2026 Machine Learning Center of Excellence Summer Associate – Time Series & Reinforcement Learning Internship

Question 2: Particle Flow Filter and Differentiable Particle Filter

Chunchao Ma

Report Structure

This report is organized into four major parts, following the format required by the assignment. You may navigate to each part using the clickable links below:

- **A. A literature review of potential methods for solving the problem** Jump to Section A
- **B. Propose your choice of methods and the reason for the choice** Jump to Section B
- **C. Present the answers to the question in clear and understandable language** Jump to Section C
- **D. Testing plans and testing results that show your implementation is correct and your results are valid** Jump to Section D

A. A literature review of potential methods for solving the problem

Since I have not yet studied or implemented the methods in Part 2 in sufficient depth, this submission focuses exclusively on Part 1. The material in Part 2 is therefore not included in order to ensure the technical accuracy and completeness of the presented work.

A short literature review

Modern financial systems generate large volumes of sequential data, including asset prices, volatility indicators, and transaction records. Extracting the underlying latent dynamics of such systems is central to risk management, portfolio optimization, and anomaly detection. State Space Models (SSMs) provide a principled probabilistic framework for representing these temporal processes.

In linear–Gaussian SSMs, the Kalman Filter (KF) offers an optimal minimum-variance estimator (Pei et al., 2019; kal, 2025). Its closed-form prediction and update equations make it efficient and widely applied in state estimation in linear systems. However, numerical issues may arise in practice: the covariance matrix can lose symmetry or positive definiteness due to floating-point errors. The Joseph-stabilized covariance update is commonly used to improve numerical robustness (kal, 2025). Recent work also highlights the need to formally verify the robustness of Kalman filter implementations, using techniques such as probabilistic model checking (Evangelidis and Parker, 2019).

Real-world financial systems are often highly nonlinear and noisy, creating challenges for classical state-space estimation methods. The Extended Kalman Filter (EKF) addresses nonlinear dynamics by locally linearizing the system around current estimates, enabling approximate Gaussian filtering in nonlinear settings (Pei et al., 2019). However, its reliance on first-order Taylor expansions can lead to significant errors when the true system exhibits strong curvature or when linearization is unstable. The Unscented Kalman Filter (UKF) improves upon EKF by propagating a set of deterministically chosen sigma points through the nonlinear transformation, capturing mean and covariance more accurately without explicit linearization (Pei et al., 2019; Wan and Van Der Merwe, 2000). Despite these improvements, both EKF and UKF may fail in the presence of severe nonlinearities – conditions commonly encountered in complex financial environments.

Particle filters offer an alternative by representing the posterior with a set of weighted samples rather than relying on Gaussian approximations or local linearization (Doucet and Johansen, 2009). This allows them to capture strong nonlinearities, heavy tails, and multimodality, albeit at substantially higher computational cost. Through importance sampling, the Particle Filter (PF) can approximate general nonlinear and non-Gaussian distributions. However, PFs suffer from weight degeneracy and poor scalability in high-dimensional problems—serious limitations for many financial applications (Doucet and Johansen, 2009). To mitigate these issues, several particle flow filtering frameworks have been proposed (Daum

et al., 2010; Daum and Huang, 2011; Li and Coates, 2017).

The key idea of particle flow is to replace the discrete Bayesian weight update with a continuous flow of probability density governed by a partial differential equation. This density flow induces a deterministic transport of particles from the prior to the posterior distribution, thereby avoiding weight collapse and resampling (Daum et al., 2010). As a result, particle flow mitigates particle degeneracy and remains effective in high-dimensional filtering problems (Daum and Huang, 2011). In the special case where both the prior density and the likelihood are Gaussian, the particle flow equation admits an exact analytic solution derived from the Fokker–Planck formulation using the method of separation of variables. This results in a closed-form linear particle flow ordinary differential equation that transports particles exactly from the prior to the posterior; this is known as the Exact Flow Daum-Huang Filter (EDH) (Daum et al., 2010).

Building on this theory, Ding and Coates (2012) provide a detailed implementation of the EDH filter and introduce its locally exact extension (LEDH), motivated by the limitations of global linearization in EDH (Ding and Coates, 2012). In LEDH, the particle flow parameters are computed using particle-wise linearizations, allowing the flow dynamics to adapt locally to the geometry of the likelihood.

While EDH and LEDH provide effective deterministic mechanisms for transporting particles toward the posterior, discretization and linearization errors imply that the resulting particles are not exact posterior samples. Li and Coates (2017) address this limitation by embedding deterministic particle flows within a standard particle filtering framework, in which the flow defines an invertible deterministic proposal mapping and the associated Jacobian is used to compute importance weights. Importance weighting and resampling are then applied to correct for approximation errors, restoring the theoretical guarantees of sequential Monte Carlo while retaining the robustness of particle flow in high-dimensional and highly informative filtering problems.

In addition to particle flow–based filtering methods, another strand of research combines particle filtering with optimal transport through kernel-based mappings in a reproducing kernel Hilbert space (Pulido and van Leeuwen, 2019). Pulido and van Leeuwen (2019) introduce the Mapping Particle Filter (MPF), which deterministically transports particles from the prior to the posterior by following a kernel-embedded gradient flow that minimizes the Kullback–Leibler divergence. This approach avoids linearization-based particle flows and largely eliminates the need for resampling. Building on this framework, Hu and Van Leeuwen (2021) extend the MPF to high-dimensional systems by introducing matrix-valued kernels, which prevent particle collapse in sparsely observed settings and enable robust assimilation of nonlinear and multimodal observations.

B. Propose your choice of methods and the reason for the choice

For the JP Morgan MLCOE TSRL 2026 Internship Question 2 on the Particle Flow Filter and the Differentiable Particle Filter, the required methods are already specified within the question. Therefore, our main task is simply to implement the methods as stated, and no additional proposal or justification of alternative methods is necessary.

C. Present the answers to the question in clear and understandable language

Part 1: From classical filters to particle flows

Build up from basic filtering for SSMs to motivate the need for particle flow methods.

1) Warm-up. For those who are not familiar with filtering, do spend some time learning these basics before the next part.

   I) Linear-Gaussian SSM with Kalman Filter

      a) Implement the Kalman filter for a multidimensional linear-Gaussian SSM (do not use `tfp.distributions.LinearGaussianStateSpaceModel`). Use synthetic data from a standard LGSSM; see examples 2 in Doucet and Johansen (2009).

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

   II) 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 Doucet and Johansen (2009)) or nonlinear tracking models (e.g., range–bearing observation model).

      b) Implement the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF). Discuss linearization accuracy limits and sigma-point failures under strong nonlinearity.

      c) Implement a standard particle filter for your model (do not use `tfp.experimental.mcmc.particle_filter`). Visualize and discuss issues such as particle degeneracy.

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

2) Deterministic and Kernel Flows

   a) Study the Exact Daum–Huang (EDH) flow and Local Exact Daum–Huang (LEDH) flow (see Daum et al. (2010); Daum and Huang (2011)), and the invertible particle flow particle filter (PF-PF) framework (see Li and Coates (2017)). Replicate the main results in Li and Coates (2017).

   b) Implement the kernel-embedded particle flow filter in an RKHS following Hu and Van Leeuwen (2021). Then compare the scalar kernel and diagonal matrix-valued kernel. Use experiments to demonstrate that matrix-valued kernels can prevent collapse of observed-variable marginals in high dimensions; plot similar figures as in Figures 2–3 of Hu and Van Leeuwen (2021).

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

Reply to Question 1, Part 1(a)

"Implement the Kalman filter for a multidimensional linear-Gaussian SSM. (Do not use `tfp.distributions.LinearGaussianStateSpaceModel`). Use synthetic data from a standard LGSSM, see examples 2 in Pei et al. (2019)."

Reply: We have Implement the Kalman filter for a multidimensional linear-Gaussian SSM and use sythetic data form staddard LGSSM as examples 2 in Pei et al. (2019). For more details, see `https://github.com/ChunchaoPeter/ml_simulation_research/tree/features_kf_lgssm/kf_lgssm`.

Based on Pei et al. (2019), we consider a discrete-time linear dynamical system of the form:

$$\mathbf{x}_t = F_t \mathbf{x}_{t-1} + B_t \mathbf{u}_t + \mathbf{w}_t, \qquad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, Q_t), \tag{1}$$

$$\mathbf{z}_t = H_t \mathbf{x}_t + \mathbf{v}_t, \qquad \mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, R_t), \tag{2}$$

where

- $\mathbf{x}_t \in \mathbb{R}^{n_x}$ is the latent (hidden) state vector,
- $\mathbf{z}_t \in \mathbb{R}^{n_z}$ is the observation vector,
- $\mathbf{u}_t$ is an optional control input,
- $\mathbf{w}_t, \mathbf{v}_t$ are mutually independent Gaussian noise terms,
- F_t, B_t, H_t are known model matrices,
- Q_t and R_t are the process and observation covariance matrices.

The Kalman filter provides recursive minimum-variance estimates of $\mathbf{x}_t$ given the observations $\mathbf{z}_{1:t}$.

Initialization

$$\hat{\mathbf{x}}_{0|0} = \mathbb{E}[\mathbf{x}_0], \qquad \Sigma_{0|0} = \mathrm{Cov}[\mathbf{x}_0]. \tag{3}$$

Prediction Step

$$\hat{\mathbf{x}}_{t|t-1} = F_t \hat{\mathbf{x}}_{t-1|t-1} + B_t \mathbf{u}_t, \tag{4}$$

$$\Sigma_{t|t-1} = F_t \Sigma_{t-1|t-1} F_t^\top + Q_t. \tag{5}$$

Update Step Innovation (residual):

$$\mathbf{r}_t = \mathbf{z}_t - H_t \hat{\mathbf{x}}_{t|t-1}, \tag{6}$$

$$S_t = H_t \Sigma_{t|t-1} H_t^\top + R_t. \tag{7}$$

Kalman gain:

$$K_t = \Sigma_{t|t-1} H_t^\top S_t^{-1}. \tag{8}$$

Posterior update:

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + K_t \mathbf{r}_t, \tag{9}$$

$$\Sigma_{t|t} = (I - K_t H_t)\Sigma_{t|t-1} \tag{10}$$

We can replace $\Sigma_{t|t} = (I - K_t H_t)\Sigma_{t|t-1}$ with $\Sigma_{t|t} = (I - K_t H_t)\Sigma_{t|t-1}(I - K_t H_t)^\top + K_t R_t K_t^\top$ (Joseph form), as the latter formulation is more robust and ensures that $\Sigma_{t|t}$ remains a positive semi-definite matrix.

The notation $t \mid t-1$ means "at time t given information up to time $t-1$". Thus, $\hat{\mathbf{x}}_{t|t-1}$ is the *predicted* state at time t before observing $\mathbf{z}_t$, while $\hat{\mathbf{x}}_{t|t}$ is the *updated* (filtered) state after incorporating $\mathbf{z}_t$.

Algorithm 1 Kalman Filter for a Linear-Gaussian State-Space Model

Input: $\{F_t, B_t, H_t, Q_t, R_t\}_{t=1}^T$, initial $(\hat{\mathbf{x}}_{0|0}, \Sigma_{0|0})$
for $t = 1$ to T **do**
  Prediction:
  $\hat{\mathbf{x}}_{t|t-1} = F_t \hat{\mathbf{x}}_{t-1|t-1} + B_t \mathbf{u}_t$
  $\Sigma_{t|t-1} = F_t \Sigma_{t-1|t-1} F_t^\top + Q_t$
  Innovation:
  $\mathbf{r}_t = \mathbf{z}_t - H_t \hat{\mathbf{x}}_{t|t-1}$
  $S_t = H_t \Sigma_{t|t-1} H_t^\top + R_t$
  Kalman Gain:
  $K_t = \Sigma_{t|t-1} H_t^\top S_t^{-1}$
  Update:
  $\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + K_t \mathbf{r}_t$
  $\Sigma_{t|t} = (I - K_t H_t)\Sigma_{t|t-1}$ or $\Sigma_{t|t} = (I - K_t H_t)\Sigma_{t|t-1}(I - K_t H_t)^\top + K_t R_t K_t^\top$

Note: In our code implementation, we replace F_t, H_t, Q_t, R_t, B_t with F, H, Q, R, B, assuming they are time-invariant.

Reply to Question 1, Part 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."

Reply:

Regarding **compare filterd means/convarinces**: According to Wikipedia (kal, 2025), the Kalman filter is described as the best possible linear estimator in the minimum mean-square-error sense. It provides an optimal state estimate when (a) the model perfectly

represents the real system, (b) the process noise is white (uncorrelated), and (c) the noise covariances are known exactly. Pei et al. (2019) demonstrated that two key concepts—optimal linear estimation for fusing uncorrelated estimates and best linear unbiased estimation for correlated variables—form the theoretical foundation of Kalman filtering. This perspective clarifies the assumptions underlying the filter's optimality. For a more detailed derivation of the Kalman filter, see Pei et al. (2019).

Regarding using Joseph stabilized covariance updates: I have added the Joseph covariance update to the original code as well.

Regarding condition number: According to Evangelidis and Parker (2019), the **condition number** of a matrix provides an indication of its numerical stability by measuring how sensitive it is to numerical errors and how close it is to being singular. In their study of Kalman filter implementations, the condition number of the estimation-error covariance matrix P^+ is used as a quantitative measure of robustness against numerical instability. It is defined as

$$\kappa(P^+) = \frac{\sigma_{\max}}{\sigma_{\min}},$$

where $\sigma_{\max}$ and $\sigma_{\min}$ denote the largest and smallest singular values of P^+, respectively. A small condition number (close to one) indicates that the matrix is **well-conditioned**, meaning it is far from singular and that numerical computations will remain stable. Conversely, a large condition number implies an **ill-conditioned** matrix, where small rounding or truncation errors can be amplified, potentially leading to the loss of positive-definiteness in P^+ and, consequently, instability in the Kalman filter. Therefore, monitoring the condition number offers a systematic means to assess the numerical reliability of the covariance matrix and to ensure the Kalman filter operates within stable numerical bounds (Evangelidis and Parker, 2019). For a more detailed description of the study *Quantitative Verification of Numerical Stability for Kalman Filters*, refer to Evangelidis and Parker (2019).

This notebook (`https://github.com/ChunchaoPeter/ml_simulation_research/blob/features_kf_lgssm/kf_lgssm/kalman_filter_analysis_optimality_stability.ipynb`) provides a comprehensive analysis of Kalman filter numerical stability for Linear Gaussian State Space Models (LGSSM). We compare two covariance update formulations: the standard form $\Sigma_{t|t} = (I - K_t H_t)\Sigma_{t|t-1}$ and the Joseph stabilized form $\Sigma_{t|t} = (I - K_t H_t)\Sigma_{t|t-1}(I - K_t H_t)^T + K_t R_t K_t^T$. Using an extended time horizon ($T = 100$) and conditioning number analysis following Evangelidis and Parker (2019),we quantify numerical precision loss via $\log_{10}(\kappa)$. In well-conditioned scenarios, both forms produce equivalent results with $\log_{10}(\kappa) \approx 0.13$ digits of precision loss. However, in ill-conditioned cases (observation noise $R = 10^{-8}$, process noise $Q = 2.0$, ratio $Q/R \approx 2 \times 10^8$), the standard form catastrophically fails—the covariance matrix collapses to zero, yielding undefined conditioning numbers—while the Joseph form maintains stability with $\log_{10}(\kappa) = 0$, preserving positive-definiteness and symmetry. This demonstrates the critical importance of the Joseph formulation for robust Kalman filtering in practical applications with high measurement precision or ill-conditioned dynamics.

Reply to Question 1, Part 2(a)

"Design a nonlinear and non-Gaussian SSM; you can use a stochastic volatility model (example 4 in Doucet and Johansen (2009)) or nonlinear tracking models (e.g., range–bearing observation model)."

Reply:

I have developed the Range–Bearing observation model that is a **state-space model** with:

- **Linear motion model** (constant velocity in Cartesian coordinates)
- **Non-linear observation model** (polar coordinates: range and bearing)

The non-linearity in the observation function, combined with additive Gaussian noise, produces a **non-Gaussian posterior distribution**, making this model suitable for testing non-linear filtering algorithms such as Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), and Particle Filter (PF).

State Space: The state at time t is a 4-dimensional vector representing 2D position and velocity:

$$\mathbf{x}_t = \begin{bmatrix} x_t \\ \dot{x}_t \\ y_t \\ \dot{y}_t \end{bmatrix} \in \mathbb{R}^4 \tag{11}$$

where:

- x_t: position in x-direction (horizontal)
- $\dot{x}_t$: velocity in x-direction
- y_t: position in y-direction (vertical)
- $\dot{y}_t$: velocity in y-direction

Motion Model (State Transition): The motion model describes how the state evolves over time using a **linear constant velocity model**:

$$\mathbf{x}_t = A\mathbf{x}_{t-1} + \mathbf{w}_t \tag{12}$$

where $\mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, Q)$ is the **process noise**.

State Transition Matrix:

$$A = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{13}$$

This matrix implements the discrete-time constant velocity model:

$$x_t = x_{t-1} + \Delta t \cdot \dot{x}_{t-1} + w_{x,t} \tag{14}$$
$$\dot{x}_t = \dot{x}_{t-1} + w_{\dot{x},t} \tag{15}$$
$$y_t = y_{t-1} + \Delta t \cdot \dot{y}_{t-1} + w_{y,t} \tag{16}$$
$$\dot{y}_t = \dot{y}_{t-1} + w_{\dot{y},t} \tag{17}$$

Process Noise Covariance:

$$Q = \begin{bmatrix} \sigma_x^2 & 0 & 0 & 0 \\ 0 & \sigma_{\dot{x}}^2 & 0 & 0 \\ 0 & 0 & \sigma_y^2 & 0 \\ 0 & 0 & 0 & \sigma_{\dot{y}}^2 \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{18}$$

where:

- σ_x^2, σ_y^2: variance of position noise
- $\sigma_{\dot{x}}^2, \sigma_{\dot{y}}^2$: variance of velocity noise

Observation Model (Measurement): The observation model is **non-linear**, converting Cartesian state to polar coordinates:

$$\mathbf{z}_t = h(\mathbf{x}_t) + \mathbf{v}_t \tag{19}$$

where $\mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, R)$ is the **measurement noise**.

Observation Function:

$$h(\mathbf{x}_t) = \begin{bmatrix} r_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} \sqrt{x_t^2 + y_t^2} \\ \arctan 2(y_t, x_t) \end{bmatrix} \in \mathbb{R}^2 \tag{20}$$

where:

- r_t: **range** (distance from origin to object), $r_t \geq 0$
- θ_t: **bearing** (angle from positive x-axis), $\theta_t \in [-\pi, \pi]$

Range Component: The range (distance) is computed as:

$$r_t = \sqrt{x_t^2 + y_t^2} \tag{21}$$

Bearing Component: The bearing (angle) uses the arctan 2 function for correct quadrant handling:

$$\theta_t = \arctan 2(y_t, x_t) = \begin{cases} \arctan\left(\frac{y_t}{x_t}\right) & \text{if } x_t > 0 \\ \arctan\left(\frac{y_t}{x_t}\right) + \pi & \text{if } x_t < 0, y_t \geq 0 \\ \arctan\left(\frac{y_t}{x_t}\right) - \pi & \text{if } x_t < 0, y_t < 0 \\ +\frac{\pi}{2} & \text{if } x_t = 0, y_t > 0 \\ -\frac{\pi}{2} & \text{if } x_t = 0, y_t < 0 \\ \text{undefined} & \text{if } x_t = 0, y_t = 0 \end{cases} \tag{22}$$

The arctan 2 function returns angles in the range $(-\pi, \pi]$.

Measurement Noise Covariance:

$$R = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\theta^2 \end{bmatrix} \in \mathbb{R}^{2\times 2} \tag{23}$$

where:

- σ_r^2: variance of range measurement noise
- σ_θ^2: variance of bearing measurement noise (in radians)

Observation Jacobian (for EKF): For the Extended Kalman Filter, we need the **Jacobian matrix** of the observation function:

$$H_t = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_t} = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial \dot{x}} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial \dot{y}} \\ \frac{\partial \theta}{\partial x} & \frac{\partial \theta}{\partial \dot{x}} & \frac{\partial \theta}{\partial y} & \frac{\partial \theta}{\partial \dot{y}} \end{bmatrix} \tag{24}$$

Partial Derivatives for Range: For $r = \sqrt{x^2 + y^2}$:

$$\frac{\partial r}{\partial x} = \frac{x}{\sqrt{x^2+y^2}} = \frac{x}{r} = \cos(\theta) \tag{25}$$

$$\frac{\partial r}{\partial y} = \frac{y}{\sqrt{x^2+y^2}} = \frac{y}{r} = \sin(\theta) \tag{26}$$

$$\frac{\partial r}{\partial \dot{x}} = 0, \quad \frac{\partial r}{\partial \dot{y}} = 0 \tag{27}$$

Partial Derivatives for Bearing: For $\theta = \arctan 2(y, x)$:

$$\frac{\partial \theta}{\partial x} = \frac{-y}{x^2+y^2} = \frac{-y}{r^2} = -\frac{\sin(\theta)}{r} \tag{28}$$

$$\frac{\partial \theta}{\partial y} = \frac{x}{x^2+y^2} = \frac{x}{r^2} = \frac{\cos(\theta)}{r} \tag{29}$$

$$\frac{\partial \theta}{\partial \dot{x}} = 0, \quad \frac{\partial \theta}{\partial \dot{y}} = 0 \tag{30}$$

Complete Jacobian Matrix:

$$H_t = \begin{bmatrix} \cos(\theta_t) & 0 & \sin(\theta_t) & 0 \\ -\frac{\sin(\theta_t)}{r_t} & 0 & \frac{\cos(\theta_t)}{r_t} & 0 \end{bmatrix} \in \mathbb{R}^{2\times 4} \tag{31}$$

The implementation details can be found here:(`https://github.com/ChunchaoPeter/ml_simulation_research/tree/main/ekf_ukf_pf`).

Reply to Question 1, Part 2(b)

"Implement the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF). Discuss linearization accuracy limits and sigma-point failures under strong nonlinearity."

Reply:

First, we consider a nonlinear and non-Gaussian state-space model; the range–bearing observation model above is one example.

State Equation

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t) + \mathbf{w}_t \tag{32}$$

where

- $\mathbf{x}_t \in \mathbb{R}^n$ is the state vector at time t,
- $f(\cdot)$ is the (potentially non-linear) state transition function,
- $\mathbf{u}_t$ is the control input (optional),
- $\mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, Q_t)$ is the process noise.

Observation Equation

$$\mathbf{z}_t = h(\mathbf{x}_t) + \mathbf{v}_t \tag{33}$$

where

- $\mathbf{z}_t \in \mathbb{R}^m$ is the observation vector at time t,
- $h(\cdot)$ is the (potentially non-linear) observation function,
- $\mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, R_t)$ is the measurement noise.

Noise Covariances

$$Q_t \in \mathbb{R}^{n \times n} \quad \text{(process noise covariance)}, \tag{34}$$

$$R_t \in \mathbb{R}^{m \times m} \quad \text{(measurement noise covariance)}. \tag{35}$$

Initial Distribution

$$\mathbf{x}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0) \tag{36}$$

Extended Kalman Filter (EKF): The Extended Kalman Filter (Pei et al., 2019) handles non-linearity by **linearization** using a first-order Taylor expansion around the current estimate.

Linearization via Jacobians: The EKF linearizes the non-linear functions using Jacobian matrices.

State Transition Jacobian:

$$F_t = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_{t-1|t-1}} \in \mathbb{R}^{n\times n} \tag{37}$$

This Jacobian is evaluated at the previous filtered state $\mathbf{x}_{t-1|t-1}$.

Observation Jacobian

$$H_t = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_{t|t-1}} \in \mathbb{R}^{m\times n} \tag{38}$$

This Jacobian is evaluated at the predicted state $\mathbf{x}_{t|t-1}$.

The main difference between the Kalman Filter and the Extended Kalman Filter is that the Extended Kalman Filter uses the Jacobian matrix to linearize nonlinear functions. Everything else is similar, and the pseudocode is shown below.

Algorithm 2 Extended Kalman Filter (EKF)

Input: non-linear functions f, h, noise covariances $\{Q_t, R_t\}_{t=1}^T$, controls $\{\mathbf{u}_t\}_{t=1}^T$, initial $(\hat{\mathbf{x}}_{0|0}, \Sigma_{0|0})$
for $t = 1$ to T **do**
  Prediction:
  $F_t = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{t-1|t-1}}$
  $\hat{\mathbf{x}}_{t|t-1} = f(\hat{\mathbf{x}}_{t-1|t-1}, \mathbf{u}_t)$
  $\Sigma_{t|t-1} = F_t \Sigma_{t-1|t-1} F_t^\top + Q_t$
  Innovation:
  $H_t = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{t|t-1}}$
  $\mathbf{r}_t = \mathbf{z}_t - h(\hat{\mathbf{x}}_{t|t-1})$
  $S_t = H_t \Sigma_{t|t-1} H_t^\top + R_t$
  Kalman Gain:
  $K_t = \Sigma_{t|t-1} H_t^\top S_t^{-1}$
  Update:
  $\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + K_t \mathbf{r}_t$
  $\Sigma_{t|t} = (I - K_t H_t)\Sigma_{t|t-1}$ **or** $\Sigma_{t|t} = (I - K_t H_t)\Sigma_{t|t-1}(I - K_t H_t)^\top + K_t R_t K_t^\top$

Unscented Kalman Filter (UKF): The Unscented Kalman Filter (Wan and Van Der Merwe, 2000) handles non-linearity using the **unscented transform**, which propagates carefully chosen **sigma points** through the non-linear functions without requiring Jacobian computation. **Key Idea:** 1. Generate a minimal set of sigma points that capture the mean and covariance of the state distribution; 2. Associate each sigma point with a weight; 3. Transform the sigma points through the non-linear function; 4. Compute a Gaussian approximation from the weighted transformed points.

Sigma Points Generation: For a state with mean $\bar{\mathbf{x}} \in \mathbb{R}^n$ and covariance $\Sigma_{\mathbf{x}} \in \mathbb{R}^{n \times n}$, generate $2n + 1$ sigma points:

$$\boldsymbol{\chi} = \{\boldsymbol{\chi}_i, W_i\}_{i=0}^{2n}, \tag{39}$$

where

$$\boldsymbol{\chi}_0 = \bar{\mathbf{x}}, \tag{40}$$

$$\boldsymbol{\chi}_i = \bar{\mathbf{x}} + \left(\sqrt{(n+\lambda)\Sigma_{\mathbf{x}}}\right)_i, \quad i = 1, 2, \ldots, n, \tag{41}$$

$$\boldsymbol{\chi}_i = \bar{\mathbf{x}} - \left(\sqrt{(n+\lambda)\Sigma_{\mathbf{x}}}\right)_{i-n}, \quad i = n+1, \ldots, 2n, \tag{42}$$

and $\left(\sqrt{(n+\lambda)\Sigma_{\mathbf{x}}}\right)_i$ denotes the i-th row of the matrix square root.

Scaling Parameters:

$$\lambda = \alpha^2(n + \kappa) - n \tag{43}$$

where

- α controls the spread of sigma points around the mean (typically $10^{-3} \leq \alpha \leq 1$),
- κ is a secondary scaling parameter (typically $\kappa = 0$),
- β incorporates prior knowledge of the distribution ($\beta = 2$ is optimal for Gaussian).

Weights for Mean

$$W_0^{(m)} = \frac{\lambda}{n + \lambda}, \tag{44}$$

$$W_i^{(m)} = \frac{1}{2(n + \lambda)}, \quad i = 1, 2, \ldots, 2n. \tag{45}$$

Weights for Covariance

$$W_0^{(c)} = \frac{\lambda}{n + \lambda} + (1 - \alpha^2 + \beta), \tag{46}$$

$$W_i^{(c)} = \frac{1}{2(n + \lambda)}, \quad i = 1, 2, \ldots, 2n. \tag{47}$$

Unscented Transform: Given sigma points $\{\boldsymbol{\chi}_i\}_{i=0}^{2n}$ and a non-linear function $g(\cdot)$:

1. **Transform sigma points:**

$$\boldsymbol{\gamma}_i = g(\boldsymbol{\chi}_i), \quad i = 0, 1, \ldots, 2n. \tag{48}$$

2. **Compute mean:**

$$\bar{\mathbf{y}} = \sum_{i=0}^{2n} W_i^{(m)} \boldsymbol{\gamma}_i. \tag{49}$$

3. **Compute covariance:**

$$\Sigma_{\mathbf{y}} = \sum_{i=0}^{2n} W_i^{(c)} (\boldsymbol{\gamma}_i - \bar{\mathbf{y}})(\boldsymbol{\gamma}_i - \bar{\mathbf{y}})^\top + \text{noise covariance.} \tag{50}$$

The implementation details can be found here:(`https://github.com/ChunchaoPeter/ml_simulation_research/tree/main/ekf_ukf_pf`).

Algorithm 3 Unscented Kalman Filter (UKF)

Input: non-linear functions f, h, noise covariances $\{Q_t, R_t\}_{t=1}^T$, controls $\{\mathbf{u}_t\}_{t=1}^T$, initial $(\hat{\mathbf{x}}_{0|0}, \Sigma_{0|0})$, UKF parameters α, β, κ
$\lambda = \alpha^2(n + \kappa) - n$
$W_0^{(m)} = \lambda/(n + \lambda)$
$W_0^{(c)} = \lambda/(n + \lambda) + (1 - \alpha^2 + \beta)$
$W_i^{(m)} = W_i^{(c)} = 1/(2(n + \lambda))$ for $i = 1, \ldots, 2n$
for $t = 1$ to T **do**
    Prediction:
    $\boldsymbol{\chi}_{0,t-1} = \hat{\mathbf{x}}_{t-1|t-1}$
    $\boldsymbol{\chi}_{i,t-1} = \hat{\mathbf{x}}_{t-1|t-1} + \left(\sqrt{(n + \lambda)\Sigma_{t-1|t-1}}\right)_i$, $i = 1, \ldots, n$
    $\boldsymbol{\chi}_{i,t-1} = \hat{\mathbf{x}}_{t-1|t-1} - \left(\sqrt{(n + \lambda)\Sigma_{t-1|t-1}}\right)_{i-n}$, $i = n + 1, \ldots, 2n$
    $\boldsymbol{\chi}_{i,t|t-1} = f(\boldsymbol{\chi}_{i,t-1}, \mathbf{u}_t)$, $i = 0, \ldots, 2n$
    $\hat{\mathbf{x}}_{t|t-1} = \sum_{i=0}^{2n} W_i^{(m)} \boldsymbol{\chi}_{i,t|t-1}$
    $\Sigma_{t|t-1} = \sum_{i=0}^{2n} W_i^{(c)} (\boldsymbol{\chi}_{i,t|t-1} - \hat{\mathbf{x}}_{t|t-1})(\boldsymbol{\chi}_{i,t|t-1} - \hat{\mathbf{x}}_{t|t-1})^\top + Q_t$
    Update:
    $\mathbf{z}_{i,t|t-1} = h(\boldsymbol{\chi}_{i,t|t-1})$, $i = 0, \ldots, 2n$
    $\hat{\mathbf{z}}_{t|t-1} = \sum_{i=0}^{2n} W_i^{(m)} \mathbf{z}_{i,t|t-1}$
    $S_t = \sum_{i=0}^{2n} W_i^{(c)} (\mathbf{z}_{i,t|t-1} - \hat{\mathbf{z}}_{t|t-1})(\mathbf{z}_{i,t|t-1} - \hat{\mathbf{z}}_{t|t-1})^\top + R_t$
    $\Sigma_{\mathbf{xz}}^{t|t-1} = \sum_{i=0}^{2n} W_i^{(c)} (\boldsymbol{\chi}_{i,t|t-1} - \hat{\mathbf{x}}_{t|t-1})(\mathbf{z}_{i,t|t-1} - \hat{\mathbf{z}}_{t|t-1})^\top$
    $K_t = \Sigma_{\mathbf{xz}}^{t|t-1} S_t^{-1}$
    $\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + K_t(\mathbf{z}_t - \hat{\mathbf{z}}_{t|t-1})$
    $\Sigma_{t|t} = \Sigma_{t|t-1} - K_t S_t K_t^\top$

Regarding Discuss linearization accuracy limits and sigma-point failures under strong nonlinearity:

The EKF suffers linearization accuracy limits because it linearizes the nonlinear system using Jacobians, giving only first-order accurate propagation of mean/covariance. This introduces large errors when curvature is strong and may cause divergence (Wan and Van Der Merwe, 2000). The UKF avoids Jacobians by using sigma points that capture mean and covariance exactly and propagate them through the true nonlinear dynamics, giving third-order accuracy (Wan and Van Der Merwe, 2000).

Under strong nonlinearity, sigma-point methods can break down because the nonlinear transformation introduces large higher-order terms that the sigma-point approximation cannot capture. In the standard UKF, this issue is aggravated in higher-dimensional systems (dimension greater than three), where the central sigma point receives a large negative weight, making the covariance update numerically unstable. When strong nonlinear functions amplify these unmodelled higher-order components, the sigma-point approximation can yield distorted mean and covariance estimates, leading to significant errors or divergence (Chang et al., 2013).

Reply to Question 1, Part 2(c)

"Implement a standard particle filter for your model (do not use `tfp.experimental.mcmc.particle_filter`). Visualize and discuss issues such as particle degeneracy."

Reply:

Particle Filter: The Particle Filter (Doucet and Johansen, 2009) is a Sequential Monte Carlo (SMC) method that handles non-linearity and non-Gaussianity by representing the posterior distribution using a set of weighted particles. Unlike EKF and UKF which approximate the posterior as a Gaussian, the particle filter can represent arbitrary distributions. **Key Idea:** Represent the filtering distribution $p(\mathbf{x}_t|\mathbf{z}_{1:t})$ using N weighted particles:

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) \approx \sum_{i=1}^{N} w_t^i \delta(\mathbf{x}_t - \mathbf{X}_t^i) \tag{51}$$

where $\{\mathbf{X}_t^i, w_t^i\}_{i=1}^N$ are particles and their associated weights.

Hidden Markov Model formulation: Here I begin indexing at 0 to maintain consistency with the KF, EKF, and UKF. Alternatively, the formulation may start at index 1 Doucet and Johansen (2009). For a state sequence $\mathbf{X}_{0:t} = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_t\}$ and observations $\mathbf{Z}_{1:t} = \{\mathbf{z}_1, \ldots, \mathbf{z}_t\}$:

Prior (Markov property):

$$p(\mathbf{x}_{0:t}) = \mu(\mathbf{x}_0) \prod_{k=1}^{t} f(\mathbf{x}_k|\mathbf{x}_{k-1}) \tag{52}$$

Likelihood (Conditional independence):

$$p(\mathbf{z}_{1:t}|\mathbf{x}_{0:t}) = \prod_{k=1}^{t} g(\mathbf{z}_k|\mathbf{x}_k) \tag{53}$$

Posterior (Bayes' rule):

$$p(\mathbf{x}_{0:t}|\mathbf{z}_{1:t}) = \frac{p(\mathbf{x}_{0:t}, \mathbf{z}_{1:t})}{p(\mathbf{z}_{1:t})} = \frac{p(\mathbf{x}_{0:t})p(\mathbf{z}_{1:t}|\mathbf{x}_{0:t})}{p(\mathbf{z}_{1:t})} \tag{54}$$

where

$$p(\mathbf{x}_{0:t}, \mathbf{z}_{1:t}) = p(\mathbf{x}_{0:t})p(\mathbf{z}_{1:t}|\mathbf{x}_{0:t}), \tag{55}$$

$$p(\mathbf{z}_{1:t}) = \int p(\mathbf{x}_{0:t}, \mathbf{z}_{1:t}) d\mathbf{x}_{0:t}. \tag{56}$$

Filtering Distribution: The goal is to compute the marginal filtering distribution:

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) = \int p(\mathbf{x}_{0:t}|\mathbf{z}_{1:t}) d\mathbf{x}_{0:t-1} \tag{57}$$

The filtering distribution can be computed recursively:
Prediction Step:

$$p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) = \int f(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1}) d\mathbf{x}_{t-1} \tag{58}$$

Update Step:

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) = \frac{g(\mathbf{z}_t|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{z}_{1:t-1})}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1})} \tag{59}$$

For non-linear or non-Gaussian systems, these integrals have no analytic solution. The particle filter approximates these distributions using Monte Carlo sampling.

Based on Sequential Importance Sampling (Doucet and Johansen, 2009), we use a sequential proposal that factorizes as:

$$q(\mathbf{x}_{0:t}|\mathbf{z}_{1:t}) = q(\mathbf{x}_0) \prod_{k=1}^{t} q(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_k) \tag{60}$$

This allows us to sample new particles $\mathbf{X}_t^i \sim q(\mathbf{x}_t|\mathbf{X}_{t-1}^i, \mathbf{z}_t)$ and update weights recursively:

$$w_t^i \propto w_{t-1}^i \frac{g(\mathbf{z}_t|\mathbf{X}_t^i) f(\mathbf{X}_t^i|\mathbf{X}_{t-1}^i)}{q(\mathbf{X}_t^i|\mathbf{X}_{t-1}^i, \mathbf{z}_t)} \tag{61}$$

In Eq. (61), w_t^i denotes the *unnormalised importance weight* of particle i at time t. It is updated recursively to correct for the discrepancy between the proposal distribution and the true filtering model. A larger weight indicates that the particle's predicted state X_t^i is more consistent with the new observation z_t.

Prior Proposal: A common choice is the **prior proposal**:

$$q(\mathbf{x}_0) = \mu(\mathbf{x}_0), \tag{62}$$

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{z}_t) = f(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{63}$$

With this choice, the weight update simplifies to:

$$w_t^i \propto w_{t-1}^i \frac{g(\mathbf{z}_t|\mathbf{X}_t^i) f(\mathbf{X}_t^i|\mathbf{X}_{t-1}^i)}{f(\mathbf{X}_t^i|\mathbf{X}_{t-1}^i)} = w_{t-1}^i \cdot g(\mathbf{z}_t|\mathbf{X}_t^i) \tag{64}$$

Resampling: SIS suffers from exponentially increasing variance as t grows, due to increasingly unbalanced importance weights. Resampling in SMC methods helps mitigate—but not eliminate—this degeneracy. One common resampling method is *multinomial resampling*. We draw an ancestry vector $\mathbf{A}^{1:N}$, where each A^i is sampled independently as

$$A^i \sim \text{Categorical}(w_t^1, w_t^2, \ldots, w_t^N), \qquad i = 1, \ldots, N. \tag{65}$$

Then set:

$$\tilde{\mathbf{X}}_t^i = \mathbf{X}_t^{A^i} \quad \text{(resampled particles)}, \tag{66}$$

$$\tilde{w}_t^i = \frac{1}{N} \quad \text{(uniform weights after resampling)}. \tag{67}$$

Algorithm 4 Particle Filter with Prior Proposal and Resampling

Input: non-linear functions f, h, noise samplers for Q_t, R_t, initial distribution $\mu(\mathbf{x}_0)$, number of particles N
Initialization (t = 0):
for $i = 1$ to N **do**
    Sample $\mathbf{X}_0^i \sim \mu(\mathbf{x}_0)$
    Set $w_0^i = 1/N$
$\hat{\mathbf{x}}_0 = \sum_{i=1}^N w_0^i \mathbf{X}_0^i$

for $t = 1$ to T **do**
    Prediction:
    for $i = 1$ to N **do**
        Sample $\mathbf{X}_t^i \sim f(\mathbf{x}_t | \mathbf{X}_{t-1}^i, \mathbf{u}_t)$ ▷ Propagate particles through dynamics

    Update:
    for $i = 1$ to N **do**
        Compute $\tilde{w}_t^i = w_{t-1}^i \cdot g(\mathbf{z}_t | \mathbf{X}_t^i)$ ▷ Weight by observation likelihood
    Normalize: $w_t^i = \tilde{w}_t^i / \sum_{j=1}^N \tilde{w}_t^j$

    State Estimation:
    $\hat{\mathbf{x}}_t = \sum_{i=1}^N w_t^i \mathbf{X}_t^i$

    Resampling:
    for $i = 1$ to N **do**
        Draw $A^i \sim \text{Categorical}(w_t^1, \ldots, w_t^N)$
        Set $\mathbf{X}_t^i = \mathbf{X}_t^{A^i}$ ▷ Resample particles
    Set $w_t^i = 1/N$ for all i ▷ Reset to uniform weights

The implementation details can be found here: (`https://github.com/ChunchaoPeter/ml_simulation_research/tree/main/ekf_ukf_pf`).

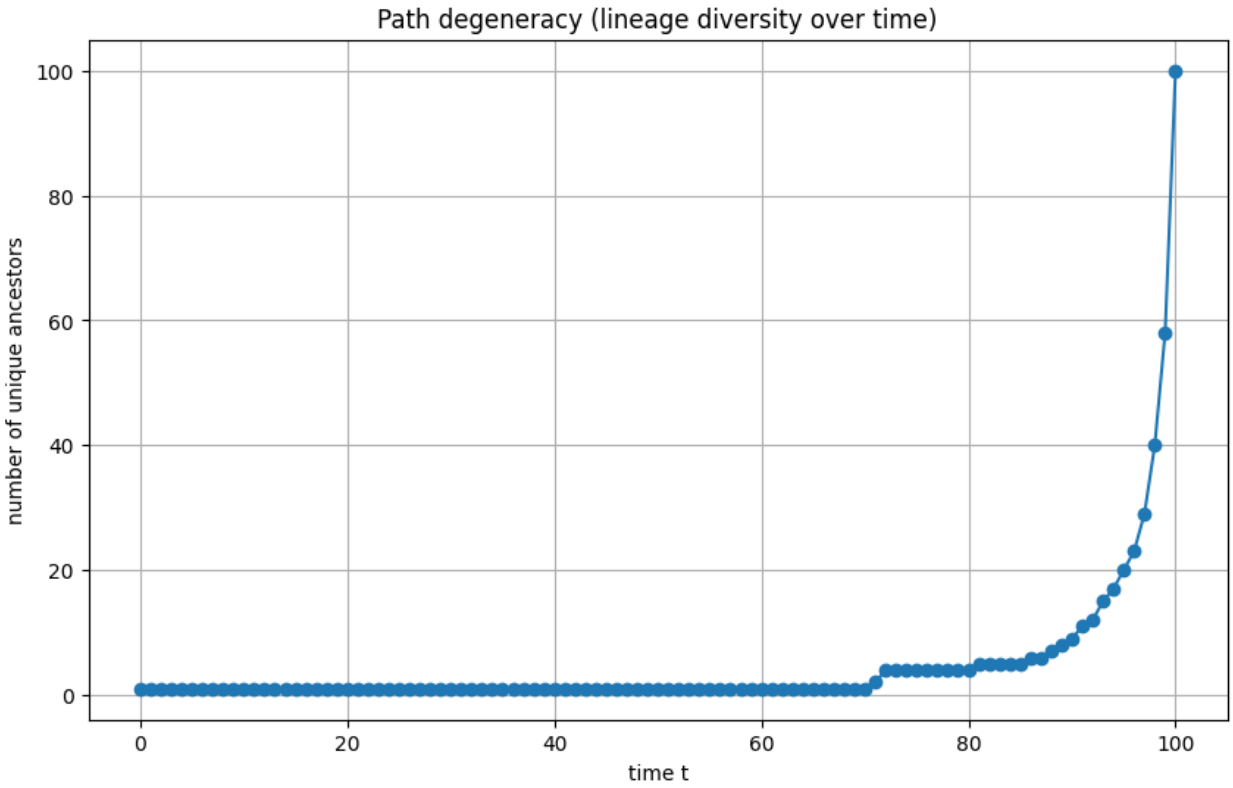

Figure 1: Particle Degeneracy

Regarding visualize and discuss issues such as particle degeneracy:
Figure 1 illustrates the path degeneracy phenomenon in particle filters. Although the particle population may appear diverse at the final time step, their ancestral lineages collapse very quickly after just a few rounds of resampling. This means that, going backward in time, nearly all particles descend from a single early ancestor. One of the reason is that Resampling repeatedly duplicates particles with high weights while discarding those with low weights. Over many iterations, this causes most particles to descend from only a few early ancestors, leading to a rapid loss of genealogical diversity.

Reply to Question 1, Part 2(d)

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

Reply:

State Space Model Comparison and Evaluation: We evaluated three state estimation algorithms—Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), and Particle Filter (PF)—on a range-bearing tracking problem with mildly non-linear dynamics. State space models are evaluated using three categories of metrics: accuracy metrics (position

RMSE, velocity RMSE, and average position error over time), computational performance metrics (runtime and peak memory usage).

Accuracy Performance: In terms of accuracy, EKF and UKF achieved nearly identical performance with position RMSE of 2.33 and 2.34 respectively, and velocity RMSE of approximately 0.42 for both filters. The Particle Filter with 100 particles demonstrated inferior accuracy with position RMSE of 4.54 and velocity RMSE of 0.53. For this mildly non-linear system, the linearization-based EKF and the sigma-point-based UKF both provide accurate state estimates, while PF accuracy is limited by the relatively small particle count used in the comparison.

Runtime Performance: The computational runtime differs dramatically across the three methods. UKF was the fastest with a runtime of 3.18 seconds, followed closely by EKF at 3.46 seconds. In stark contrast, the Particle Filter required 102.69 seconds to process the same trajectory. This represents a speedup factor of approximately 32× for UKF and 30× for EKF compared to PF. The significant computational advantage of EKF and UKF stems from their parametric representation of the posterior distribution as a Gaussian, requiring only mean and covariance updates, whereas PF must propagate and resample hundreds of particles at each time step.

Memory Efficiency: Peak memory usage shows similar disparities. UKF was the most memory-efficient algorithm, using only 1.97 MB of RAM, which represents just 23.9% of the Particle Filter's memory footprint. EKF required 3.45 MB (41.8% of PF memory), while PF consumed 8.25 MB. The memory requirements of PF scale linearly with the number of particles, making it particularly demanding for real-time applications or resource-constrained environments. UKF's superior memory efficiency compared to EKF is noteworthy, as it achieves this while maintaining derivative-free operation through sigma point sampling.

Note: All experiments for this question were performed on a 2017 15-inch MacBook Pro equipped with a 3.1 GHz quad-core Intel Core i7 processor, Radeon Pro 560 graphics, 16 GB of LPDDR3 memory, and running macOS Ventura 13.7.8. For more detail, see the notebook at `https://github.com/ChunchaoPeter/ml_simulation_research/blob/main/ekf_ukf_pf/compare_pf_ekf_ukf.ipynb`

Reply to Question 2, Part (a)

"Study the Exact Daum–Huang (EDH) flow and Local Exact Daum–Huang (LEDH) flow (see Daum et al. (2010); Daum and Huang (2011)), and the invertible particle flow particle filter (PF-PF) framework (see Li and Coates (2017)). Replicate the main results in Li and Coates (2017)."

Reply:

Daum et al. (2010); Daum and Huang (2011) introduce particle flow as an alternative to discrete Bayesian weight updates, formulating the measurement update as a continuous transformation of the probability density governed by a partial differential equation. In theory, particle flow methods avoid particle degeneracy, remove the need for resampling, and scale more effectively to high-dimensional state spaces. In this section, we describe a special case of particle flow underlying the Exact Daum–Huang (EDH), Local Exact Daum–Huang

(LEDH), and the invertible particle-flow particle filter (PF-PF) framework.

The system is defined as

$$\mathbf{x}_k = f_k(\mathbf{x}_{k-1}) + \mathbf{w}_k, \tag{68}$$
$$\mathbf{y}_k = \gamma_k(\mathbf{x}_k) + \mathbf{v}_k, \tag{69}$$

where

- $\mathbf{x}_k \in \mathbb{R}^{d_x \times 1}$ is the hidden state,
- $\mathbf{y}_k \in \mathbb{R}^{d_y \times 1}$ is the observation,
- $\mathbf{w}_k$ and $\mathbf{v}_k$ denote process and observation noise.

Here, f_k is assumed to be linear such that

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{w}_k. \tag{70}$$

The goal is to recursively estimate the system state $\mathbf{x}_k$ at each time step k. Based on Bayesian theory, the unnormalized posterior is given by

$$\tilde{p}(\mathbf{x}_k \mid \mathbf{y}_{1:k}) = p(\mathbf{y}_k \mid \mathbf{x}_k)\, p(\mathbf{x}_k \mid \mathbf{y}_{1:k-1}). \tag{71}$$

We define

$$g(\mathbf{x}_k) = p(\mathbf{x}_k \mid \mathbf{y}_{1:k-1}), \tag{72}$$
$$h(\mathbf{x}_k) = p(\mathbf{y}_k \mid \mathbf{x}_k), \tag{73}$$

as the predictive prior and the likelihood, respectively.

Following Daum et al. (2010), we introduce a homotopy function $\phi(\mathbf{x}_k, \lambda)$:

$$\phi(\mathbf{x}_k, \lambda) = \log g(\mathbf{x}_k) + \lambda \log h(\mathbf{x}_k), \qquad \lambda \in [0, 1]. \tag{74}$$

Two special cases are of particular interest:

$$\lambda = 0 \Rightarrow \phi(\mathbf{x}_k, 0) = \log g(\mathbf{x}_k), \tag{75}$$
$$\lambda = 1 \Rightarrow \phi(\mathbf{x}_k, 1) = \log \tilde{p}(\mathbf{x}_k \mid \mathbf{y}_{1:k}). \tag{76}$$

Intuitively, λ acts as a continuation parameter that smoothly deforms the prior density into the posterior density. At $\lambda = 0$, the density corresponds to the prior, while at $\lambda = 1$, it coincides with the posterior. This continuous deformation naturally motivates the introduction of a *particle flow*, which transports particles from the prior to the posterior via a continuous transformation.

For the Exact Daum–Huang (EDH) flow, suppose that the particle dynamics are governed by the ordinary differential equation

$$\frac{d\mathbf{x}}{d\lambda} = \psi(\mathbf{x}, \lambda). \tag{77}$$

By invoking the Fokker–Planck equation, the flow field $\psi(\mathbf{x}, \lambda)$ is required to satisfy the following relationship (Ding and Coates, 2012):

$$\frac{\partial \phi}{\partial \mathbf{x}} \psi(\mathbf{x}, \lambda) + \log(h) = -\operatorname{Tr}\left(\frac{\partial \psi}{\partial \mathbf{x}}\right). \tag{78}$$

A solution to this equation defines the *exact* particle flow that transports the probability density continuously from the prior to the posterior.

When both $\log g$ and $\log h$ can be expressed as polynomials in the state variables—such as in Gaussian and other exponential-family distributions—the differential equation $\frac{d\mathbf{x}}{d\lambda}$ admits a closed-form solution derived from 78. In the linear Gaussian setting, let $\bar{\mathbf{x}}$ represent the predicted value of $\mathbf{x}$ and let P denote the associated prediction error covariance. The measurement process is assumed to follow

$$y_k = H\mathbf{x_k} + \mathbf{v}_k,$$

where H is the measurement matrix and R is the covariance of the measurement noise. Under these assumptions, Daum and Huang obtain the following affine flow expression Daum et al. (2010); Ding and Coates (2012):

$$\frac{d\mathbf{x}}{d\lambda} = A(\lambda)\mathbf{x} + b(\lambda). \tag{79}$$

The matrix-valued coefficient $A(\lambda)$ is given by

$$A(\lambda) = -\frac{1}{2} P H^\top \left(\lambda H P H^\top + R\right)^{-1} H, \tag{80}$$

and the corresponding offset term $b(\lambda)$ is defined as

$$b(\lambda) = \left(I + 2\lambda A(\lambda)\right)\left[\left(I + \lambda A(\lambda)\right) P H^\top R^{-1} z + A(\lambda)\bar{\mathbf{x}}\right]. \tag{81}$$

In the case of nonlinear measurement models, local approximations can be introduced via Taylor expansion. The predicted mean $\bar{\mathbf{x}}$ may be estimated directly from the particle ensemble, while linearizing the measurement function yields an approximate measurement matrix $H_{\bar{\mathbf{x}}}$.

Algorithm 5 Original Exact Flow Daum–Huang Filter (EDH)

Initialization: Draw $\{\mathbf{x}_0^i\}_{i=1}^N$ from the prior $p(\mathbf{x}_0)$
Set $\hat{\mathbf{x}}_0$ and $\mathbf{m}_0$ as the mean; P_0 as the covariance matrix
for $k = 1$ to T **do**
    Propagate particles:

$$\mathbf{x}_{k|k-1}^i = f_k(\mathbf{x}_{k-1}^i) + \mathbf{v}_k$$

    Calculate the mean value $\bar{\mathbf{x}}_k$
    Apply UKF/EKF prediction:

$$(\mathbf{m}_{k-1|k-1}, P_{k-1|k-1}) \to (\mathbf{m}_{k|k-1}, P_{k|k-1})$$

    for $j = 1, \ldots, N_\lambda$ **do**
        Set $\lambda = j\Delta\lambda$
        Calculate $H_{\mathbf{x}}$ by linearizing $\gamma_k(\cdot)$ at $\bar{\mathbf{x}}_k$
        Calculate A and b from 80 and 81 using $P_{k|k-1}$, $\bar{\mathbf{x}}_k$, and $H_{\mathbf{x}}$
        for $i = 1, \ldots, N$ **do**
            Evaluate $\dfrac{d\mathbf{x}_k^i}{d\lambda}$ for each particle from 79
            Migrate particles:

$$\mathbf{x}_k^i = \mathbf{x}_k^i + \Delta\lambda \cdot \frac{d\mathbf{x}_k^i}{d\lambda}$$

        Re-evaluate $\bar{\mathbf{x}}_k$ using the updated particles $\{\mathbf{x}_k^i\}$
    Apply UKF/EKF update:

$$(\mathbf{m}_{k|k-1}, P_{k|k-1}) \to (\mathbf{m}_{k|k}, P_{k|k})$$

    Estimate $\hat{\mathbf{x}}_k$ from the particles $\{\mathbf{x}_k^i\}$ using $P_{k|k}$
    Optional: redraw particles $\mathbf{x}_k^i \sim \mathcal{N}(\hat{\mathbf{x}}_k, P_{k|k})$

The local EDH (Ding and Coates, 2012) closely resembles the standard EDH formulation. As implied by the term local, it introduces a key modification whereby the measurement function is linearized for each particle, yielding $H_{\mathbf{x}_i}$, A^i, and b^i. These modifications are summarized in Algorithm 6.

Algorithm 6 Modified Local Exact Flow Daum–Huang Filter (LEDH) (replaces lines 7–17 of Algorithm 5)

1: **for** $j = 1, \ldots, N_\lambda$ **do**
2: Set $\lambda = j\Delta\lambda$;
3: **for** $i = 1, \ldots, N$ **do**
4: Calculate $H_{\mathbf{x}_i}$ by linearizing $\gamma_k(\cdot)$ at $\mathbf{x}^i$;
5: Calculate A^i and b^i from 80 and 81 using $P_{k|k-1}$, $\bar{\mathbf{x}}$ and $H_{\mathbf{x}_i}$;
6: Evaluate $\dfrac{d\mathbf{x}_k^i}{d\lambda}$ for each particle from 79;
7: Migrate particles:

$$\mathbf{x}_k^i = \mathbf{x}_k^i + \Delta\lambda \cdot \frac{d\mathbf{x}_k^i}{d\lambda};$$

8: Re-evaluate $\bar{\mathbf{x}}_k$ using updated particles $\mathbf{x}_k^i$.
9: Apply UKF/EKF update:

$$(\mathbf{m}_{k|k-1}, P_{k|k-1}) \to (\mathbf{m}_{k|k}, P_{k|k});$$

10: Estimate $\hat{\mathbf{x}}_k$ from the particles $\mathbf{x}_k^i$ using $P_{k|k}$;
11: Set $\mathbf{m}_{k|k} = \hat{\mathbf{x}}_k$;
12: **Optional:** redraw particles $\mathbf{x}_k^i \sim \mathcal{N}(\hat{\mathbf{x}}_k, P_{k|k})$;

Regarding the invertible particle flow particle filter (PF-PF) framework (Li and Coates, 2017), deterministic particle flows are embedded within a standard particle filtering scheme by treating the particle flow as an invertible deterministic proposal. Specifically, an invertible mapping

$$\mathbf{x}_k^i = T(\mathbf{x}_{k-1}^i, \mathbf{z}_k)$$

is constructed to transport particles from the predictive prior to the posterior distribution. This mapping induces a proposal density of the form

$$q(\mathbf{x}_k^i \mid \mathbf{x}_{k-1}^i, \mathbf{z}_k) = \frac{p(\mathbf{x}_{k-1}^i \mid \mathbf{z}_{1:k-1})}{\left|\det \nabla_{\mathbf{x}} T(\mathbf{x}_{k-1}^i, \mathbf{z}_k)\right|},$$

which can be evaluated exactly through the associated Jacobian determinant. Within this framework, particle filtering and particle flow are naturally combined, allowing the particle flow to serve as an efficient proposal mechanism. This idea is further applied to the EDH and LEDH particle flow formulations in order to assess their performance within the PF-PF framework.

Algorithm 7 Particle Flow Particle Filtering (EDH)

Initialization: Draw $\{\mathbf{x}_0^i\}_{i=1}^N$ from the prior $p_0(\mathbf{x})$. Set $\hat{\mathbf{x}}_0$ and $\hat{P}_0$ to be the mean and covariance of $p_0(\mathbf{x})$.
Set $\{w_0^i\}_{i=1}^N = \frac{1}{N}$.
for $k = 1$ to T **do**
    Apply EKF/UKF prediction to estimate P

$$(\hat{\mathbf{x}}_{k-1}, P_{k-1}) \to (\mathbf{m}_{k|k-1}, P)$$

    for $i = 1, \ldots, N$ **do**
        Propagate particles: $\boldsymbol{\eta}_0^i = f_k(\mathbf{x}_{k-1}) + \mathbf{w}_k$
        Set $\boldsymbol{\eta}_1^i = \boldsymbol{\eta}_0^i$
    Compute flow linearization point: $\bar{\boldsymbol{\eta}}_0 = f_k(\hat{\mathbf{x}}_{k-1}), \quad \bar{\boldsymbol{\eta}} = \bar{\boldsymbol{\eta}}_0$
    Set $\lambda = 0$
    for $j = 1, \ldots, N_\lambda$ **do**
        $\lambda \leftarrow \lambda + \epsilon_j$
        Compute shared flow parameters $A^j(\lambda)$ and $b^j(\lambda)$ from 80 and 81, linearized at $\bar{\boldsymbol{\eta}}$
        Migrate mean: $\bar{\boldsymbol{\eta}} \leftarrow \bar{\boldsymbol{\eta}} + \epsilon_j\big(A^j(\lambda)\bar{\boldsymbol{\eta}} + b^j(\lambda)\big)$
        for $i = 1, \ldots, N$ **do**
            Migrate particles: $\boldsymbol{\eta}_1^i \leftarrow \boldsymbol{\eta}_1^i + \epsilon_j\big(A^j(\lambda)\boldsymbol{\eta}_1^i + b^j(\lambda)\big)$
    for $i = 1, \ldots, N$ **do**
        Set $\mathbf{x}_k^i = \boldsymbol{\eta}_1^i$
        Update weights: $w_k^i = \frac{p(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i)\, p(\mathbf{z}_k|\mathbf{x}_k^i)}{p(\boldsymbol{\eta}_0^i|\mathbf{x}_{k-1}^i)} w_{k-1}^i$
    Normalize $\{w_k^i\}_{i=1}^N$
    Measurement update: $(\mathbf{m}_{k|k-1}, P_{k|k-1}) \to (\mathbf{m}_{k|k}, P_k)$
    Estimate: $\hat{\mathbf{x}}_k = \sum_{i=1}^N w_k^i \mathbf{x}_k^i$
    Optional: Resample $\{\mathbf{x}_k^i, w_k^i\}_{i=1}^N$

For simplicity, in Algorithms 7 and 8, we focus on illustrating the core idea of the Particle Flow Particle Filter (PF-PF) with EDH and LEDH. Accordingly, the matrices A and b are computed using Equations 80–81. A more detailed formulation of these functions is provided in the original paper (Li and Coates, 2017). In addition, the original work presents a rigorous theoretical justification for the use of invertible particle flows within the PF-PF framework.

The code for each algorithm is implemented in the `deterministic_kernel_flows` folder of our GitHub repository. The file `edh.py` implements both the EDH and LEDH algorithms, with usage demonstrated in the notebook `edh_ledh_demo.ipynb`. The files `pfpf_edh.py` and `pfpf_ledh.py` implement the Particle Flow Particle Filter (PF-PF) with EDH and LEDH, respectively. Demonstrations of these algorithms are provided in `pfpf_edh_demo.ipynb` and `pfpf_ledh_demo_with_comparison.ipynb`. To replicate the main results in Li and Coates (2017), we additionally provide `acoustic_function.py`, which contains the acoustic tracking

Algorithm 8 Particle Flow Particle Filtering (LEDH)

Initialization: Draw $\{\mathbf{x}_0^i\}_{i=1}^N$ from the prior $p_0(\mathbf{x})$. Set $\hat{\mathbf{x}}_0$ and $\hat{P}_0$ to be the mean and covariance of $p_0(\mathbf{x})$.
Set $\{w_0^i\}_{i=1}^N = \frac{1}{N}$.
for $k = 1$ to T **do**
    for $i = 1, \ldots, N$ **do**
        Apply EKF/UKF prediction: $(\mathbf{x}_{k-1}^i, P_{k-1}^i) \to \mathbf{m}_{k|k-1}^i, P^i)$
        Initialize flow: $\bar{\boldsymbol{\eta}}^i = f_k(\mathbf{x}_{k-1}^i), \quad \boldsymbol{\eta}_0^i = f_k(\mathbf{x}_{k-1}^i) + \mathbf{w}_k, \quad \boldsymbol{\eta}_1^i = \boldsymbol{\eta}_0^i, \quad \theta^i = 1$
        Set $\bar{\boldsymbol{\eta}}_0^i = f_k(\mathbf{x}_{k-1}^i)$
    Set $\lambda = 0$
    for $j = 1, \ldots, N_\lambda$ **do**
        $\lambda \leftarrow \lambda + \epsilon_j$
        for $i = 1, \ldots, N$ **do**
            Compute particle-wise flow parameters $A_j^i(\lambda)$ and $b_j^i(\lambda)$ from 80 - 81 linearized at $\bar{\boldsymbol{\eta}}^i$
            Migrate mean: $\bar{\boldsymbol{\eta}}^i \leftarrow \bar{\boldsymbol{\eta}}^i + \epsilon_j \big(A_j^i(\lambda)\bar{\boldsymbol{\eta}}^i + b_j^i(\lambda)\big)$
            Migrate particles: $\boldsymbol{\eta}_1^i \leftarrow \boldsymbol{\eta}_1^i + \epsilon_j \big(A_j^i(\lambda)\boldsymbol{\eta}_1^i + b_j^i(\lambda)\big)$
            Update Jacobian determinant: $\theta^i \leftarrow \theta^i \left|\det\big(I + \epsilon_j A_j^i(\lambda)\big)\right|$
    for $i = 1, \ldots, N$ **do**
        Set $\mathbf{x}_k^i = \boldsymbol{\eta}_1^i$
        Update weights: $w_k^i = \frac{p(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i)\, p(y_k|\mathbf{x}_k^i)\, \theta^i}{p(\boldsymbol{\eta}_0^i|\mathbf{x}_{k-1}^i)} w_{k-1}^i$
    Normalize $\{w_k^i\}_{i=1}^N$
    for $i = 1, \ldots, N$ **do**
        Apply EKF/UKF update: $(\mathbf{m}_{k|k-1}^i, P^i) \to (\mathbf{m}_{k|k}^i, P_k^i)$
    Estimate: $\hat{\mathbf{x}}_k = \sum_{i=1}^N w_k^i \mathbf{x}_k^i$
    Optional: Resample $\{\mathbf{x}_k^i, P_k^i, w_k^i\}_{i=1}^N$

model, along with a corresponding demonstration notebook `acoustic_function_demo.ipynb`. Further implementation details can be found in the relevant code within the GitHub repository.

Reply to Question 2, Part (b)

"Implement the kernel-embedded particle flow filter in an RKHS following Hu and Van Leeuwen (2021). Then compare the scalar kernel and diagonal matrix-valued kernel. Use experiments to demonstrate that matrix-valued kernels can prevent collapse of observed-variable marginals in high dimensions; plot similar figures as in Figures 2–3 of Hu and Van Leeuwen (2021)."

Reply:

Particle Flow Filtering (PFF) (Hu and Van Leeuwen, 2021) combines ideas from particle filtering and optimal transport. The particle evolution is defined through a continuous flow governed by the ordinary differential equation

$$\frac{d\mathbf{x}_{k,s}}{ds} = f_{k,s}(\mathbf{x}_{k,s}), \qquad s \in [0,\infty), \quad \mathbf{x}_{k,s} \in \mathbb{R}^{n_x}, \tag{82}$$

where $\mathbf{x}_{k,s}$ denotes the state of a particle at discrete time index k and pseudo-time s. The pseudo-time variable s plays a role analogous to the homotopy (or λ) parameter commonly used in particle flow formulations, such as those appearing in Algorithm 8.

Let

$$q_{k,0}(\mathbf{x}) = p(\mathbf{x}) \quad \text{(prior)}, \qquad q_{k,\infty}(\mathbf{x}) = p(\mathbf{x} \mid \mathbf{y}) \quad \text{(posterior)}. \tag{83}$$

The goal is to choose a suitable flow field $f_{k,s}$ such that the discrepancy between the intermediate density $q_{k,s}(\mathbf{x})$ and the target posterior $q_{k,\infty}(\mathbf{x})$ decreases as the pseudo-time s increases. The discrepancy is measured using the Kullback–Leibler (KL) divergence,

$$\mathrm{KL}(q_{k,s} \,\|\, q_{k,\infty}) = \int q_{k,s}(\mathbf{x}) \log \frac{q_{k,s}(\mathbf{x})}{q_{k,\infty}(\mathbf{x})}\, d\mathbf{x}. \tag{84}$$

The flow field $f_{k,s}$ is assumed to lie in a reproducing kernel Hilbert space (RKHS) associated with a matrix-valued kernel

$$K : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_x \times n_x}. \tag{85}$$

Under the RKHS representation, the flow can be written as

$$f_{k,s}(\mathbf{x}) = \langle K(\mathbf{x}, \cdot), f_{k,s}(\cdot) \rangle_{\mathcal{H}} \in \mathbb{R}^{n_x}. \tag{86}$$

To ensure that the KL divergence decreases over pseudo-time, we require

$$\frac{d}{ds}\mathrm{KL}(q_{k,s} \,\|\, q_{k,\infty}) \leq 0. \tag{87}$$

More precisely, we aim to maximize the rate of decrease of the KL divergence, i.e., move in the steepest descent direction in function space. After calculation, the optimal flow is given by

$$f_s(\cdot) = D \int q_s(\mathbf{x}) \left[\nabla_{\mathbf{x}} \cdot K(\mathbf{x}, \cdot) + K(\mathbf{x}, \cdot) \nabla_{\mathbf{x}} \log p(\mathbf{x} \mid \mathbf{y})\right] d\mathbf{x} = D I_f, \tag{88}$$

Algorithm 9 Particle Flow Filter Update at Time k

Input: Prior particles $\{\mathbf{x}_{k-1}^j\}_{j=1}^N$, observation $\mathbf{y}_k$, observation operator $H(\cdot)$, likelihood $p(\mathbf{y}_k \mid \mathbf{x})$, prior density $p(\mathbf{x})$
Initialization:
for $j = 1, \ldots, N$ **do**
    $\mathbf{x}_{k,0}^j \leftarrow H(\mathbf{x}_{k-1}^j, \mathbf{y}_k)$
$i \leftarrow 0$
repeat
    for $j = 1, \ldots, N_p$ **do**
        $\mathbf{x}_{k,s+1}^j \leftarrow \mathbf{x}_{k,s}^j + \varepsilon\, f_{k,s}(\mathbf{x}_{k,s}^j)$
    $s \leftarrow s + 1$
until stopping criterion is met
Output: Posterior particles $\{\mathbf{x}_{k,s}^j\}_{j=1}^N$

where D is a positive-definite matrix; for notational simplicity, the explicit dependence of f_s and q_s on the time index k is suppressed.

The derivative of the KL divergence along the flow satisfies

$$\frac{d}{ds}\mathrm{KL}(q_{k,s} \,\|\, q_{k,\infty}) = -\langle I_f, DI_f \rangle \leq 0, \tag{89}$$

which guarantees decrease since $D \succ 0$. The matrix D is chosen to ensure correct physical dimensions of the flow and is commonly set to a localized prior covariance matrix. Algorithm 9 illustrates the implementation of the Particle Flow Filter.

The implementation of each algorithm is provided in the `deterministic_kernel_flows` directory of our GitHub repository. The file `pff.py` contains the implementation of the Particle Flow Filter (PFF) algorithms. The notebook `pff_demo_100_dimension.ipynb` shows the demonstration and includes a comparison between the scalar kernel and the diagonal matrix-valued kernel. In `pff_showcase_matrix_valued_kernel.ipynb`, we conduct experiments to reproduce figures analogous to Figures 2–3 in Hu and Van Leeuwen (2021). Additional implementation details are available in the corresponding source code within the GitHub repository.

Reply to Question 2, Part (c)

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

Reply:

In the notebook `compare_edh_ledh_pff_with_range_model.ipynb`, We implement the EDH and LEDH algorithms using the state-space model designed in the previous particle-filter question.

The current PFF implementation in `pff.py` does not yet generalize to arbitrary datasets; extending the implementation to more general settings is planned as future work.

Finally, we present a brief comparison of EDH, LEDH, and PFF based on the preceding experiments and the theoretical formulations of each algorithm.

D. Testing plans and testing results that show your implementation is correct and your results are valid

All implementations are validated through comprehensive tests (Unit Tests and Integration Tests) covering KF, EKF, UKF, PF, EHD, LEDH, PFPF-EHD, PFPF-LEDH, PFF and the range–bearing model. More details can be found in the **tests** folder under the **kf_lgssm**, **ekf_ukf_pf**, and **deterministic_kernel_flows** directories, respectively, in this repository: `https://github.com/ChunchaoPeter/ml_simulation_research/tree/main`.

Bibliography

(2025). Kalman filter. `https://en.wikipedia.org/wiki/Kalman_filter`. Accessed: 2025-02-14.

Chang, L., Hu, B., Li, A., and Qin, F. (2013). Unscented type kalman filter: limitation and combination. *IET Signal Processing*, 7(3):167–176.

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

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

Ding, T. and Coates, M. J. (2012). Implementation of the daum-huang exact-flow particle filter. In *2012 IEEE Statistical Signal Processing Workshop (SSP)*, pages 257–260. IEEE.

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

Evangelidis, A. and Parker, D. (2019). Quantitative verification of numerical stability for kalman filters. In *International Symposium on Formal Methods*, pages 425–441. Springer.

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

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

Pei, Y., Biswas, S., Fussell, D. S., and Pingali, K. (2019). An elementary introduction to kalman filtering. *Communications of the ACM*, 62(11):122–133.

Pulido, M. and van Leeuwen, P. J. (2019). Sequential monte carlo with kernel embedded mappings: The mapping particle filter. *Journal of Computational Physics*, 396:400–415.

Wan, E. A. and Van Der Merwe, R. (2000). The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 adaptive systems for signal processing, communications, and control symposium (Cat. No. 00EX373)*, pages 153–158. Ieee.