_1^2)$ noise as Gaussian (with mean -1.27 and variance $\pi^2/2$), the PF uses the exact likelihood of the SV model. This allows it to capture the true posterior density, including the heavy tails and skewness that the Gaussian approximations ignore. The phenomenon of particle degeneracy is monitored using the effective sample size (ESS), visualized in Figure 5. The ESS fluctuates

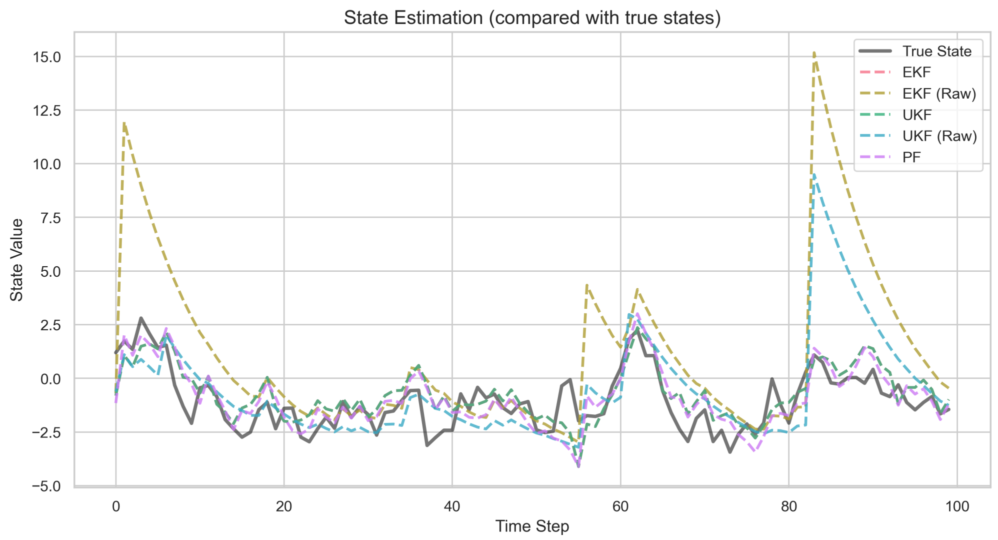


Figure 4: This figure plots the trajectories of the estimated states under different methods.

significantly over time. High volatility updates often lead to a few particles carrying the majority of the weight, causing the ESS to drop sharply. I utilize a resampling threshold of $N/2 = 500$ (red dashed line). As seen in the plot, whenever the ESS drops below this line, systematic resampling is triggered, restoring the ESS to N (1000). This mechanism prevents weight collapse, ensuring the filter maintains sample diversity throughout the trajectory.

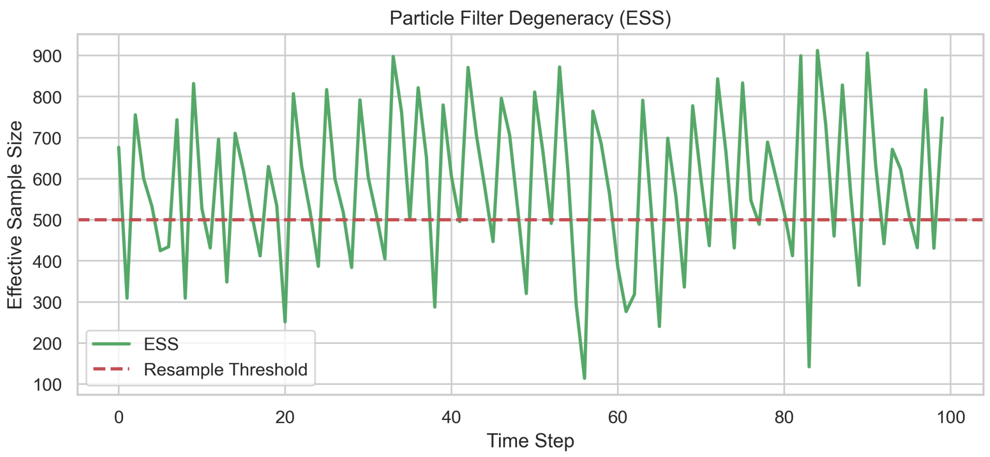


Figure 5: This figure plots the effective sample size (ESS) of particle filter method.

The performance of the three different methods is presented in Table 1. The EKF, UKF, and PF perform comparably. The transformed SV model is close enough to a linear Gaussian system that the optimal linear estimator (approximated by EKF) performs as well as the non-linear estimator (PF). The PF does not provide a statistically significant advantage in RMSE for this specific parameter set. The EKF is the most efficient, requiring only 0.05s. The UKF is the slowest (0.2797s) due to the overhead of sigma-point generation and reconstruction, which is unnecessary given the linearity of the transformed model. The UKF is the most memory-efficient (35KB), while the PF's memory usage scales with the number of particles.

Table 1: Performance

Algorithm	RMSE	Runtime (s)	Peak Memory (KB)
EKF (Transformed)	0.9693	0.0483	704.39
UKF (Transformed)	0.9693	0.2692	32.26
Particle Filter	0.9742	0.2451	272.15

For the standard SV model, the transformed EKF is the optimal choice. It provides the best balance of accuracy and speed. The log-transformation successfully linearizes the system, allowing the EKF to match the accuracy of the computationally expensive Particle Filter. The Particle Filter is a valid approach but is over-engineered for this specific problem unless the noise distribution deviates significantly further from Gaussianity.

3.3 Particle Flow Particle Filter

In this section, I will replicate the experiment Multi-Target Acoustic Tracking in Section V in Li and Coates (2017) with particle flow particle filter (PF-PF).

3.3.1 Experiment Setup

I simulate the movement of $C = 4$ independent targets within a 2D region. The state vector $x_k \in \mathbb{R}^{16}$ consists of the position and velocity components for all four targets:

$$x_k = \left[x_k^{(1)}, y_k^{(1)}, \dot{x}_k^{(1)}, \dot{y}_k^{(1)}, \dots, x_k^{(4)}, y_k^{(4)}, \dot{x}_k^{(4)}, \dot{y}_k^{(4)}\right]^\top . \tag{3}$$

The targets evolve according to a linear Constant Velocity (CV) model:

$$x_k = F x_{k-1} + v_k, \tag{4}$$

where F is the block-diagonal state transition matrix constructed using a time step $dt = 1$, and v_k is the Gaussian process noise with covariance Q. To ensure the targets remain within the region of interest, I implement a boundary limitation mechanism: if a target exceeds the boundaries defined by $[-5, 45]$, I reflect its velocity to keep it within the tracking area.[2] I deploy a network of $N_s = 25$ sensors arranged in a uniform grid pattern at intervals of 10 meters, covering coordinates from $(0, 0)$ to $(40, 40)$. The sensors measure the superposition of acoustic signals emitted by the targets. The measurement $z_k^{(s)}$ at the s-th sensor is modeled as an additive amplitude function that decays with distance

$$z_k^{(s)} = \sum_{c=1}^{4} \frac{\Psi}{\left\| p_k^{(c)} - r^{(s)} \right\|_2 + d_0} + w_k^{(s)}, \tag{5}$$

[2]If a target is far from the sensors, the sensors cannot provide the correct data for the PFPF.

where $p_k^{(c)}$ is the position of the c-th target, $r^{(s)}$ is the location of the s-th sensor, and $||\cdot||_2$ denotes the Euclidean distance. I set the signal amplitude $\Psi = 10$ and the offset parameter $d_0 = 0.1$. The measurement noise w_k is Gaussian with a covariance matrix $R = \sigma_w^2 I$, where $\sigma_w^2 = 0.01$. This small measurement noise results in a highly peaked likelihood function, which typically causes severe weight degeneracy in standard bootstrap particle filters.

I initialize the simulation with the specific starting coordinates provided in the configuration (e.g., Target 1 starts at $[12, 6]$). The simulation runs for $T = 40$ time steps. For the filtering algorithm, I use $N_p = 500$ particles. To support the flow and EKF prediction steps, I compute the Jacobian of the measurement function analytically as defined in the measurement_jacobian method. The process noise covariance Q used in the filter is set with larger entries (diagonal elements of 3 for position and 0.03 for velocity) to account for model uncertainty during tracking.

3.3.2 Results

The performance of the PF-PF is evaluated using two distinct flow variations: the Localized Exact Daum-Huang (LEDH) flow and the Exact Daum-Huang (EDH) flow. The evaluation metrics are the optimal mass transfer (OMAT) error for accuracy and execution time for computational efficiency. The multi-target tracking trajectories of PF-PF (LEDH) are shown in Figure 6. The simulation yields the following average results over the 40 time steps:

- PF-PF (LEDH): Average OMAT error of 0.5731m with a runtime of 28.65s.
- PF-PF (EDH): Average OMAT error of 3.1939m with a runtime of 5.31s.

These results align with the theoretical expectations and the findings reported by Li and Coates (2017). The LEDH algorithm achieves significantly higher accuracy, reducing the tracking error to less than 1 meter. In contrast, the EDH algorithm, while much faster, suffers from a substantially higher tracking error.

The difference in execution time is a direct consequence of how the flow parameters are computed in the run_flow method. In LEDH, I calculate the linearization matrices A and b for each particle individually. This requires N_p matrix inversions per flow step, leading to a higher computational load. In EDH, I compute the flow parameters only once per step based on the sample mean of the particles. This treats the flow as a single global affine transformation applied to the entire particle cloud, drastically reducing the number of matrix operations.

I observe a distinct difference in the stability of the error trajectories between the two methods, as shown in Figure 7. The LEDH error remains consistently low and stable throughout the simulation. By linearizing the measurement model locally at each particle's position, the LEDH flow correctly guides particles. This allows the filter to maintain a precise representation of the four distinct targets.

I notice that the EDH error exhibits a large fluctuation, rising significantly before eventually decreasing. This behavior—which differs slightly from the smoother averaged plots in the original paper—can be attributed to the limitations of the EDH approximation in this specific scenario. The EDH flow relies on linearizing the measurement function $h(x)$ at the mean of the particle

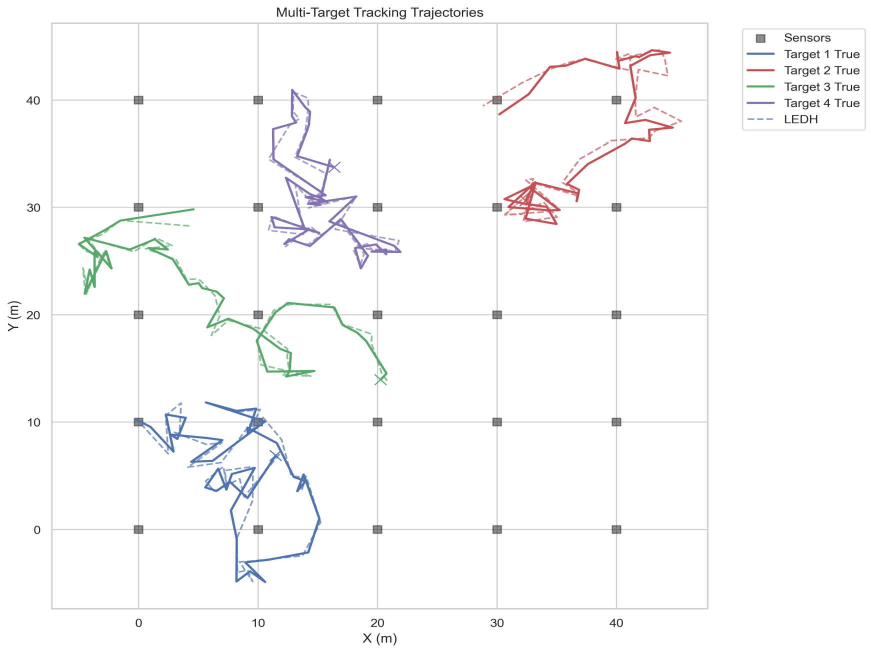


Figure 6: This figure plots the multi-target tracking trajectories of PF-PF (LEDH).

distribution ($\bar{\eta}$). In a multi-target tracking scenario, the posterior distribution is often multi-modal (with peaks centered at each target). The mean of such a distribution often falls in a region of low probability (e.g., the empty space between targets). When the targets are well-separated, the mean $\bar{\eta}$ does not represent any actual target. Calculating the Jacobian and flow parameters at this empty location results in a flow field that may not point towards the true high-probability regions. This inaccurate flow pushes particles away from the true states, causing the OMAT error to spike. The error eventually decreases later in the simulation due to the Importance Sampling step. Even if the flow is suboptimal, the weight update penalizes particles far from the measurements. Additionally, the systematic resampling step eliminates particles with negligible weights, allowing the filter to recover, albeit with a delay.

3.4 Kernel-Embedded Particle Flow Filter

In this experiment, I evaluate the performance of the Kernel-Embedded Particle Flow Filter (PFF) in Hu and Van Leeuwen (2021) using the high-dimensional Lorenz 96 model and compare the behavior of the filter using a scalar-valued kernel versus a matrix-valued kernel.

3.4.1 Experiment Setup

I follow the experimental design in Section 3 in Hu and Van Leeuwen (2021). I employ the Lorenz 96 model with a state dimension of $N_x = 1000$. The forcing constant is set to $F = 8$, which induces chaotic behavior. I integrate the system dynamics using the fourth-order Runge-Kutta (RK4) method with a physical time step of $\Delta t = 0.01$. To generate the ground truth state ($\mathbf{x}_{\text{true}}$),

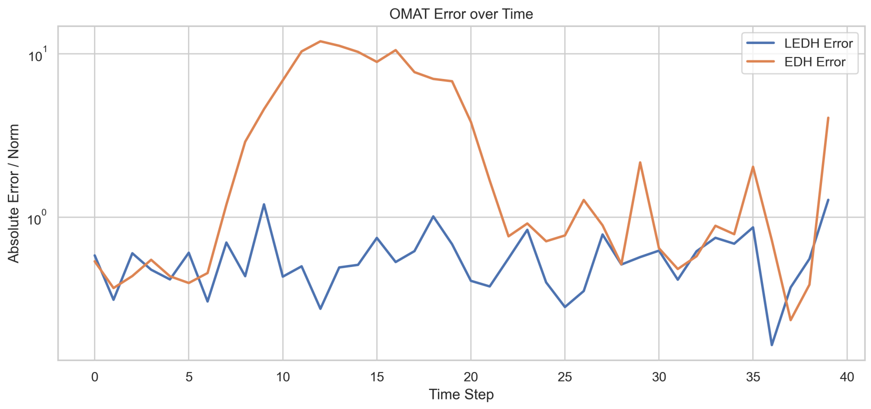


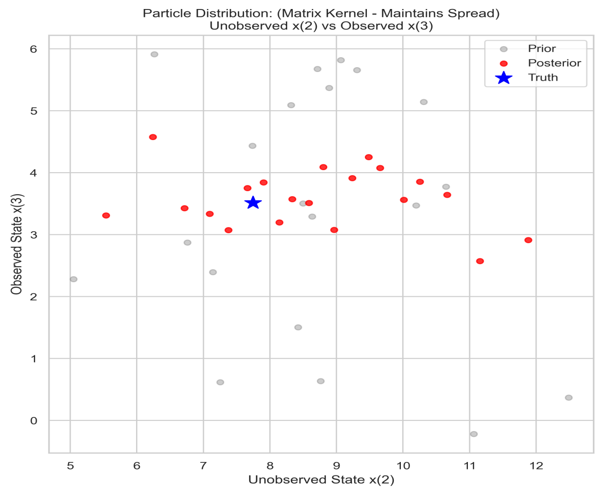
Figure 7: This figure shows the OMAT of the PF-PF (LEDH) and PF-PF (EDH).

I initialize the system and run the simulation for 1,000 time steps. I construct a sparse observation network where only 25% of the state variables are observed. Specifically, I observe every 4th grid point, resulting in an observation dimension of $N_y = 250$. The synthetic observations $\mathbf{y}$ are generated by adding uncorrelated Gaussian noise to the true state values at the observed locations, with an observation noise standard deviation of $\sigma_{\text{obs}} = 0.5$. I use an ensemble size of $N_p = 20$ particles. The prior ensemble is initialized by perturbing the true state with isotropic Gaussian noise with a standard deviation of $\sigma_{\text{prior}} = 2$.

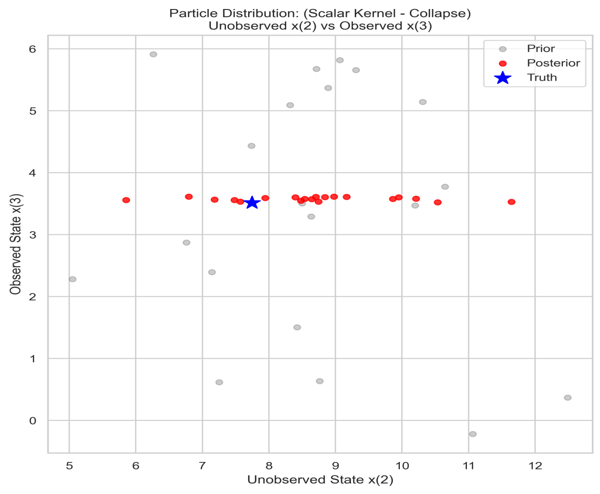
For the flow integration in pseudo-time (λ), I set the pseudo-time steps: $N_{\text{steps}} = 100$, the pseudo-time step size: $\Delta s = 0.05$, the width of the kernel: $\alpha = 1/N_p$.

3.4.2 Experiment Result

I implement the Kernel-Embedded PFF as described in Hu and Van Leeuwen (2021), comparing the scalar-valued kernel and the matrix-valued kernel. The results, visualized in Figure 8, replicate the Figure 3 in Hu and Van Leeuwen (2021) and clearly demonstrate the necessity of the matrix-valued kernel for high-dimensional filtering.



(a) Matrix-valued



(b) Scalar

Figure 8: This figure compares the matrix-valued kernel and the scalar-valued kernel. It replicates the Figure 3 in Hu and Van Leeuwen (2021).

3.5 Deterministic and Kernel Flows on Stochastic Volatility Model

In this section, I will implement the EDH, LEDH, kernel PFF on the stochastic volatility(SV) model.

3.5.1 Experiment Setup

I use the same parameters of the stochastic volatility model as those in Section 3.2.1. To evaluate the performance and stability of the proposed methods, I compare four distinct filter implementations: the standard EDH with log-transformation (EDH-Log), the Exact EDH without transformation (EDH-Exact), the Exact Localized EDH (LEDH-Exact), and the Kernel-Embedded Particle Flow Filter (Kernel PFF).

For all filters, the simulation duration is set to $T = 100$ time steps. The particle ensemble size is set to $N_p = 100$. To ensure a fair comparison of the flow integration process, all methods utilize the same pseudo-time configuration with $N_\lambda = 50$ steps, resulting in a step size of $\Delta s = 0.02$. The particles are initialized by sampling from the stationary distribution of the state process, $x_0 \sim \mathcal{N}(0, \frac{\sigma^2}{1-\alpha^2})$, ensuring the initial ensemble variance matches the system's inherent uncertainty.

3.5.2 Experiment Result

This experiment compares four PFF variants on the SV model. The SV model poses a specific challenge due to its highly nonlinear observation equation, where noise enters multiplicatively: $y_k = \beta \exp(x_k/2) w_k$.

The Root Mean Square Error (RMSE) results reveal a distinct bifurcation in performance between methods that effectively handle the model structure and those that rely on direct linearization. According to Figure 9, the EDH_Log method achieves the lowest error. This is expected because the logarithmic transformation $z_k = \log(y_k^2)$ transforms the observation equation into a linear form. And my experiment in Section 3.2.2 has proved the effectiveness of logarithmic transformation. The Kernel method performs nearly as well as EDH_Log (difference of < 0.05) without requiring the manual log-transformation. This demonstrates the robustness of the Kernel PFF. By using the exact gradient of the log-likelihood $\nabla_x \log p(y|x)$ and mapping the flow in a Reproducing Kernel Hilbert Space (RKHS), the Kernel method captured the posterior geometry effectively. It proves that non-parametric flow estimation can handle severe nonlinearities where standard linearization fails.

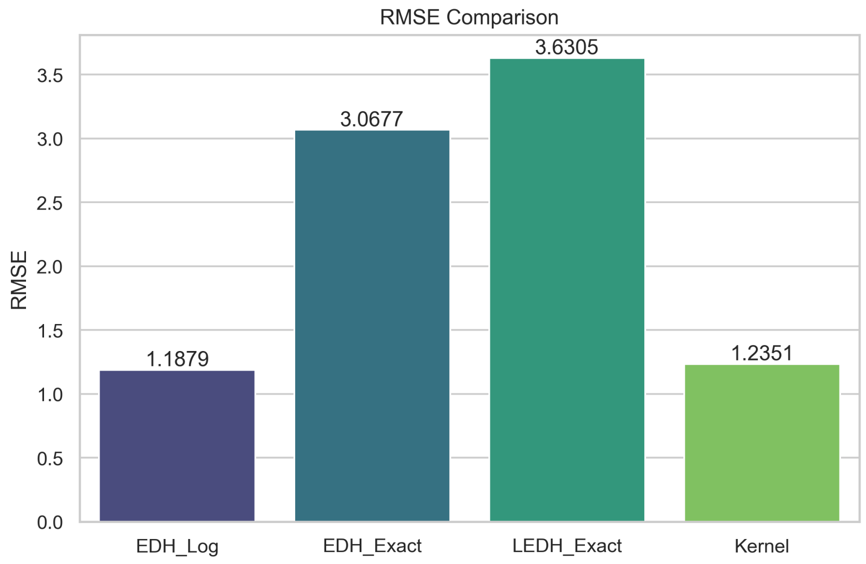


Figure 9: This figure plots the RMSE of the four different methods.

Both EDH_Exact and LEDH_Exact methods (operating on the raw SV model) fail significantly, with errors nearly triple that of the best methods. For the EDH_Exact method, linearizing $h(x) = \exp(x/2)$ around the global mean $\bar{x}$ is insufficient. The exponential function is convex; thus, the value at the mean is not the mean of the values (Jensen's inequality). The linearized slope at $\bar{x}$ does not represent the gradients required by particles in the tails of the distribution, leading to incorrect flow updates. In terms of the LEDH_Exact method, particles at the upper tail of the distribution (large x) produce massive gradients due to the $\exp(x/2)$ term, while particles at the lower tail produce near-zero gradients in the SV model. This disparity creates an extremely stiff ODE for the flow. Individual particles likely experience massive, unstable drifts, causing the ensemble mean to diverge wildly.

I do the stability diagnostics in Figure 10. The LEDH_Exact and EDH_Exact methods likely exhibit large spikes in flow magnitude. In the context of particle flow, high drift magnitude often indicates numerical stiffness. When the measurement y_k is an outlier, the exponential gradient forces particles to move too fast within the pseudo-time steps ($\Delta\lambda$), leading to overshooting and poor estimation. The EDH_Log and kernel methods likely show more consistent, lower-magnitude flow profiles. The EDH_Log method is stable because the transformed measurement function is linear (constant gradient slope of 1). The kernel method is stable because the RBF kernel acts as a smoothing operator. Even if the likelihood gradient is steep, the kernel convolution averages out the drift influence from neighboring particles, preventing individual particles from exploding.

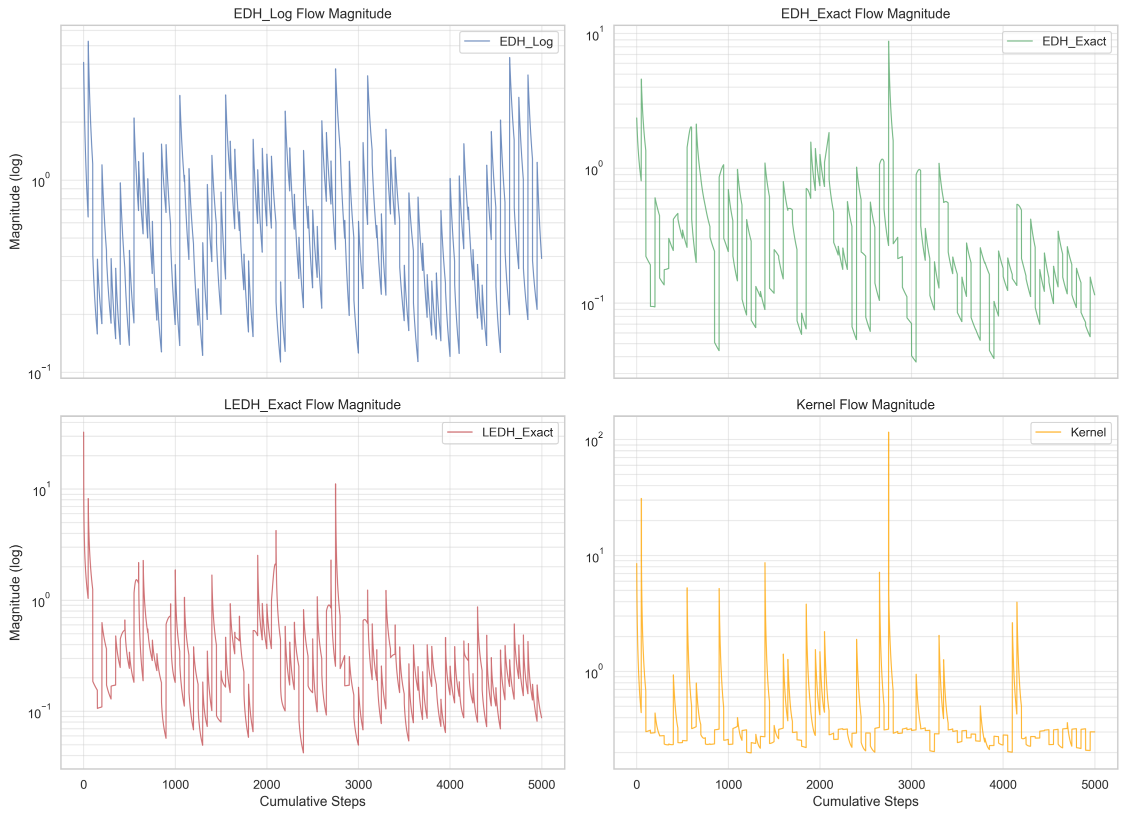


Figure 10: This figure plots the flow magnitude of the four different methods.

According to Figure 11, the True State (black line) and EDH_Log/kernel estimates (pink/purple dashed) likely overlap closely, capturing the sudden jumps in volatility. The Exact methods (green/teal dashed) likely show a lagging behavior. They may fail to capture the amplitude of the volatility spikes. Because they operate on the raw exponential scale, they are highly sensitive to measurement noise—a single large $|y_k|$ can pull the un-transformed filters too far, while the log-transform dampens this noise.

4 Conclusion

This report systematically evaluated filtering algorithms from the Kalman Filter to the Kernel-Embedded Particle Flow Filter. I demonstrate that while the Joseph-form Kalman Filter ensures stability for linear systems, nonlinearities (like in the SV model) require advanced techniques. The Particle Flow Filter effectively mitigates degeneracy, but standard EDH/LEDH flows can fail in multi-modal or highly nonlinear settings. The Kernel-Embedded PFF proves to be a robust alter-

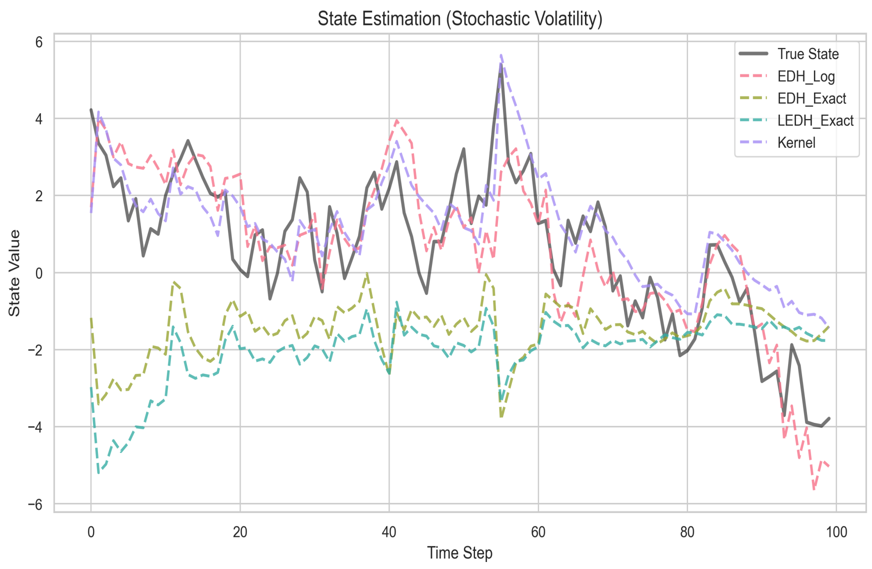


Figure 11: This figure plots the estimated trajectories of the four different methods and the true trajectory.

native, performing well in both high-dimensional chaotic systems and highly nonlinear observation models without requiring manual linearization transformations.

References

Bucy, R. S. and P. D. Joseph (2005). *Filtering for stochastic processes with applications to guidance*, Volume 326. American Mathematical Soc.

Daum, F. and J. Huang (2011). Particle degeneracy: root cause and solution. In *Signal Processing, Sensor Fusion, and Target Recognition XX*, Volume 8050, pp. 367–377. SPIE.

Daum, F., J. Huang, and A. Noushin (2010). Exact particle flow for nonlinear filters. In *Signal processing, sensor fusion, and target recognition XIX*, Volume 7697, pp. 92–110. SPIE.

Harvey, A., E. Ruiz, and N. Shephard (1994). Multivariate stochastic variance models. *The Review of Economic Studies 61*(2), 247–264.

Hu, C.-C. and P. J. Van Leeuwen (2021). A particle flow filter for high-dimensional system applications. *Quarterly Journal of the Royal Meteorological Society 147*(737), 2352–2374.

Jacquier, E., N. G. Polson, and P. E. Rossi (2002). Bayesian analysis of stochastic volatility models. *Journal of Business & Economic Statistics 20*(1), 69–87.

Johansen, A. (2009). A tutorial on particle filtering and smoothing: Fifteen years later.

Li, Y. and M. Coates (2017). Particle filtering with invertible particle flow. *IEEE Transactions on Signal Processing 65*(15), 4102–4116.