From Classical Filters to Particle Flows
Report of JP Morgan MLCOE TSRL 2026 Internship
Question 2

Yaxin Feng

January 19, 2026

1 Part I.

1.1 Warm-up.

For those who are not familiar with filtering, we need to spend some time learning these basics before the next part.

1.1.1 Linear-Gaussian SSM with Kalman Filter

a) Implement the Kalman filter for a multidimensional linear-Gaussian SSM.
Use synthetic data from a standard LGSSM, see examples 2 in [15].

Example 1. Linear Gaussian model [15]. Here, $\mathcal{X}=\mathbb{R}^{n_x}, \mathcal{Y}=\mathbb{R}^{n_y}, X_1 \sim \mathcal{N}(0, \Sigma)$ and

$$\begin{aligned} X_n &= A X_{n-1}+B V_n, \\ Y_n &= C X_n+D W_n \end{aligned}$$

where $V_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}\left(0, I_{n_v}\right), W_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}\left(0, I_{n_w}\right)$ and A, B, C, D are matrices of appropriate dimensions. Note that $\mathcal{N}(m, \Sigma)$ denotes a Gaussian distribution of mean m and variance-covariance matrix Σ, whereas $\mathcal{N}(x ; m, \Sigma)$ denotes the Gaussian density of argument x and similar statistics. In this case $\mu(x)=\mathcal{N}(x ; 0, \Sigma)$, $f\left(x^{\prime} \mid x\right)=\mathcal{N}\left(x^{\prime} ; A x, B B^{\mathrm{T}}\right)$ and $g(y \mid x)=\mathcal{N}\left(y ; C x, D D^{\mathrm{T}}\right)$. As inference is analytically tractable for this model, it has been extremely widely used for problems such as target tracking and signal processing.

Kalman Filter (KF) Algorithm for Ex. 1. Kalman filter recursion steps is the process of iteratively calculating the optimal state estimate by alternating between two phases: Prediction (Time Update) and Update (Measurement Update).

Step 1: Prediction Step

The Prediction step projects the posterior estimate from the previous time step, $\hat{X}_{n-1|n-1}$ and $P_{n-1|n-1}$, forward to the current time n. This results in a prior estimate, before the measurement Y_n is seen.

A. State Prediction (Prior Estimate)

The previous optimal state estimate is projected forward using the State Transition Matrix (A).

$$\hat{X}_{n|n-1} = A\hat{X}_{n-1|n-1}$$

B. Covariance Prediction

The previous error covariance is projected forward, and the uncertainty introduced by the Process Noise Covariance ($Q = BB^T$) is added.

$$P_{n|n-1} = AP_{n-1|n-1}A^T + Q$$

Step 2: Measurement Update (Correction)

The Update step incorporates the new measurement, Y_n, to correct the prior estimate, resulting in the optimal posterior estimate.

A. Innovation Covariance

This step calculates the innovation covariance, using the Observation Matrix (C) and the Observation Noise Covariance ($R = DD^T$).

$$S_n = CP_{n|n-1}C^T + R$$

B. Kalman Gain

The Kalman Gain (K_n) is calculated to determine how much the filter should trust the current measurement (R) versus its own prediction ($P_{n|n-1}$).

$$K_n = P_{n|n-1}C^T S_n^{-1}$$

C. State Update (Posterior Estimate)

The final state estimate is calculated by adding the weighted difference between the actual measurement (Y_n) and the expected measurement ($C\hat{X}_{n|n-1}$) to the prior estimate. The difference $\tilde{Y}_n = Y_n - C\hat{X}_{n|n-1}$ is known as the innovation or measurement residual.

$$\hat{X}_{n|n} = \hat{X}_{n|n-1} + K_n\tilde{Y}_n$$

D. Covariance Update

The error covariance is updated, reflecting the reduced uncertainty after the measurement is incorporated. (I is the identity matrix).

Standard Form:

$$P_{n|n} = (I - K_nC)P_{n|n-1}$$

Joseph Form:

$$P_{n|n} = (I - K_nC)P_{n|n-1}(I - K_nC)^T + K_nRK_n^T$$

The Joseph Form is a better way to calculate $P_{n|n}$ because it offers enhanced numerical stability and guaranties that the covariance matrix remains valid, which is crucial for the long-term operation of the filter.

Case 1. Consider a 2 dimensional constant velocity (CV) problem as the example here.

Let the state vector

$$X_n = \begin{bmatrix} p_{x,n} \\ v_{x,n} \\ p_{y,n} \\ v_{y,n} \end{bmatrix} \in \mathbb{R}^4,$$

here are the recursion steps in mathematical form, repeating for $n = 1, 2, 3, \ldots, N$.

State transition matrix (A):

$$A = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where the time step is $\Delta t = 0.1$.

Process noise coupling matrix (B):

$$B = \begin{bmatrix} \Delta t^2/2 & 0 \\ \Delta t & 0 \\ 0 & \Delta t^2/2 \\ 0 & \Delta t \end{bmatrix}$$

Observation matrix (C): measures p_x and p_y only.

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Observation noise coupling matrix (D):

$$D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

After generating the synthetic data from the LGSSM, the Kalman Filter is applied for target tracking; see Fig. 1.

b) Analyze its filtering optimality and numerical stability: compare filtered means/covariances to the Kalman recursion; use Joseph stabilized covariance updates; discuss conditioning number.

1. Filtered Means and Statistical Consistency

By analyzing the filtered means' estimation error residuals against the theoretical covariance bounds, the filter's statistical validity is confirmed. Since the estimation error is modeled as a Gaussian random variable with covariance $P_{n|n}$, theoretical statistics dictate that the actual error should fall within the $\pm 2\sigma$ (two standard deviations) bounds approximately 95% of the time.

The results of Case 1 in Fig. 2 demonstrate that the filter is statistically consistent. The estimation errors for both the X-position (blue) and Y-position (orange) fluctuate randomly around zero, confirming that the estimator is unbiased. Furthermore, the errors remain strictly confined within the shaded 2σ bounds. The plot illustrates a clear "convergence phase" during the first 10 time steps, where the confidence region shrinks rapidly as the filter processes initial measurements. Following this, the bounds plateau, indicating that the filter has reached a steady state where the

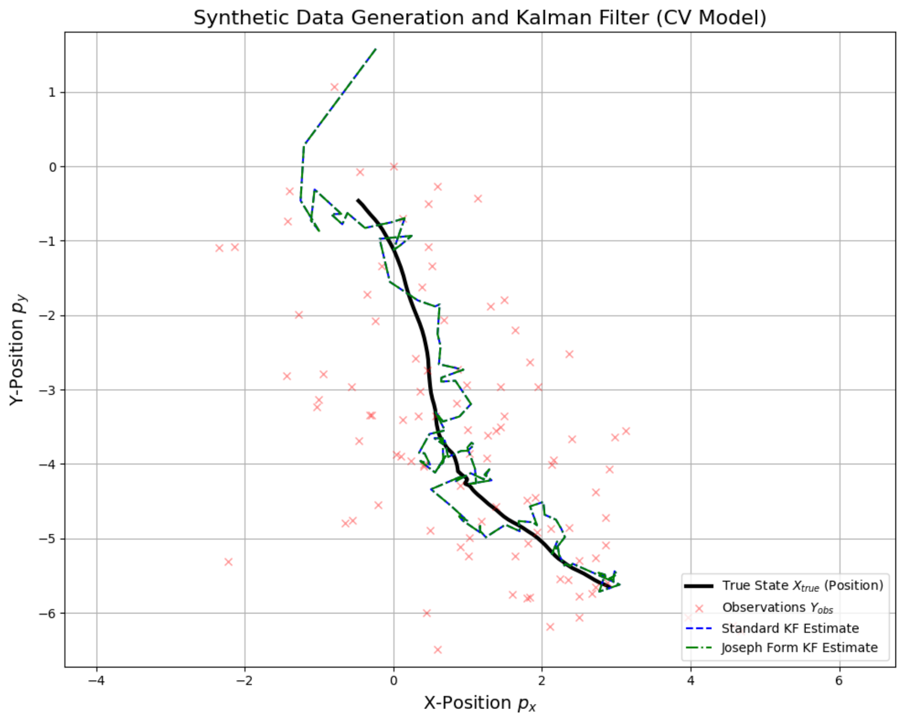

Figure 1: CV Model synthetic data generation and results of Joseph/Standard kalman filters.

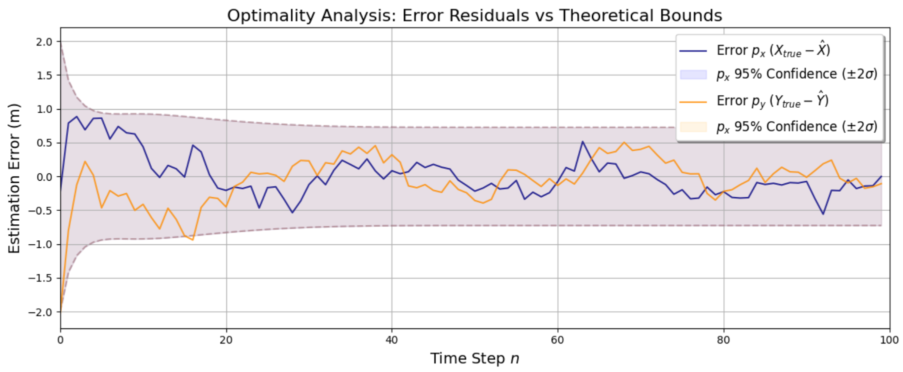

Figure 2: Comparison of condition number of $\kappa(P_{n|n})$ during the Joseph kalman recursion with different q_{factor}.

information gained from new measurements is in equilibrium with the uncertainty introduced by process noise.

2. Geometric Interpretation of Tracking Performance by the Filtered Means and Covariances

The tracking performance is visualized through the trajectory and covariance ellipses in 2D space. The plot displays in Fig. 3 is the true state trajectory (black line) against the noisy raw observations (red crosses) and the filter's means (estimated path, dashed lines). This experiment is based on the Case 1.

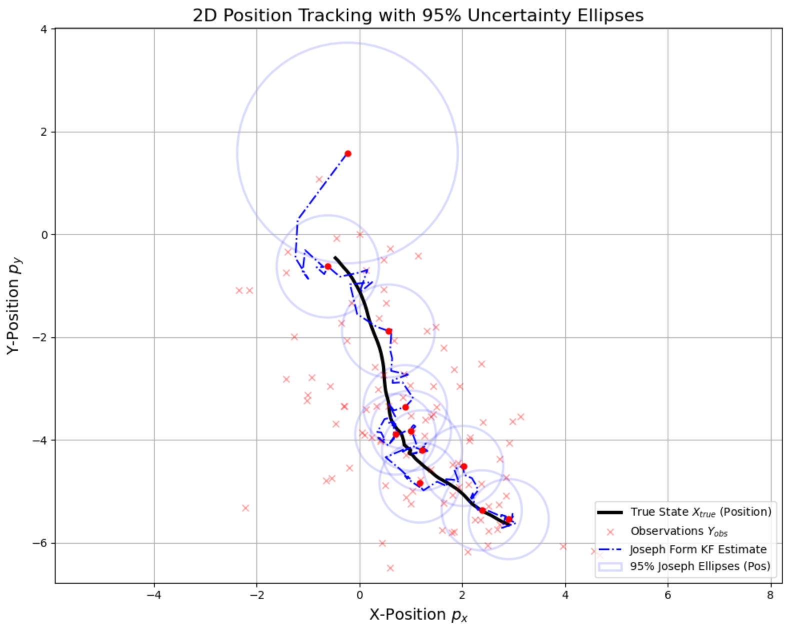

Figure 3: True path X_{true} and uncertainty ellipses drawed by the filtered mean $\hat{X}_{n|n}$ and the covariance $P_{n|n}$ during the Joseph Kalman Recursion.

The covariance matrices are represented as uncertainty ellipses surrounding the estimated state. At initialization ($t = 0$), the large circle at the top-left reflects the high initial uncertainty ($P_{0|0}$). As the simulation progresses, these ellipses rapidly shrink, visually demonstrating the reduction in entropy (uncertainty) as the filter incorporates sensor data. The circular shape of these ellipses corresponds to the balanced observation noise model, where the uncertainty resolves symmetrically in both the X and Y dimensions. The estimate smoothly tracks the true state, effectively rejecting the measurement noise visible in the raw observations.

3. Numerical Stability Analysis via Condition Number

The numerical health of the filter is assessed using the condition number of the estimation error covariance matrix [10], denoted as $\kappa(P_{n|n})$. It is a measure of how close it is to being singular (non-invertible). The higher the condition number, the worse the numerical stability. This metric is calculated as the ratio of the largest to the smallest eigenvalue, $\kappa(P_{n|n}) = \lambda_{\max}/\lambda_{\min}$, indicating the eccentricity of the uncertainty ellipsoid.

In the scenario with Balanced Observation Noise ($D = I$), like in Fig. 4, the filter demonstrates robust numerical stability. The condition number remains relatively low, oscillating between 10^1 and 10^2. The analysis reveals that higher process noise (e.g., $q = 10$) acts as a stabilizing factor, "inflating" uncertainty in all directions uniformly and maintaining a more spherical covariance. Conversely, when process noise is low ($q = 0.001$), the condition number increases; the filter becomes highly confident in the directly observed position variables ($\lambda_{\min} \to 0$) while remaining uncertain about unobserved velocities. This experiment is based on the Case 1, except q_{factor}.

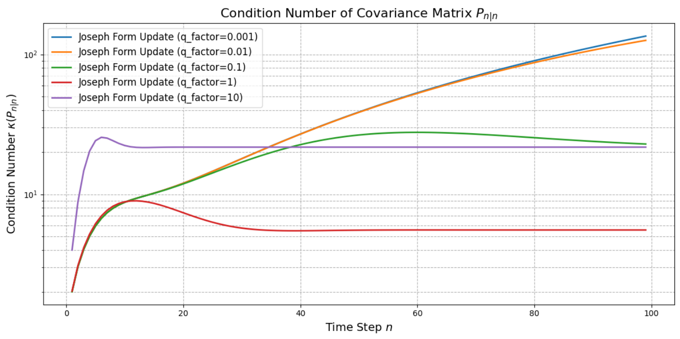

Figure 4: Comparison of condition number of $P_{n|n}$ during the Joseph Kalman Recursion with different q_{factor}.

In the scenario with Unbalanced Observation Noise ($\tilde{D} = \text{diag}(1, 10^{-4})$), like in Fig. 5, the system exhibits a fundamentally different behavior. The condition number exhibits a high baseline of approximately 10^8, accurately reflecting the 10^4 magnitude disparity between the precise and noisy sensors. Interestingly, the effect of process noise is reversed compared to the balanced case. Here, lower process noise ($q = 0.001$) improves conditioning (lowering κ from 10^8 to 10^7) because the strong reliance on the physics model smooths the noisy sensor data, reducing the variance disparity. High process noise ($q = 100$) prevents this smoothing, locking the condition number at a higher instability level. This experiment is based on the Case 1, except q_{factor} and a new $\tilde{D}$.

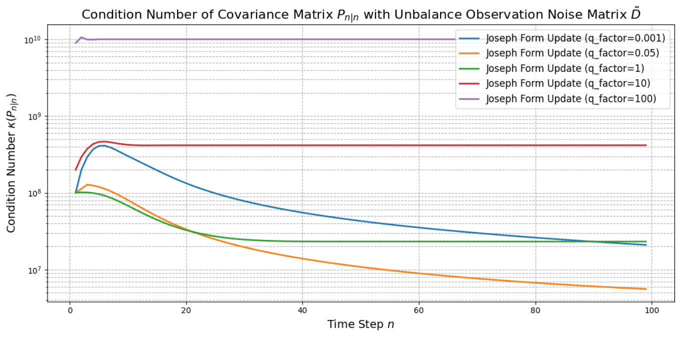

Figure 5: Comparison of condition number of $P_{n|n}$ during the Joseph Kalman Recursion with different q_{factor} and unbalance $\tilde{D}$.

1.1.2 Nonlinear/Non-Gaussian SSM with EKF/UKF and Particle Filter

a) Design a nonlinear and non-Gaussian SSM, you can use a stochastic volatility model (example 4 in [15]) or nonlinear tracking models (e.g. Range-Bearing observation model)

Example 2. Stochastic Volatility (SV) model [15]. We have $\mathcal{X} = \mathcal{Y} = \mathbb{R}$, $X_1 \sim \mathcal{N}\left(0, \frac{\sigma^2}{1-\alpha^2}\right)$ and

$$
\begin{aligned}
X_n &= \alpha X_{n-1} + \sigma V_n \\
Y_n &= \beta \exp\left(X_n/2\right) W_n
\end{aligned}
$$

where $V_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1)$ and $W_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1)$. In this case we have $\mu(x) = \mathcal{N}\left(x; 0, \frac{\sigma^2}{1-\alpha^2}\right)$, $f\left(x' \mid x\right) = \mathcal{N}\left(x'; \alpha x, \sigma^2\right)$ and $g(y \mid x) = \mathcal{N}\left(y; 0, \beta^2 \exp(x)\right)$. Note that this choice of initial distribution ensures that the marginal distribution of X_n is also $\mu(x)$ for all n. This type of model, and its generalisations, have been very widely used in various areas of economics and mathematical finance: inferring and predicting underlying volatility from observed price or rate data is an important problem. Figure 1 shows a short section of data simulated from such a model with parameter values $\alpha = 0.91, \sigma = 1.0$ and $\beta = 0.5$ which will be used below to illustrate the behaviour of some simple algorithms.

Generate the synthetic data from the Nonlinear/Non-Gaussian SSM; see Fig. 6:

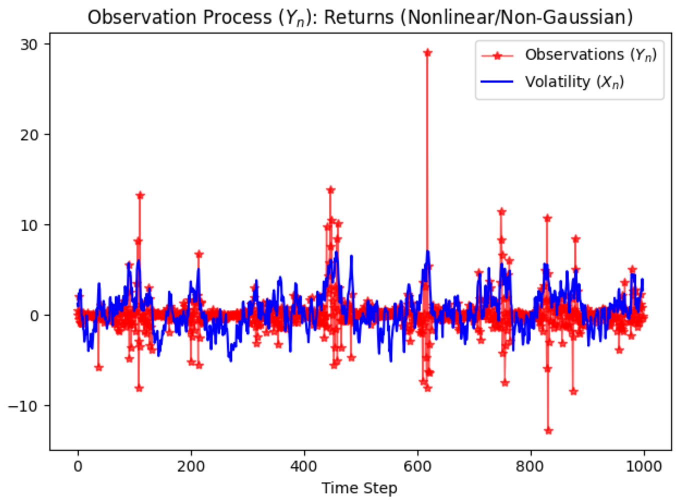

Figure 6: Simulation of the SV model of example 4 in [15].

Remark 1. The SV model defines $Y_n = \beta \exp\left(X_n/2\right) \cdot W_n$, where W_n flips the sign randomly. This means Y_n fluctuates around zero, $\tilde{h}(X_n) = \mathbb{E}[Y_n \mid X_n] = 0$. Therefore, transformation is needed before applying the kalman filter series approaches.

- Square-transformation:

$$Z_n = Y_n^2 = \beta^2 \exp(X_n) W_n^2,$$

Measurement Function: $h(X_n) = \mathbb{E}[Z_n | X_n] = \beta^2 \exp(X_n)$ (Since $\mathbb{E}[W^2] = 1$)

Measurement Noise Variance (R_n): $\text{Var}(Z_n) = 2 \cdot (\beta^2 \exp(X_n))^2$ (Since $\text{Var}(W^2) = 2$)

- Log-square-transformation:

$$Z_n = \log(Y_n^2) = \underbrace{X_n}_{\text{Linear}} + \underbrace{(\log \beta^2 + \log W_n^2)}_{\text{Constant + Noise}},$$

  the measurement function become linear too.

b) Implement the Extended Kalman filter (EKF) and Unsent Kalman Filter (UKF), discuss linearization accuracy limits and sigma point failures under strong nonlinearity.

Remark 2. Log-square-transformation makes the problem to be linear, the extended kalman filter (EKF) and unscented kalman filter (UKF) is likely to degenerate to classical KF. In this problem, non-linear models of EKF and UKF need to discussed, thus I analyze the square-transformation.

Extended Kalman Filter (EKF).

Standard Kalman Filters work only for linear systems. To handle nonlinearity, the EKF uses Jacobian matrices (matrices of partial derivatives) to linearize the state transition and measurement functions at each time step.

State Transition Jacobian (F): Linearizes the function predicting the next state.

Measurement Jacobian (H): Linearizes the function mapping the state to the observed measurement.

The EKF handles the nonlinearity by linearizing the functions around the current estimate using Jacobians.

EKF Algorithm for Ex. 2.

Step 1: Prediction

The state transition is linear ($X_n = \alpha X_{n-1} + V_n$), so we use standard Kalman prediction.

A. Predicted State: $\hat{X}_{n|n-1} = \alpha \hat{X}_{n-1|n-1}$

B. Predicted Covariance: $P_{n|n-1} = \alpha^2 P_{n-1|n-1} + \sigma^2$

Step 2: Estimation Update (Correction)

We linearize the measurement $Z = \beta^2 \exp(X)$ around the predicted state.

A. Measurement Jacobian (H_n): Compute the derivative with respect to X.

$$H_n = \frac{\partial}{\partial X}\left(\beta^2 e^X\right)\Big|_{X=\hat{X}_{n|n-1}} = \beta^2 e^{\hat{X}_{n|n-1}}$$

B. Observation Noise (R_n): Calculate variance based on the current estimate,

$$R_n = 2 \cdot \left(\beta^2 e^{\hat{X}_{n|n-1}}\right)^2$$

C. Innovation Covariance (S_n):

$$S_n = H_n P_{n|n-1} H_n^T + R_n$$

D. Kalman Gain (K_n):
$$K_n = P_{n|n-1} H_n^T S_n^{-1}$$

E. State Update:
$$\hat{X}_{n|n} = \hat{X}_{n|n-1} + K_n \left(Z_n - \beta^2 e^{\hat{X}_{n|n-1}} \right)$$

F. Covariance Update:
$$P_{n|n} = (I - K_n H_n) P_{n|n-1}$$

Unscented Kalman Filter (UKF).

While the EKF linearizes the function using derivatives (Jacobians), the UKF uses the Unscented Transform. It picks a small set of sample points—called Sigma Points—propagates them through the actual nonlinear function, and then reconstructs the Gaussian distribution.

UKF Algorithm for Ex. 2.

UKF addresses nonlinearity by performing a statistical linear regression using a set of representative sample points. This method is called the Unscented Transform (UT) [16]. The UKF's accuracy depends heavily on how well the small set of sigma points can capture the behavior of the non-linear function over the spread defined by the current covariance P.

Step 1: Prediction

A. Generate Sigma Points ($\mathcal{X}_{n-1}$): Create sample points around $\hat{X}_{n-1}$ based on covariance P_{n-1}.

B. Propagate Sigma Points: Pass each point through the process model: $\mathcal{X}^* = \alpha \mathcal{X}_{n-1}$.

C. Predicted Mean & Covariance: Calculate $\hat{X}_{n|n-1}$ and $P_{n|n-1}$ from the weighted samples $\mathcal{X}^*$.

Step 2: Estimation Update (Correction)

A. Redraw Sigma Points: Generate new points $\mathcal{X}_{n|n-1}$ around the predicted mean $\hat{X}_{n|n-1}$.

B. Transform Sigma Points: Pass each point through the exact measurement function.
$$\mathcal{Z}_i = \beta^2 \exp(\mathcal{X}_i)$$

C. Predicted Observation ($\hat{Z}$): Calculate the weighted mean of $\mathcal{Z}_i$.

D. Observation Noise (R_n): Calculate dynamic variance based on $\hat{Z}$.
$$R_n = 2 \cdot \hat{Z}^2$$

E. Innovation Covariance (S_n): Calculate the covariance of the transformed points plus noise.
$$S_n = \sum W_c (\mathcal{Z}_i - \hat{Z})^2 + R_n$$

F. Cross Covariance (P_{xz}): Calculate correlation between state sigma points and measurement sigma points.
$$P_{xz} = \sum W_c (\mathcal{X}_i - \hat{X}_{n|n-1})(\mathcal{Z}_i - \hat{Z})$$

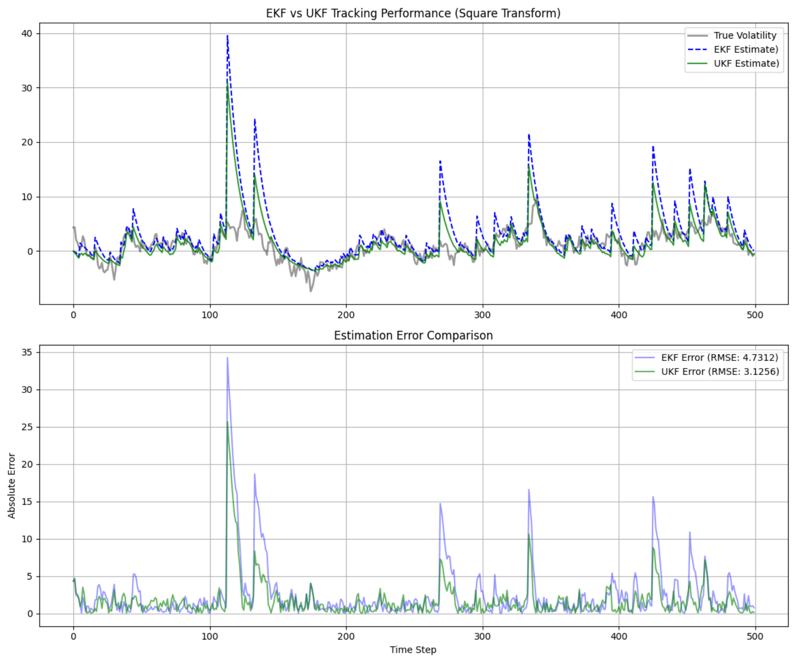

Figure 7: Comparison of the SV model in [15] by EKF and UKF.

G. State Update:

$$K_n = P_{xz} S_n^{-1}$$

$$\hat{X}_{n|n} = \hat{X}_{n|n-1} + K_n (Z_n - \hat{Z})$$

$$P_{n|n} = P_{n|n-1} - K_n S_n K_n^T$$

Linearization Accuracy Limits:

The main accuracy limitation of the Extended Kalman Filter (EKF) [18] is its reliance on a first-order Taylor series approximation, which introduces errors in highly nonlinear systems.

In the 7, The EKF has severe instability due to the limitations of first-order linearization on the exponential measurement function $h(X) = \beta^2 e^X$. The filter relies on the Jacobian $H = \beta^2 e^{\hat{X}}$, which makes the update step highly sensitive to the current state estimate. When the true volatility spikes, the filter's estimate is initially low, resulting in a shallow gradient (small H). A large observation residual divided by this small slope forces a massive, disproportionate upward correction, causing the estimate to "launch" far above the truth (e.g., the spikes near time steps 120 and 420). Conversely, once the estimate is high, the gradient becomes extremely steep, causing the filter to become "stiff" and react too slowly to downward trends. This asymmetry creates the visible "ratcheting" effect where the EKF jumps up violently but decays slowly, leading to the highest Root Mean Square Error (RMSE) of 4.731.

Sigma Point Failures under Strong Nonlinearity:

Sigma Point Failure in Strong Nonlinearity Sigma Point filters (like UKF) fundamentally fail under strong nonlinearity because their fixed, symmetric sampling scheme breaks down when the posterior distribution becomes highly skewed or multimodal [17]. The "Non-Local Sampling" problem occurs when the weighted mean of the sigma points falls into low-probability regions (e.g., the valley between two modes), leading to physically invalid estimates. Additionally, as uncertainty or dimensionality increases, sigma points must be pushed into the "tails" of the distribution to capture the full variance; if the nonlinearity is severe in these tail regions, the transformed points can cause catastrophic covariance inflation or violate physical domain constraints (e.g., generating negative values for strictly positive parameters), often leading to numerical crashes [21]. In such specific cases of quadratic or exponential nonlinearity, the bias introduced by the UKF sampling approximation can actually exceed the linearization bias of the EKF, causing the filter to diverge [12].

The operation $Z_n = Y_n^2$ creates a specific failure mode driven by Exponential Bias amplification; see Fi. 7. As detailed in the provided algorithm, the sigma points are transformed via the convex function $Z_i = \beta^2 \exp(\mathcal{X}_i)$. Because the exponential curve rises sharply, the sigma points on the upper end of the distribution $(\mu + \sigma)$ map to massive values, while the lower points are compressed near zero. When the UKF calculates the weighted mean $\hat{Z}$ (Step 2C), these massive upper-tail values disproportionately dominate the result, pulling the predicted measurement far above the true mode (peak) of the underlying distribution. The filter effectively "hallucinates" a higher measurement because it attempts to fit a symmetric Gaussian approximation to a highly skewed, heavy-tailed Log-Normal distribution.

This failure is directly observable in the tracking plots, where the UKF estimate (green line) consistently spikes significantly higher than the True Volatility (gray line) during peak events. The analytic correction used in the algorithm, $R_n = 2 \cdot \hat{Z}^2$ (Step 2D), attempts to mitigate this by telling the filter to "trust the measurement less" when values are high. However, the error plot demonstrates that this is insufficient: the fundamental mean-shift bias remains, resulting in the UKF overshooting the peaks and generating large errors (RMSE: 3.126) that mirror the EKF's behavior rather than correcting it.

On the other hand, if using log-square-transformation

$$Z_n = \log(Y_n^2),$$

the EKF and UKF perform much better, the results are displayed in Fig. 8.

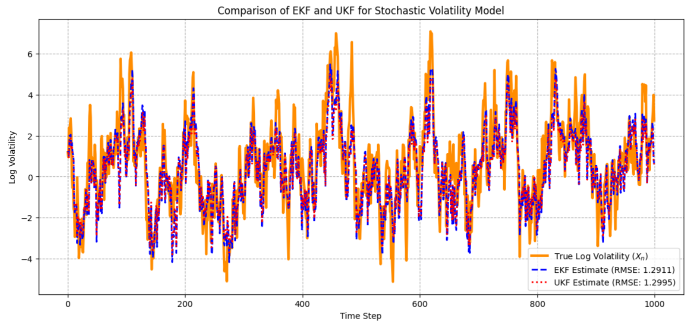

Figure 8: Comparison of the SV model in [15] by EKF and UKF.

c) Implement a standard particle filter for your model. Visualize and discuss issues such as particle degeneracy.

Particle Filter (PF).

The Particle Filter (PF), also known as the Bootstrap Filter or Sequential Importance Resampling (SIR), is the standard algorithm for tracking non-linear and non-Gaussian systems.

Unlike the EKF or UKF, it does not assume the posterior is Gaussian. Instead, it represents the entire distribution using a cloud of random samples (particles).

PF Algorithm for Ex. 2

Step 1: Initialization

Generate M particles $\{X_0^{(i)}\}_{i=1}^M$ drawn from the initial state distribution.

$$X_0^{(i)} \sim \mathcal{N}\left(0, \frac{\sigma^2}{1-\alpha^2}\right)$$

Assign initial weights $w_0^{(i)} = 1/M$.

Step 2: Prediction (Propagation)

At each time step n, propagate every particle forward using the system's process equation. This moves the cloud of particles according to the physics of the volatility.

$$X_n^{(i)} = \alpha X_{n-1}^{(i)} + \sigma V_n^{(i)}, \quad V_n^{(i)} \sim \mathcal{N}(0,1)$$

Step 3: Update (Weighting via Likelihood)

Weigh each particle by calculating the probability of seeing the actual observation Y_n given the particle's state $X_n^{(i)}$.

The Likelihood Function: The observation model is $Y_n = \beta e^{X_n/2} W_n$. This implies that given a specific state X, the observation Y follows a Gaussian distribution centered at 0 with a standard deviation of $\beta e^{X/2}$.

$$p(Y_n|X_n^{(i)}) = \mathcal{N}\left(Y_n; 0, (\beta e^{X_n^{(i)}/2})^2\right)$$

Weight Calculation:

$$w_n^{(i)} \propto \frac{1}{\sqrt{2\pi}(\beta e^{X_n^{(i)}/2})} \exp\left(-\frac{Y_n^2}{2(\beta e^{X_n^{(i)}/2})^2}\right)$$

Step 4: Estimation

The state estimate $\hat{X}_n$ is the weighted average of the particles.

$$\hat{X}_n \approx \sum_{i=1}^{M} w_n^{(i)} X_n^{(i)}$$

Step 5: Calculate Effective Sample Size (ESS) & Conditional Resampling

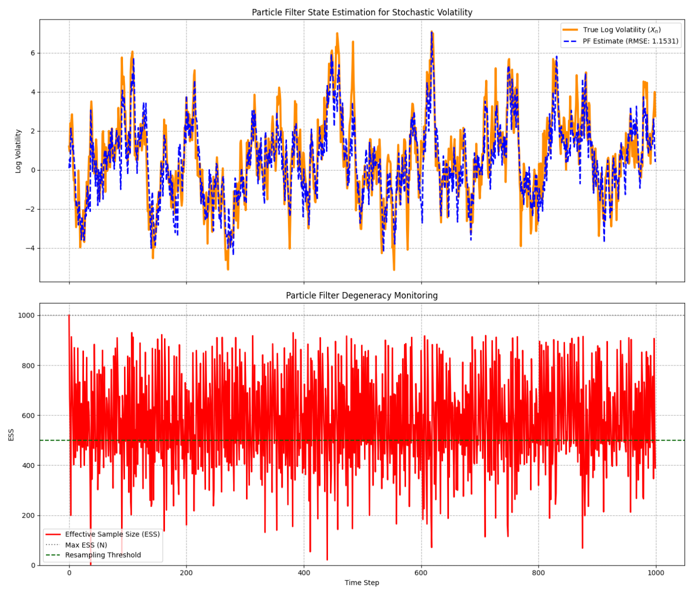

Figure 9: Comparison of the SV model in [15] by PF.

Check if the particle set has degenerated (i.e., if a few particles hold all the weight) using the Effective Sample Size (ESS).

Calculate ESS (N_{eff}): $N_{eff} = \frac{1}{\sum_{i=1}^{M}(w_n^{(i)})^2}$

Check Threshold: Compare N_{eff} to a threshold N_{th} (typically $M/2$). If $N_{eff} < N_{th}$: perform resampling. Select M new particles from the current set with replacement, probability proportional to weights $w_n^{(i)}$. The reset all weights to $1/M$. Otherwise, do not resample and keep the current particles and their uneven weights for the next step.

Degeneracy Problem: In a standard Particle Filter (PF), after multiple cycles of prediction and update, the distribution of weights often becomes heavily skewed. Most of the probability mass concentrates on a single particle (or a very small group).

The ESS ranges from 1 to N, and its value provides immediate feedback on the state of the particle set. When ESS $\approx N$, it means excellent health. All particles have roughly equal weights and contribute equally. When ESS ≈ 1 , it means critical health (Weight Degeneracy). Only one or a few particles have significant weight; all others are negligible.

The main purpose of calculating the ESS is to decide when to resample. It can play the role of resampling threshold: particle filters typically set a threshold (e.g., $\text{ESS}_{\text{threshold}} = N/2$, or a fixed value like 50). If the calculated ESS drops below this threshold, the algorithm triggers a resampling step.

In summary, an average ESS of 577.55 suggests the Particle Filter is well-tuned, and the proposal distribution is generating particles efficiently.

Experiment Result. The result of Particle Filter (PF) is shown in the Fig. 9. The RMSE of the result by PF is lower than those by the EKF and UKF methods.

d) Compare PF and EKF/UKF performance. How to evaluate your SSMs? What metrics can you use? In practice, we care about the runtime and memory, could you also compare the runtime and peak memory(CPU/GPU RAM) for each SSM?

The evaluation metric of the SSMs is RMSE, which are 4.7312 for EKF (square), 3.1256 UKF (square) , 1.2911 for EKF (log-square), 1.2995 UKF (log-square) and 1.1531 (PF), respectively; shown in the Fig. 8 and 9. The RMSE of PF is the lowest.

The metric for runtime (latency) can be Floating Point Operations (FLOPs), and the metric for Peak Memory is primarily the size of the covariance matrix P and the number of particles N.

According to [17], both EKF and UKF have an $\mathcal{O}(n_x^3)$ complexity. [2] and [15] discusses the PF's memory bottleneck, noting that as the state dimension n_x increases, the number of particles N often needs to increase exponentially to maintain accuracy (the curse of dimensionality), exacerbating the memory and runtime cost $\mathcal{O}(N \cdot n_x)$, while the former one confirms the linear dependence on particle count for PF runtime ($\mathcal{O}(N)$).

Method	Runtime (ms)	Peak Memory (KB)
EKF (Square)	7.017625	8.468750
UKF (Square)	32.501625	9.984375
PF (N=100)	59.643333	22.547852
PF (N=1000)	460.705792	72.850586
PF (N=5000)	2197.692333	338.475586

Table 1: Run time and peak memory comparison between EKF, UKF and PF.

1.2 Deterministic and Kernel Flows

a) Study the Exact Daum-Huang (EDH) flow and Local Exact Daum-Huang (LEDH) flow, (see [9] and [8]), and the invertible particle flow particle filter (PF-PF) framework (see [19]).

The exact Daum and Huang filter (EDH) [9, 8] is a theoretically derived particle flow filter designed to address the challenges of sequential state estimation, particularly the problem of weight degeneracy in high-dimensional or highly informative measurement scenarios. It achieves this by modeling a deterministic flow that transports particles from the prior distribution to the desired posterior distribution.

The EDH is incorporated into an encompassing particle filter framework PF-PF (EDH) [19]. In this method, the EDH flow is used as a highly accurate proposal distribution, which follows the Fokker Planck equation with zero diffusion. The zero diffusion particle flow filters involve no random displacements of particles, the flows are deterministic.

The trajectory of the ith particle η_λ^i in [19] follows the ordinary differential equation (ODE):

$$\frac{d\eta_\lambda^i}{d\lambda} = \zeta(\eta_\lambda^i, \lambda) = A(\lambda)\eta_\lambda^i + b(\lambda), \tag{1}$$

where $A(\lambda) = -\frac{1}{2}PH^T\left(\lambda HPH^T + R\right)^{-1}H$, $b(\lambda) = (I+2\lambda A(\lambda))\left[(I+\lambda A(\lambda))PH^TR^{-1}z + A(\lambda)\bar{\eta}_0\right]$ for linear problem. If the problem is nonlinear, $A(\lambda) = -\frac{1}{2}PH(\lambda)^T\left(\lambda H(\lambda)PH(\lambda)^T + R\right)^{-1}H(\lambda)$ and $b(\lambda) = (I + 2\lambda A(\lambda))\left[(I+\lambda A(\lambda))PH(\lambda)^TR^{-1}(z - e(\lambda)) + A(\lambda)\bar{\eta}_0\right]$ where $e(\lambda) = h\left(\bar{\eta}_\lambda, 0\right) -$

$H(\lambda)\bar{\eta}_\lambda$, where the Jacobians of the measurement function is $H(\lambda) = \frac{\partial h(\eta,0)}{\partial \eta}\Big|_{\eta=\bar{\eta}_\lambda}$. This is the linearization operation like the EKF algorithm. Here, $\zeta : \mathbb{R}^d \to \mathbb{R}^d$ is governed by the Fokker-Planck equation and additional flow constraints, it is also names continuity equation

$$\frac{\partial p(\eta_\lambda^i, \lambda)}{\partial \lambda} = -\text{div}(p(\eta_\lambda^i, \lambda)\zeta(\eta_\lambda^i, \lambda)) = -p(\eta_\lambda^i, \lambda)\text{div}(\zeta(\eta_\lambda^i, \lambda)) - \frac{\partial p(\eta_\lambda^i, \lambda)}{\partial \eta_\lambda^i}\zeta(\eta_\lambda^i, \lambda), \tag{2}$$

where $p(\eta_\lambda^i, \lambda)$ is the probability density of η_λ^i at time λ of the flow. Because the flow parameters (A and b) are calculated using the global mean ($\overline{\eta}$), the Jacobian determinant term in the weight update is common to all particles and cancels out during normalization. This makes the PF-PF (EDH) computationally efficient.

The Localized Exact Daum and Huang (LEDH) filter is an advanced particle flow technique that is a variation of the Exact Daum and Huang (EDH) filter. It is designed to address the challenges of nonlinear filtering problems, particularly by making the flow locally adaptive to the posterior density. It needs to track the local flow field of every particles. By performing local linearization, the LEDH constructs a more accurate and robust proposal distribution than the EDH, which is why the PF-PF (LEDH) often exhibits lower tracking error than the PF-PF (EDH) in highly challenging nonlinear scenarios.

In LEDH, the ith particle move by the ODE equaitons: $\zeta\left(\eta_\lambda^i, \lambda\right) = A^i(\lambda)\eta_\lambda^i + b^i(\lambda), i = 1, \cdots, N$, where the $A^i(\lambda)\eta_\lambda^i$ and $b^i(\lambda)$ are different within every particle according to [19].

The key operation in [19] of the EDH flow is the invertable mapping $\eta_1^i = T\left(\eta_0^i; z_k, x_{k-1}^i\right)$ of each particle, which is a kind of pushforward map from the reference distribution. $\dot{T}(\cdot)$ is the Jacobian determinant of the map $T(\cdot)$. The importance weight of each particle in the kth step is $w_k^i \propto \frac{p\left(\eta_1^i|x_{k-1}^i\right)p\left(z_k|\eta_1^i\right)\left|\dot{T}\left(\eta_0^i;z_k,x_{k-1}^i\right)\right|}{p\left(\eta_0^i|x_{k-1}^i\right)} w_{k-1}^i$.

Experiment. The replicated result of the PF-PF (EDH) and PF-PF (LEDH) in [19] comparing with the true data is shown in the Fig. 10, 11 and 12.

For the example 1 in Fig. 10, acoustic tracking model, it has the linear dynamic model, the measurement equation involves nonlinear Euclidean distances in the :

$$z = \sum \frac{\Psi}{\|x - R^s\|_2 + d_0},$$

thus I use Jacobian of the measurement function as linearization.

The example 2 is high-dimensional linear gaussian model (Fig. 11), in which the dynamics is $x_k = \alpha x_{k-1} + v_k$ and the measurement is $z_k = x_k + w_k$, thus it is like a KF model. The paper uses this example to prove that their particle flow method converges to the optimal KF solution in simple cases. The MSE curves for PF-PF (LEDH) and PF-PF (EDH) are nearly identical. Both filters successfully converge to a low error state,thus the EDH is preferred here because it is computationally cheaper ($O(1)$ matrix inversion vs $O(N)$ matrix inversions for LEDH).

In example 3 shown in Fig. 12, The measurement is count data derived from a Poisson distribution where the rate depends exponentially on the state: Rate$\lambda = m_1 e^{m_2 x_k}$, thus linearization is needed by using Jacobian. The MSE stabilizes between 1.5 and 4.0, indicating the filter is not diverging despite the high dimensionality (144 dimensions). The estimated state reconstruction (right) captures the spatial structure of the true state (left) remarkably well in the heatmap.

b) Implement the kernel-embedded particle flow filter (kernel PFF) in an RKHS following Hu(21). Then compare the scalar kernel and diagonal matrix-valued kernel. Use experiments to demon-

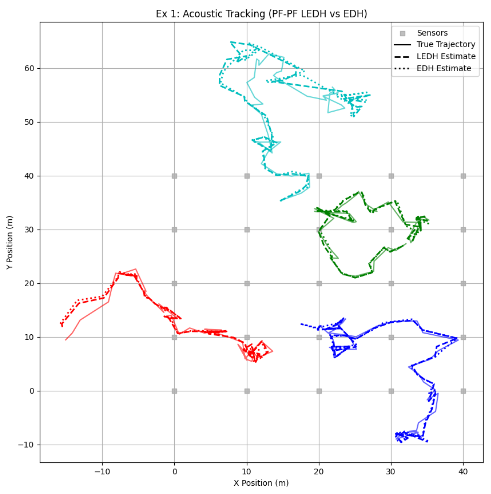

Figure 10: Result of the example 1: acoustic tracking model in [19] by PF.

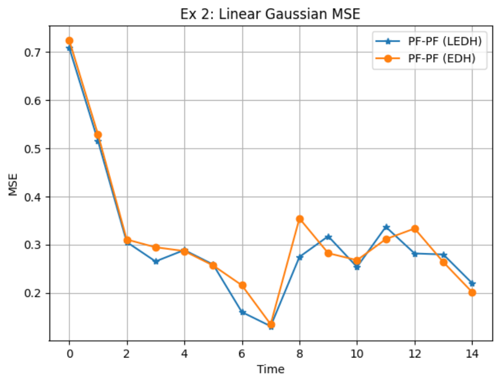

Figure 11: Result of example 2: large spatial sensor networks in [19] by PF.

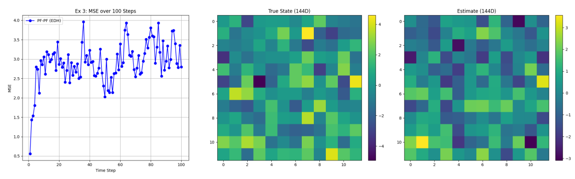

Figure 12: Result of example 3: the skewed-t dynamic model in [19] by PF.

strate the matrix-valued kernel can prevent collapse of observed-variable marginals in high-dimensional, plot similar figures as figure 2-3 in Hu(21)[13].

The motivation of the study of kernel-embedded particle flow filter (kernel PFF) is to investigate how to apply the PFF to high-dimensional nonlinear problems. The scalar kernel version is insufficient in high-dimensional sparsely observed settings. The paper proposes the matrix-valued kernels version of PFF. This is also a kind of recently developed Monte Carlo filter based on a deterministic flow that avoids weight degeneracy problem and has high potential on high-dimensional complicated situations.

When the particle flow (the vector field) is assumed to be embedded in reproducing kernel Hilbert space (RKHS), the analytical expression of the particle flow that minimizes the Kullback-Leibler divergence starting from the prior and ending to the posterior probability distribution function can be derived.

The key components in [13] are listed below. Assume the particles move according to ODE:

$$\frac{d}{ds}\mathbf{x}_s = \mathbf{f}_s\left(\mathbf{x}_s\right), s \in [0, \infty], \tag{3}$$

where $\mathbf{x}_s$ is a state at pseudo time $s, \mathbf{f}_s$ is the particle flow, thus the probability density function (*pdf*) of these particles follows the continuity equation (Liouville equation)

$$\frac{\partial q_s(\mathbf{x})}{\partial s} + \nabla_{\mathbf{x}} \cdot (\mathbf{f}_s(\mathbf{x}) q_s(\mathbf{x})) = 0. \tag{4}$$

The prior *pdf* and target (posterior) *pdf* are $q_0(\mathbf{x}) = p(\mathbf{x}), q_\infty(\mathbf{x}) = p(\mathbf{x} \mid \mathbf{y})$, respectively. The velocity filed $\mathbf{f}_s$ is chosen such that the distance between the prior and posterior *pdf* decreases during the pseudo time s, and the distance is measured by Kullback-Leibler divergence (KL divergence):

$$KL\left(q_s\right) = \int q_s(\mathbf{x}) \log\left(\frac{q_s(\mathbf{x})}{q_\infty(\mathbf{x})}\right) d\mathbf{x}.$$

To make the pseudo time rate change of KL-divergence negative, *i.e.* $\frac{dKL}{ds} \leq 0$,

$$\begin{aligned}\mathbf{f}_s(\cdot) &= \langle \mathbf{f}(\mathbf{x}, \cdot), \mathbf{f}_s(\cdot) \rangle \\ &= \mathbf{D} \int q_s(\mathbf{x}) \left\{ \nabla_{\mathbf{x}} \cdot \mathbf{K}(\mathbf{x}, \cdot) + \mathbf{K}(\mathbf{x}, \cdot) \nabla_{\mathbf{x}} \log(p(\mathbf{x} \mid \mathbf{y})) \right\} d\mathbf{x}.\end{aligned}$$

The Equation (6) in [13] is

$$\mathbf{f}_s(\mathbf{x}) = \frac{1}{N_p} \mathbf{D} \sum_{i=1}^{N_p} \left\{ \mathbf{K}\left(\mathbf{x}_s^i, \mathbf{x}\right) \nabla_{\mathbf{x}_s^i} \log\left(\mathbf{x}_s^i \mid \mathbf{y}\right) + \nabla_{\mathbf{x}_s^i} \cdot \mathbf{K}\left(\mathbf{x}_s^i, \mathbf{x}\right), \right\} \tag{5}$$

which is the Monte Carlo approximation of the integral equation (4) in [13]. It means the flow at the current position x depends on a contribution from every other particle x^i in the swarm. D is the preconditioner matrix (typically the prior covariance), which scales the flow to the right physical dimensions.

In the scaled-kernel method, the kernel is a scalar function of the global distance:

$$\mathbf{K}_S\left(\mathbf{x}_s^i, \mathbf{x}\right) = K\left(\mathbf{x}_s^i, \mathbf{x}\right) \mathbf{I}_{n_x} = \exp\left(-\frac{1}{2}\left(\mathbf{x}_s^i - \mathbf{x}\right)^T \mathbf{A}\left(\mathbf{x}_s^i - \mathbf{x}\right)\right) \mathbf{I}_{n_x}.$$

The scalar K depends on the sum of squared distances across all dimensions. However, when the particles have unbalance distance in different dimensions, for example $x^{(1)} = (0,0)$, $x_{(2)} = (100,1)$, the sum of squared distances across all dimensions will make the interaction (repulsion and attraction) almost vanishes in the dimension 1.

In the proposed matrix-kernel method in [13], the kernel is a diagonal matrix, treating each dimension m independently:

$$\mathbf{K}_M(\mathbf{x},\mathbf{z}) = \operatorname{diag}\left(\left[K_{(1)}(\mathbf{x},\mathbf{z}), K_{(2)}(\mathbf{x},\mathbf{z}), \ldots, K_{(n_x)}(\mathbf{x},\mathbf{z})\right]\right),$$

where

$$K_{(a)}(\mathbf{x},\mathbf{z}) = K_{(a)}\left(x_{(a)}, z_{(a)}\right) = \exp\left(-\frac{1}{2}\frac{\left(x_{(a)} - z_{(a)}\right)^2}{\alpha {\sigma_{(a)}}^2}\right).$$

The interaction in dimension a depends only on the distance in dimension a. Even if particles are far apart in unobserved dimensions, they can be close in the observed dimension, generating a strong local repulsion $k_m \approx 1$. Therefore, the diversity is maintained in all dimensions.

c) Compare EDH,LEDH, and kernel PFF on the SSM you designed in last particle filter question. Analyze when each method excels or fails(nonlinearity, observation sparsity, dimension, conditioning). Include stability diagnostics (flow magnitude, Jacobian conditioning).

One key difference between [19] and [13] is the definition of the velocity field between the prior *pdf* and posterior *pdf*. In [19], it uses a pre-defined linear (approximation) vector field

$$V(x,\lambda) = A(\lambda)x + b(\lambda).$$

A and b are analytical solution to the Riccati differential equation derived from the continuity equation [9]. [13] finds the velocity by optimizing a cost function (Steepest Descent on KL).

Another difference is that the [13] uses kernel methods for the high-dimensional problem.

2 Part II.

2.1 Stochastic particle flow

a) Other than the deterministic particle flow, we can construct stochastic particle flow as shown in Dai(21) [6] and Dai(22) [7]. Please replicate the main results of Dai(22) [7]

[6] describes the particle dynamics via stochastic differential equation (SDE)

$$dx = f(x,\lambda)d\lambda + q(x,\lambda)dw_\lambda \tag{6}$$

instead of the deterministic ODE dynamics in the previous sections [19] and [13], and the flow is the solution of this Fokker Planck equation $\frac{\partial p}{\partial \lambda} = -\operatorname{div}(pf) + \frac{1}{2}\nabla_x^T(pQ)\nabla_x$. This is a more general situation that can describe the exact flow EDH 1.2, stochastic flow with fixed Q, the diagnostic noise flow, and the approximate flow.

Denote $p(x) = p_x(x|z), g(x) = p_x(x)$ and $h(x) = p_z(z|x)$, according to the Bayes' Theorem, the posterior conditional density function of x for a given measurement z is $p_x(x|z) = \frac{p_x(x)p_z(z|x)}{p_z(z)}$, in which $p_z(z) = \int_x p_z(z|x)p_x(x)dx$ is the normalization factor.

As the particle flow filters particle flows can be defined through homotopy, [6] proposes a new conditional probability density $p(x, \lambda) = \frac{g(x)h^{\lambda}(x)}{c(\lambda)}$, and the log transformation of it is

$$\log p(x, \lambda) = \log g(x) + \lambda \log h(x) - \log c(\lambda), \tag{7}$$

where (λ) is the normalization factor so that $p(x, \lambda)$ remains a *pdf*, $p(x, 0) = g(x)$ is the density function of the prior distribution and $p(x, 1) = p(x)$ is that of the posterior distribution. Solving the stiffness ODE by homotopy-log transform, and 13 proposes an optimal control method to calculate the optimal $\beta^*(\lambda)$ rather than using a straight λ in the homotopy transform.

The particle flow $x(\lambda)$ follows the below equation, which can be derived from the Fokker Planck equation:

$$\nabla_x \log h = -\left(\nabla_x \nabla_x^T \log p\right) f - \nabla_x \operatorname{div}(f) - \left(\nabla_x f\right)^T \left(\nabla_x \log p\right) + \nabla_x \left[\frac{1}{2p} \nabla_x^T (pQ) \nabla_x\right], \tag{8}$$

[7] proposes an optimal control model for Stiffness Mitigation, in which minimize the condition number of the Hessian matrix of the posterior density function:

$$\begin{aligned}
&\frac{d\beta}{d\lambda} = u(\lambda), \\
&\beta(0) = 0, \beta(1) = 1, \\
&J(\beta, u) = \int_0^1 \left[\frac{1}{2}u^2 + \mu \kappa_\nu(M)\right] d\lambda,
\end{aligned}$$

where $M = -\nabla_x \nabla_x^T \log p = -\nabla_x \nabla_x^T \log p_0 - \beta \nabla_x \nabla_x^T \log h$.

By applying the Pontryagin maximum principle, the optimization problem become a Hamiltonian system, which can be solved by a one-dimensional second order two point boundary value problem (TPBVP),

$$\begin{aligned}
&\frac{d^2\beta^*}{d\lambda^2} = \mu \left.\frac{\partial \kappa_\nu(M)}{\partial \beta}\right|_{\beta = \beta^*}, \\
&\beta^*(0) = 0, \beta^*(1) = 1.
\end{aligned}$$

In the experiment, the author use shooting method to solve the TPBVP, and the root finding process in the shooting method is the bisection method. The solution is named **optimal homotopy** $\beta^*(\lambda)$.

The replication of case in the Section 4 in [7] is shown in the Fig. 13. Experiment shows that $J_{Linear} = 1.6781, J_{Optimal} = 1.5155$.

b) Repeating part 1, if we use the optimized particle flow of Dai(22) [7] as the proposal for the particle flow particle filter of Li(17)[19], does it improve the result of the LEDH particle flow particle filter of Li(17)[19]

According to the [6], [19] models the particle flow by the analytical solution of the ODE, it is a special case in [6], thus the latter one should be able to improve the PF-PF (LEDH).

In my experiment on the Case 2, the optimized particle flow of [7] improves PF-PF (LEDH) 14 a little. It is solved by the shooting method combined with Newton-Raphson (the default method of *scipy.integrate.solve_bvp*()) rather than the Bisection method in the paper. The average RMSE (Avg RMSE) of the straight λ (baseline) is 13.74, and the Avg RMSE of the optimal $\beta^*(\lambda)$ is 13.50. However, the reimplementation of [7] is not close enough to the result in the paper enough, and the implementaion is not precise, it should had been able to further tuned and improved.

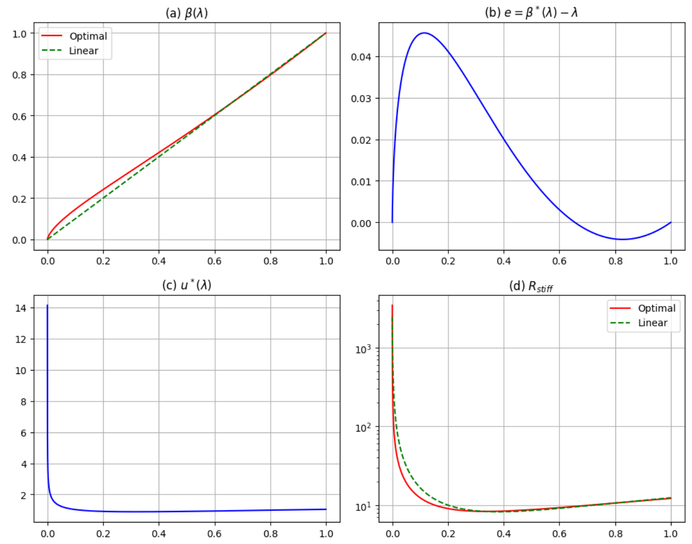

Figure 13: Replication of the experiment [7].

Case 2. This is a target tracking model.

Linear dynamic equation: $x_{k+1} = Fx_k + w_k$, $w_k \sim \mathcal{N}(0, 0.2^2 \cdot I_4)$

Nonlinear-bearing model as the measurement: $z_k = h(x_k) + v_k$, where the oservation function $(h(x))$ is

$$h(x_k) = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} \text{atan2}(p_y - y_{s1},\ p_x - x_{s1}) \\ \text{atan2}(p_y - y_{s2},\ p_x - x_{s2}) \end{bmatrix}$$

and the measurement noise is $v_k \sim \mathcal{N}(0, R), R = \begin{bmatrix} 0.0005 & 0 \\ 0 & 0.0005 \end{bmatrix}$. Here, $(x_{s1}, y_{s1}) = (-150, 0)$ and $(x_{s2}, y_{s2}) = (150, 0)$.

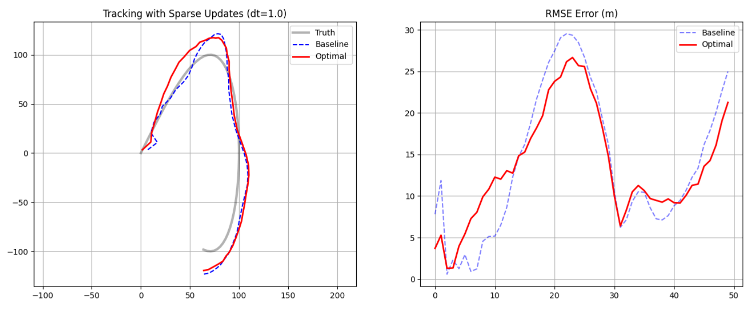

Figure 14: PF-PF (LEDH) by the optimal homotopy $\beta^*(\lambda)$.

2.2 Differentiable particle filter with OT resampling

2.2.1 Implement a differentiable PF following Corenflos(21) and Chen(23)

a) Start with soft-resampling (mixture with uniform) to enable backpropagation through weights. This method is useful to ensure your training loop (backpropagation, optimizer, loss function) is working.

b) Upgrade to entropy-regularized optimal transport (OT) resampling with Sinkhorn algorithm. Tune the regularization parameter and iterations for bias-variance-speed trade-off.

Remark 3. The answers to these two questions are combined.

Let the current particle set at time t be described by locations x_t^i and normalized weights w_t^i for $i = 1 \dots N$. The goal of resampling is to transform the set of weighted particles (the posterior) into a set of unweighted (uniform) particles:

Source Distribution (μ): The empirical measure of the current weighted particles:

$$\mu = \sum_{i=1}^{N} w_t^i \delta_{x_t^i}$$

Target Distribution (ν): We want the resampled particles to have uniform weights ($1/N$):

$$\nu = \sum_{j=1}^{N} \frac{1}{N} \delta_{\tilde{x}_t^j}$$

In a standard Particle Filter (PF), the resampling step is the computational bottleneck for differentiability. Standard resampling is to select N particles from the current set based on their weights (e.g., Multinomial or Systematic resampling). This is a discrete and stochastic operation (selecting indices). Therefore, the differentiable resampling methods are needed. Soft-resampling and OT-resampling are discussed below.

Soft-Resampling Process [4]:

1. Compute importance multinomial distribution by a linear interpolation between the multinomial distribution of the original weighted particles $\text{Mult}(W_t^i)$ and the multinomial distribution with equal weights $\text{Mult}(\frac{1}{N_p})$: $\tilde{W}_t^i = \alpha W_t^i + (1-\alpha)\frac{1}{N_p}$, where $W_t^i = \frac{w_t^i}{\sum_{j=1}^{N_p} w_t^j}$ is the normalized original particles' weights;

2. The indices A_t of selected particles are sampled from the importance multinomial distribution: $\text{Mult}(\tilde{W}_t)$.

3. Soft resampling updates particles' weights to correct the bias caused by sampling from the importance distribution: $w_{new}^i = W_t^{A_t^i} / \tilde{W}_t^{A_t^i}$.

However, soft resampling still relies on sampling from multinomial distributions, thus it does not change the discrete nature of the resampling step. The non-differentiable part of resampling is ignored in soft resampling [5].

OT resampling (Iterative Sinkhorn Algorithm) [5]:

The OT resampling methods seek to find a transport map that moves the particles from the weighted distribution to the uniform distribution while minimizing a predefined 'cost function'.

1. Mechanism: Discrete Optimal Transport (Sinkhorn-Knopp).

This method solves an optimization problem to find a "soft" transport plan P that moves the mass of the particles x_t (weighted by w_t) to a uniform distribution (weights $1/N$) with minimum effort.

2. Mathematical Formulations:

Cost Matrix: squared Euclidean distance between all particle pairs.

$$C_{ij} = \|x^i - x^j\|^2$$

Entropic Regularized OT: find matrix P to minimize:

$$\min_P \sum_{i,j} P_{ij} C_{ij} - \epsilon H(P)$$
$$\text{Subject to: } \sum_j P_{ij} = w^i \text{ and } \sum_i P_{ij} = 1/N.$$

Sinkhorn Algorithm:

Initialization:

Start with $u = [1, 1, \dots]$ and $v = [1, 1, \dots]$, compute the kernel matrix $K_{ij} = e^{-C_{ij}/\epsilon}$.

Iterations:

Step 1: Row Normalization (Update u)

$$u \leftarrow \frac{w}{Kv}$$

Step 2: Column Normalization (Update v)

$$v \leftarrow \frac{1/N}{K^T u}$$

Transport Plan after convergence:

$$P = \text{diag}(u) \cdot K \cdot \text{diag}(v)$$

Barycentric Mapping:

$$x_{new} = N \cdot P^T x_{old}$$

Remark 4. I apply a stablized sinkhorn iteration in the implementaion, which if performed on the log domain:

Sinkhorn Iterations on log domain (Stabilized version): the code solves this using dual potentials f and g:

$$f_i \leftarrow \log(w^i) - \text{LogSumExp}_j(g_j - C_{ij}/\epsilon)$$
$$g_j \leftarrow \log(1/N) - \text{LogSumExp}_i(f_i - C_{ij}/\epsilon)$$

Barycentric Mapping: the particles are moved according to the plan P.

$$x^i_{\text{new}} = N \sum_j P_{ij} x^j$$

The OT resampling is a differentiable resampling, and it preserves particle diversity better than Standard PF. However, it is extremely slow ($O(N^2)$) due to the inner optimization loop at every time step.

The regularization tuning of OT resampling method based on variance-bias-speed trade-off is shown in Tab. 4. When $\epsilon = 0.10$ and the sinkhorn iteration step is 10, the l_2 transport cost $\|x_{\text{new}} - x_{\text{old}}\|_2$ between the new sampled particles and the old particles is 0.6524, the variance ratio (New Variance / Old Variance) is close to 0.8340 (the closer to 1 is the better) with a relatively fast speed 8.42 ms. Even the parameter pair (0.01,20) gives a little lower variance ratio, the bias is nearly 0.1 higher than (0.1,10). Although the parameter pair (0.01,50) achieves the best variance-bias trade-off, the speed is nearly 4.5 times lower than the selected (0.10,10) pair's.

DPF experiment on Example 3 between soft-resampler and OT-resampler are compared in Fig. 15, Tab. 4. The DPF(OT) achieves lower RMSE and higher ESS.

(ϵ,Iters)	Bias (Transport distance)	Var Ratio	Speed(ms)
(1.0, 5)	0.7691	0.2046	23.83
(0.5, 10)	0.7694	0.4207	5.61
(0.1, 10)$*$	0.6524	0.8351	8.42
(0.1, 20)	0.7468	0.8340	9.93
(0.01, 50)	0.4696	0.9606	35.59

Table 2: Regularization parameter and iterations tuning for bias-variance-speed trade-off (OT-resampler). (0.10, 10) achieves the best trade-off thus it is selected as the set of parameters of the experiments.

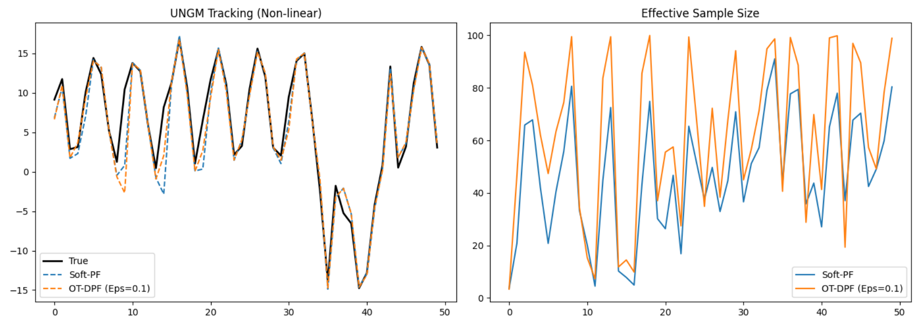

Figure 15: Methods comparison between DPF with OT resampling in the UNGM case.

Method	RMSE	Grad Norm
Soft-PF	2.5012	1.6936
DPF (OT) (Eps=0.1)	2.4047	11.3128

Table 3: Performance and differentiability comparison between soft-resampler and OT-resampler.

2.2.2 There are many other approaches for DPF (see Chen(23)), compare other algorithms(at least two) with OT resampling. Which metrics and diagnostics you can choose? Accuracy? Differentiability? Efficiency?

Case 3. Consider a LGSSM here:

State Transition Equation:

$$X_n = 0.5X_{n-1} + V_n,$$

where process noise $V_n \sim \mathcal{N}(0, 0.1^2)$.

Observation:

$$Y_n = HX_n + W_n,$$

where the measurement noise $W_n \sim \mathcal{N}(0, 0.1^2)$

Example 3. Section 3.1 in [1]: A non-linear state space model

The SSM model names univariate nonstationary growth model (UNGM) [11]:

$$X_n = \frac{X_{n-1}}{2} + 25\frac{X_{n-1}}{1+X_{n-1}^2} + 8\cos(1.2n) + V_n$$
$$Y_n = \frac{X_n^2}{20} + W_n,$$

where $X_1 = 0, V_n \sim \text{IID}\ \mathcal{N}\left(0, \sigma_V^2\right)$ and $W_n \sim \text{IID}\ \mathcal{N}\left(0, \sigma_W^2\right)$. We generated the observations $y_{1:50}$ according to this modelwith $\sigma_V^2 = 1\sigma_W^2 = 1$.

We compare several DPF with OT resampling, including DPF (OT), Neural map based on OT, PFNet (OT), normalizing flow based on OT. There are also some other methods included here, such as the standard particle filter (PF), Neural map based on soft-resampling, PFNet based on soft-resampling (SPF in [4]) and the NF (OT) based on OT; see Fig. 16 and 17.

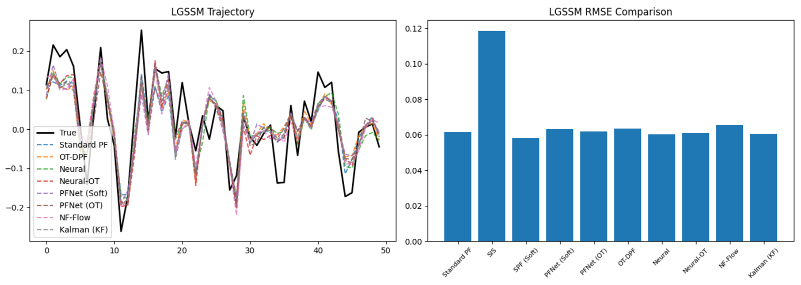

Figure 16: Methods comparison between DPF with OT resampling in the LGSSM model.

Bonus 1. HMC with invertible flows and differentiable resampling

Background: Hamiltonian Monte Carlo is usually a much better method than Metropolis Hasting algorithm for MCMC as discussed in [20]. You can find many useful functions in Tensorflow Probability and the experimental library. Unfortunately particle filters are not naturally differentiable, limiting their use in gradient-based Bayesian inference (e.g. HMC).

a) Consider the problem shown in section 3.1 of [1] (equation 14 and 15). Please estimate the model using the invertible PF-PF of Li(17)[19].

Method	Acc (RMSE)	Diff (Grad)	Eff (Time ms)
Standard PF	0.0616	-	28.64
SPF (Soft)	0.0582	0.0098	35.61
PFNet (Soft)	0.0631	0.0096	38.82
PFNet (OT)	0.0618	0.0102	164.30
DPF (OT)	0.0635	0.0102	189.23
Neural	0.0602	0.0536	115.35
Neural-OT	0.0608	0.0109	279.07
NF-Flow	0.0655	0.0115	255.15
Kalman (KF)	0.0605	0.0101	24.92

Table 4: Results comparison between different DPF methods and some other filters in the LGSSM experiment.

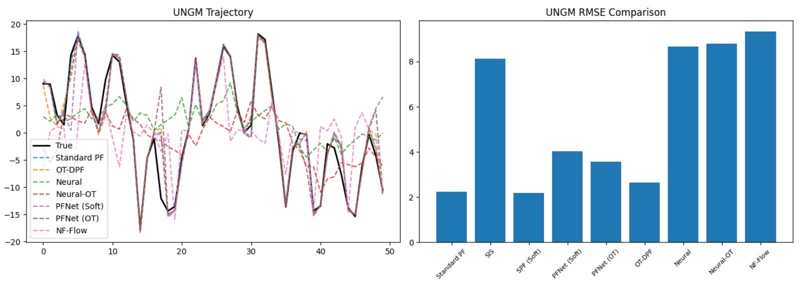

Figure 17: Methods comparison between DPF with OT resampling in the UNGM model.

Method	Acc (RMSE)	Diff (Grad)	Eff (Time ms)
Standard PF	2.2268	0.3853	34.02
SPF (Soft)	2.1992	1.5522	40.70
PFNet (Soft)	4.0282	4.7121	40.51
PFNet (OT)	3.5633	4.0506	170.06
DPF (OT)	2.6580	20.2168	165.14
Neural	8.6571	6.0819	49.75
Neural-OT	8.8051	2884916.2500	146.54
NF-Flow	9.3413	35754.3789	142.88

Table 5: Results comparison between different DPF methods and some other filters in the UNGM experiment.

This problem is illustrated in Ex. 3: Consider the SSM

$$X_n = \frac{X_{n-1}}{2} + 25\frac{X_{n-1}}{1+X_{n-1}^2} + 8\cos(1.2n) + V_n$$

$$Y_n = \frac{X_n^2}{20} + W_n,$$

where $X_1 = 0.1, V_n \sim$ IID $\mathcal{N}\left(0, \sigma_V^2\right)$ and $W_n \sim$ IID $\mathcal{N}\left(0, \sigma_W^2\right)$. We generated the observations $y_{1:100}$ according to this model with $\sigma_V^2 = 10$ and $\sigma_W^2 = 1$.

We generated the observations $y_{1:100}$ according to this model (eq. 14 and 15) with $\sigma_V^2 = \sigma_W^2 = 10$.

We estimate this model by the method of [19] is shown in the Fig. 18. Experiment shows that $RMSE = 4.9682$.

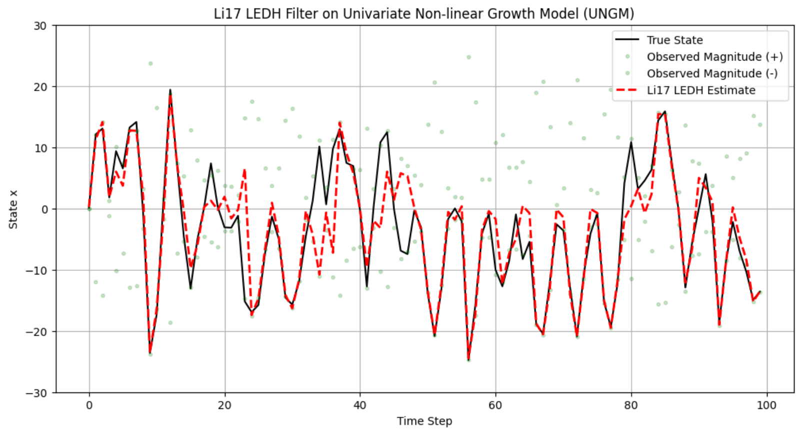

Figure 18: Estimation of PF-PF(LEDH) in [7].

b) Incorporate entropy-regularized OT resampling in Corenflos(21), we can make the particle filter differentiable. Is it possible to apply the algorithm to Li(17)[19] and apply HMC to the problem of the simple nonlinear state space model of section 3.1 shown in part a. of this question? Compare HMC with PMMH in terms of acceptance, chain ESS, runtime, and others.

The comparison between DPF-HMC with particle Gibbs in the nonlinear example of section3.1 in [1].

The results comparison between the DPF-HMC-Li17 and PMMH is shown in the Fig. 19.

c) Discuss advantages and challenges you encountered: differentiability-bias trade-off, OT regularization effects, gradient stability and variance, and others.

HMC has OT regularization, and it is a gradient-available method, while the PMMH uses purely particle filter that is not differentiable. Therefore, the speed of HMC-OT has the potential to be further improved by using neural network to replace the OT-resampling step.

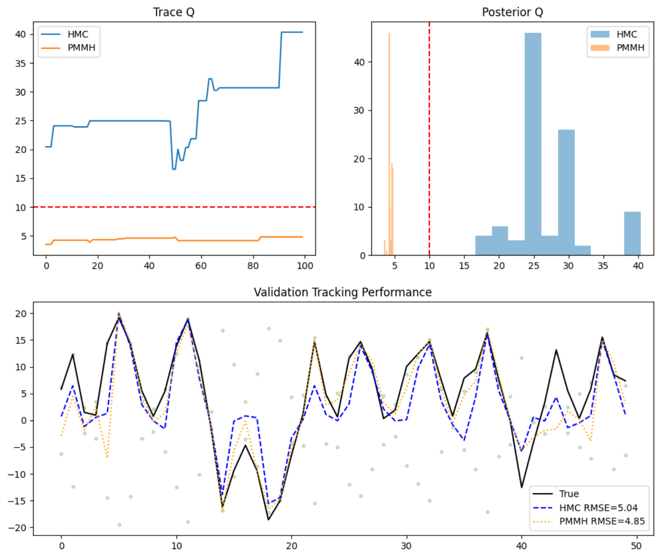

Figure 19: Comparison between HMC-DPF and PMMH.

Bonus 2. Neural Acceleration of OT resampling

Background: Sinkhorn-based OT resampling can be costly. Solving repeatedly the optimal transport problem in bonus question 1 can be very time consuming. A natural idea is to use neural networks to approximate or accelerate optimization steps in OT.

a) Can we use the idea in Chaudhari(25) [3] to avoid applying the Sinkhorn algorithm repeatedly? What additional input variables such as the model parameters and other necessary variables must we give to the neural network so that we can avoid learning the solution neural network repeatedly?

[3] avoid applying the Sinkhorn algorithm repeatedly using neural network as the Monge-Ampère Map, which learnes deterministic transport that is the optimal transport mapping, name it **GradNet OT**: $T(x) = \nabla_x \psi(x)$.

In the network, ψ functions is the convex (guaranteed by the ICNN structure) in this work, the network is searching to find the optimal map. Besides, it feeds time and state together into the convex network:

$$\psi(x,t) = \text{ICNN}(x,t)$$

The structure of ICNN means: a network $f(x)$ is convex with respect to x if the weights connected to the recursive path are non-negative.

$$z_{k+1} = \sigma(W_z^{(k)} z_k + W_x^{(k)} x + b_k),$$

where $W_z^{(k)} \geq 0$ are weights transforming the hidden state z must be non-negative, and σ is the activation function must be convex and non-decreasing (e.g., ReLU or Softplus).

PINO Loss (Physics-Informed): The objective function used in the code is:

$$\mathcal{L} = \mathcal{L}_{\text{Likelihood}} + \lambda_1 \mathcal{L}_{\text{Entropy}} + \lambda_2 \mathcal{L}_{\text{Reg}}.$$

This loss function will be illustrated at the next part.

b) The algorithm, as shown in equation 4 of the paper, solves a partial differential equation. Does the theory of neural operator (see Jha(25) [14]) apply? If it does, what method or architecture can we leverage to improve the training of the operator?

[14] use neural operator instead of the normal neural network mapping like multi-layer perceptron (MLP). We can design a DeepONet OT: $T(x) = \nabla_x \psi_{\text{DeepONet}}(x)$, which has the same mathematical identity as the the GradNet OT that satisfy Brenier's Theorem, and the unique optimal transport map for the squared Euclidean cost is the gradient of a convex function. Since both ψ functions in GradNet OT and DeepONet OT are convex (guaranteed by the ICNN structure), both networks are searching the exact same function space to find the optimal map.

DeepONet OT Architecture

DeepONet assumes the potential is a dot product of time-dependent weights and spatial basis functions:

$$\psi(x,t) = \sum_k \underbrace{w_k(t)}_{\text{Branch}} \cdot \underbrace{\phi_k(x)}_{\text{Trunk}}$$

Loss function: Physics-Informed Neural Operator Learning.

This combines the architecture of DeepONet with the physics constraints of Optimal Transport (Monge-Ampère). It learns a mapping from the functional coefficients (Context) to the Transport Operator.

Mathematical Formulation: Architecture (Branch & Trunk):

Trunk (τ): An ICNN (Input Convex Neural Network) that learns the geometric basis of transport.

Branch (b): Encodes the dynamic environment (Observations y_t, Time t).

Potential: The transport potential ψ is the dot product:

$$\psi(x,t) = \sum_{k=1}^{K} b_k(t) \cdot \tau_k(x)$$

Transport Map: The map is the gradient of this potential with respect to x (Brenier's Theorem).

$$T(x,t) = x + \nabla_x \psi(x,t)$$

(Note: We add x (Identity) for stability, learning the displacement).

Same PINO Loss (Physics-Informed) as the GradNet OT is used here:

$$\mathcal{L} = \underbrace{-\log p(y_t|T(x))}_{\text{Likelihood}} - \lambda_1 \underbrace{\log \det(\nabla T)}_{\text{Monge-Ampère (Entropy)}} + \lambda_2 \underbrace{\|T(x) - x\|^2}_{\text{Transport Cost}}$$

The likelihood pushes particles toward high-probability regions (Data fidelity).

Entropy/Monge-Ampère prevents ”Mode Collapse.” Maximizing the determinant of the Jacobian keeps the particles spread out.

Transport cost is the regularization. It can ensures that we find the Optimal map (minimum movement) among all valid maps.

The results comparison between the Designed DeepONet-OT and the other methods including the standard PF, PDF (OT), GradNet OT [3] is shown in the Fig. 20. The methods time span is $T = 50$, particle numbers are 1000, training epochs of the GradNet-OT and DeepONet-OT are 150.

2.2.2 shows that DeepONet-OT performs compatative, which has similar RMSE to the DPF-OT with parameters ($\epsilon = 0.1$, Iteration= 10), but a much faster inference speed. GradNet-OT is slightly faster than the DeepONet OT, but the prediction result is bad with high RMSE. The standard PF performs the best so far, but the neural networks-based methods have the potential to be improved.

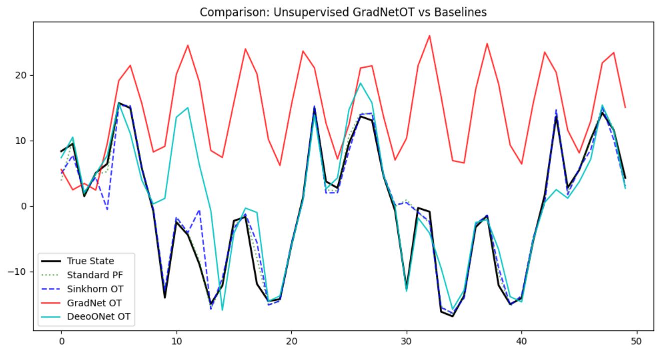

Figure 20: Comparison between the designed DeepONet and the other methods.

Method	RMSE	Time (ms)
Standard PF	2.2069	2.93
DPF-OT (0.1,10)	2.7636	46.22
GradNet OT	18.1886	7.76
DeepONet OT	2.7709	9.34

Table 6: Performance and differentiability comparison between designed DeepONet OT and the other DPF methods.

Bonus 3. Particle-Flow Inference for Neural State-Space Models

Background: Background: State-Space LSTM(SSL) models combine LSTM transition with probabilistic emissions and use Particle Gibbs(PG) for joint posterior sampling, see Zheng(17)[22].

a) Compare your DPF-HMC with particle Gibbs on example 1 and 2 in Zheng(17) [22]. What metrics can you use for comparison?

The core of the paper [22] is Algorithm 1 (PGAS). I replicate this algorithm of the UNGM 21. Deside the network LSTM_θ,

$$\text{Transition:} \quad z_t = \mathrm{LSTM}_\theta(z_{t-1}, h_{t-1}) + \epsilon_t$$

$$\text{Emission:} \quad y_t = \frac{z_t^2}{20} + \nu_t \quad \text{(We assumed this was known/fixed)}$$

In the code, the M-Step tried to make the LSTM weights θ approximate the complex dynamics

$$\text{Transition:} \quad z_t = 0.5 z_{t-1} + 25\frac{z_{t-1}}{1 + z_{t-1}^2} + 8\cos(1.2(t-1)) + \epsilon_t.$$

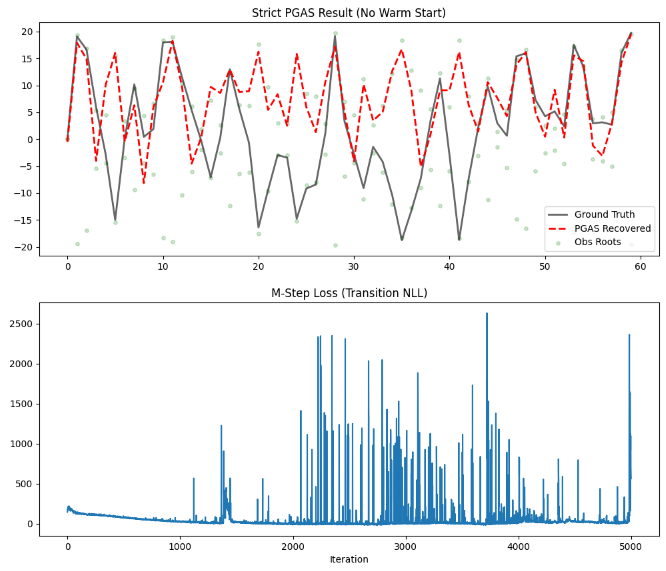

Figure 21: Result on particle gibbs Sampler.

b) Summarize your final DPF method and write down the entire pipeline in detail. Which type of particle flow did you use? How exactly did you perform the resampling? What modifications did you make yourself? What techniques did you use? Do you think the current version is best? If you have more 3 months, how would you further optimize it to achieve a better DPF?

According to the work flow of the previous problems, there are two final DPF methods. One is a kind of differentiable particle filter method, which uses the idea of OT resampler to achieve the differentiable operation, and the iterative operation of the Sinkhorn algorithm of OT is surrogated by neural Monge-Ampère map network. The particle flow is an OT flow. Another is the DPF that follow the PF-PF (LEDH) [19] using HMC algorithm, the particle flow is not OT but Fokker Planck equation/continuity equation. The currect works have many improvement space, for example, combining these two methods is one of the improvement direction. Various neural networks/ neural operators can be chosen, the hyper-parameters can be tuned, different sampling

References

[1] Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. "Particle markov chain monte carlo methods". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 72.3 (2010), pp. 269–342.

[2] M Sanjeev Arulampalam et al. "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". In: *IEEE Transactions on signal processing* 50.2 (2002), pp. 174–188.

[3] Shreyas Chaudhari, Srinivasa Pranav, and José MF Moura. "GradNetOT: Learning Optimal Transport Maps with GradNets". In: *arXiv preprint arXiv:2507.13191* (2025).

[4] Xiongjie Chen and Yunpeng Li. "An overview of differentiable particle filters for data-adaptive sequential Bayesian inference". In: *arXiv preprint arXiv:2302.09639* (2023).

[5] Adrien Corenflos et al. "Differentiable particle filtering via entropy-regularized optimal transport". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 2100–2111.

[6] Liyi Dai and Fred Daum. "A new parameterized family of stochastic particle flow filters". In: *arXiv preprint arXiv:2103.09676* (2021).

[7] Liyi Dai and Fred Daum. "Stiffness mitigation in stochastic particle flow filters". In: *IEEE Transactions on Aerospace and Electronic Systems* 58.4 (2022), pp. 3563–3577.

[8] Fred Daum and Jim Huang. "Particle degeneracy: root cause and solution". In: *Signal Processing, Sensor Fusion, and Target Recognition XX*. Vol. 8050. SPIE. 2011, pp. 367–377.

[9] Fred Daum, Jim Huang, and Arjang Noushin. "Exact particle flow for nonlinear filters". In: *Signal processing, sensor fusion, and target recognition XIX*. Vol. 7697. SPIE. 2010, pp. 92–110.

[10] Alexandros Evangelidis and David Parker. "Quantitative verification of numerical stability for Kalman filters". In: *International Symposium on Formal Methods*. Springer. 2019, pp. 425–441.

[11] Neil J Gordon, David J Salmond, and Adrian FM Smith. "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". In: *IEE proceedings F (radar and signal processing)*. Vol. 140. 2. IET. 1993, pp. 107–113.

[12] Fredrik Gustafsson and Gustaf Hendeby. "Some relations between extended and unscented Kalman filters". In: *IEEE Transactions on Signal Processing* 60.2 (2011), pp. 545–555.

[13] Chih-Chi Hu and Peter Jan Van Leeuwen. "A particle flow filter for high-dimensional system applications". In: *Quarterly Journal of the Royal Meteorological Society* 147.737 (2021), pp. 2352–2374.

[14] Prashant K Jha. "From Theory to Application: A Practical Introduction to Neural Operators in Scientific Computing". In: *arXiv preprint arXiv:2503.05598* (2025).

[15] Adam Johansen. “A tutorial on particle filtering and smoothing: Fifteen years later”. In: (2009).

[16] Simon J Julier and Jeffrey K Uhlmann. “New extension of the Kalman filter to nonlinear systems”. In: *Signal processing, sensor fusion, and target recognition VI*. Vol. 3068. Spie. 1997, pp. 182–193.

[17] Simon J Julier and Jeffrey K Uhlmann. “Unscented filtering and nonlinear estimation”. In: *Proceedings of the IEEE* 92.3 (2004), pp. 401–422.

[18] Rudolph Emil Kalman. “A new approach to linear filtering and prediction problems”. In: (1960).

[19] Yunpeng Li and Mark Coates. “Particle filtering with invertible particle flow”. In: *IEEE Transactions on Signal Processing* 65.15 (2017), pp. 4102–4116.

[20] Radford M Neal et al. “MCMC using Hamiltonian dynamics”. In: *Handbook of markov chain monte carlo* 2.11 (2011), p. 2.

[21] Rudolph Van Der Merwe and Eric A Wan. “The square-root unscented Kalman filter for state and parameter-estimation”. In: *2001 IEEE international conference on acoustics, speech, and signal processing. Proceedings (Cat. No. 01CH37221)*. Vol. 6. IEEE. 2001, pp. 3461–3464.

[22] Xun Zheng et al. “State space LSTM models with particle MCMC inference”. In: *arXiv preprint arXiv:1711.11179* (2017).