Report for the problems

Haowu Duan

November 24, 2025

Abstract

The current state of the report is for part one and not polished. I organize my understanding of the problem and derive the key formulas in the overview section. Limitations of each methods mostly have been addressed in the overview in details. When it comes to more recent methods, the literature review is not complete. I have what I need to procede to work on part 2 and the bonus. I will refine part one later. As for testing, due to lack of experience, it is still work in progress.

Contents

1 Overview for part one:

In this section, I want to lay out clear theoretical foundation for the problem at hand. Let us start with the general case for a State space model with additive noise. The evolution is governed by

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}) + \mathbf{w}_t, \quad \mathbf{w}_t \sim p_w(\cdot) \tag{1}$$

and the observation model is,

$$\mathbf{y}_t = h(\mathbf{x}_t) + \mathbf{v}_t, \quad \mathbf{v}_t \sim p_v(\cdot) \tag{2}$$

if there is no measurement uncertainty, then we are just dealing with the evolution of the state $\mathbf{x}_t$. Obviously, we want to be able to determine $p_t(\mathbf{x}_t)$. Add the path integral discussion here.

1.1 1-d Linear Gaussian model and Kalman filter

For most of methods we discuss here, except the flow methods. We can run the evolution and filtering independently. Here we first discuss how Kalman filter works in one dimension then we summarize what do we want to get out of a filter. In one dimension,

$$\begin{aligned} x_t =& ax_{t-1} + w_t, \quad w_t \sim \mathcal{N}(0, Q) \\ y_t =& hx_t + v_t, \quad v_t \sim \mathcal{N}(0, R) \end{aligned} \tag{3}$$

we have the following initial condition,

$$p(x_0) = \mathcal{N}(x_0 | m_0, P_0) \tag{4}$$

sensitivity for initial condition should be discussed as well.

Notation:

- $\mathbf{m}_t^-$= *a priori* state estimate (mean of the predicted distribution)
- $\mathbf{m}_t$= *a posteriori* state estimate (mean of the updated distribution)
- $\mathbf{P}_t^-$ = *a priori* error covariance
- $\mathbf{P}_t$ = *a posteriori* error covariance
- $p(x_k|y_{1:k-1})$= prior distribution
- $p(x_k|y_{1:k})$ = posterior Distribution
- $p(y_k|x_k)$ = likelihood

For Kalman filter, the noise and operators are both Gaussian, so the distributions stay at Gaussian at all times. All we need to do is to keep track of the mean and variance of the prior distribution $p(x_k|y_{1:k})$.

In the predict step, the goal is to compute $p(x_t|y_{1:t-1})$ from $p(x_{t-1}|y_{1:t-1})$. Given $p(x_{t-1}|y_{1:t-1}) = \mathcal{N}(x_{t-1}|m_{t-1}, P_{t-1})$, we have:

$$p(x_t|y_{1:t-1}) = \int dx_{t-1}\, p(x_t|x_{t-1})\, p(x_{t-1}|y_{1:t-1}) \tag{5}$$

From the state evolution model:

$$p(x_t|x_{t-1}) = \mathcal{N}(x_t|ax_{t-1}, Q) \tag{6}$$

predicted/evolved mean is,

$$\begin{aligned} m_t^- &= \mathbb{E}[x_t|y_{1:t-1}] && (7)\\ &= \mathbb{E}[\mathbb{E}[x_t|x_{t-1}, y_{1:t-1}]|y_{1:t-1}] && (8)\\ &= \mathbb{E}[ax_{t-1}|y_{1:t-1}] && (9)\\ &= am_{t-1} && (10) \end{aligned}$$

predicted/evolved variance

$$\begin{aligned} P_t^- &= \mathrm{Var}[x_t|y_{1:t-1}] && (11)\\ &= \mathbb{E}[\mathrm{Var}[x_t|x_{t-1}, y_{1:t-1}]|y_{1:t-1}] + \mathrm{Var}[\mathbb{E}[x_t|x_{t-1}, y_{1:t-1}]|y_{1:t-1}] && (12)\\ &= \mathbb{E}[Q|y_{1:t-1}] + \mathrm{Var}[ax_{t-1}|y_{1:t-1}] && (13)\\ &= Q + a^2P_{t-1} && (14) \end{aligned}$$

from the predict step, we get,

$$p(x_t|y_{1:t-1}) = \mathcal{N}(x_t|m_t^-, P_t^-) \tag{15}$$

where

$$\boxed{m_t^- = am_{t-1}, \quad P_t^- = a^2P_{t-1} + Q} \tag{16}$$

In the update step, we need to incorporate observation y_t using Bayes' rule.

$$p(x_t|y_{1:t}) = \frac{p(y_t|x_t)\, p(x_t|y_{1:t-1})}{p(y_t|y_{1:t-1})} \tag{17}$$

From the observation model:

$$p(y_t|x_t) = \mathcal{N}(y_t|hx_t, R) \tag{18}$$

We need to compute the product of two Gaussians:

$$\begin{aligned} p(x_t|y_{1:t}) &\propto \mathcal{N}(y_t|hx_t, R) \cdot \mathcal{N}(x_t|m_t^-, P_t^-) && (19)\\ &\propto \exp\left(-\frac{1}{2}\frac{(y_t - hx_t)^2}{R}\right) \exp\left(-\frac{1}{2}\frac{(x_t - m_t^-)^2}{P_t^-}\right) && (20) \end{aligned}$$

Expanding the exponents:

$$-\frac{1}{2}\left[\frac{(y_t - hx_t)^2}{R} + \frac{(x_t - m_t^-)^2}{P_t^-}\right] = -\frac{1}{2}\left[\frac{y_t^2 - 2y_thx_t + h^2x_t^2}{R} + \frac{x_t^2 - 2x_tm_t^- + (m_t^-)^2}{P_t^-}\right] \tag{21}$$

Collecting terms in x_t^2 and x_t:

$$= -\frac{1}{2}\left[\left(\frac{h^2}{R} + \frac{1}{P_t^-}\right)x_t^2 - 2\left(\frac{hy_t}{R} + \frac{m_t^-}{P_t^-}\right)x_t + \text{const}\right] \tag{22}$$

This is a Gaussian in x_t. Completing the square, we have $\frac{1}{P_t} = \frac{h^2}{R} + \frac{1}{P_t^-}$. P is the uncertainty so the inverse of P is like precision, by adding information about the measurement $\frac{h^2}{R}$, we get additional precision which is exactly what we are aiming for. When $\frac{h^2}{R} = 0$, namely large

noise from observation, we say we can't trust the measurement and we should stick with the dynamics. We see that both mean and variance will receive no update. On the other hand, if $\frac{h^2}{R} \ll \frac{1}{P_t^-}$, then observation will carry more weight and the result from dynamics should be disregarded. The updated variance is

$$P_t = \frac{RP_t^-}{h^2 P_t^- + R} \tag{23}$$

For simplicity, we define the **Kalman gain:**

$$K_t = \frac{P_t^- h}{h^2 P_t^- + R} \tag{24}$$

Then:

$$P_t = (1 - K_t h) P_t^- \tag{25}$$

After algebra:

$$m_t = m_t^- + K_t(y_t - h m_t^-) \tag{26}$$

In summary, we get,

$$\boxed{\begin{aligned} K_t &= \frac{P_t^- h}{h^2 P_t^- + R} \\ m_t &= m_t^- + K_t(y_t - h m_t^-) \\ P_t &= (1 - K_t h) P_t^- \end{aligned}} \tag{27}$$

at this point, we can already see that what filter does is to correct the distribution based on the observed data point. Essentially, we want the mean and covariance from the updated distribution at each time step.

For higher dimension, all we have to do is to promote the scalar recursive relation into matrices.
Prediction Step:

$$\hat{\mathbf{x}}_t^- = \hat{\mathbf{F}} \hat{\mathbf{x}}_{t-1}^+ \tag{28}$$

$$\mathbf{P}_t^- = \hat{\mathbf{F}} \mathbf{P}_{t-1}^+ \hat{\mathbf{F}}^T + \mathbf{Q} \tag{29}$$

Correction Step:

$$\hat{\mathbf{K}}_t = \mathbf{P}_t^- \mathbf{H}^T \left[\mathbf{H} \mathbf{P}_t^- \mathbf{H}^T + \mathbf{R}\right]^{-1} \quad \text{(Kalman gain)} \tag{30}$$

$$\hat{\mathbf{x}}_t^+ = \hat{\mathbf{x}}_t^- + \mathbf{K}_t(\mathbf{y}_t - \mathbf{H} \hat{\mathbf{x}}_t^-) \tag{31}$$

$$\hat{\mathbf{P}}_t^+ = (\mathbf{I} - \mathbf{K}_t \mathbf{H}) \mathbf{P}_t^- \tag{32}$$

1.2 Generalization of Kalman filter

We know that Kalman filter requires linear models and Gaussian noise. But this is very restrictive. For the cases where the noises are Gaussian but models are non-linear, we have extended Kalman filter (EKF) and unscented Kalman Filter (UKF). The idea for the extension is somehow make approximation so that we can use Kalman filter.

EKF linearizes the models via first-order Taylor expansion around current estimate.

$$\mathbf{F}_t = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{t-1}^+}, \quad \mathbf{H}_t = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_t^-} \tag{33}$$

Apply Kalman filter equations using time-varying $\mathbf{F}_t, \mathbf{H}_t$. Evolve using time-dependent operators,

$$\hat{\mathbf{x}}_t^- = f(\hat{\mathbf{x}}_{t-1}^+) \tag{34}$$
$$\mathbf{P}_t^- = \mathbf{F}_t \mathbf{P}_{t-1}^+ \mathbf{F}_t^T + \mathbf{Q} \tag{35}$$

Update using $\mathbf{H}_t$ and predicted measurement $\hat{\mathbf{y}}_t^- = h(\hat{\mathbf{x}}_t^-)$.

$$\hat{\mathbf{K}}_t = \mathbf{P}_t^- \mathbf{H}_t^T \left[\mathbf{H}_t \mathbf{P}_t^- \mathbf{H}_t^T + \mathbf{R}\right]^{-1} \quad \text{(Kalman gain)} \tag{36}$$
$$\hat{\mathbf{x}}_t^+ = \hat{\mathbf{x}}_t^- + \mathbf{K}_t(\mathbf{y}_t - \mathbf{H}_t \hat{\mathbf{x}}_t^-) \tag{37}$$
$$\hat{\mathbf{P}}_t^+ = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t)\mathbf{P}_t^- \tag{38}$$

However, there is strong limitations of this approach, namely, only first-order accurate; can diverge for strong nonlinearity; requires Jacobian computation (may be expensive or analytically intractable); assumes Gaussian noises. As for the UKF, propagate uncertainty through nonlinear functions using **sigma points** (also called unscented points) - deterministically chosen samples that capture mean and covariance.

Sigma Points: For state with mean $\hat{\mathbf{x}}$ and covariance $\mathbf{P}$, generate $2n + 1$ points:

$$\mathcal{X}^{(0)} = \hat{\mathbf{x}} \tag{39}$$
$$\mathcal{X}^{(i)} = \hat{\mathbf{x}} + \left(\sqrt{(n+\lambda)\mathbf{P}}\right)_i, \quad i = 1, \ldots, n \tag{40}$$
$$\mathcal{X}^{(i)} = \hat{\mathbf{x}} - \left(\sqrt{(n+\lambda)\mathbf{P}}\right)_{i-n}, \quad i = n+1, \ldots, 2n \tag{41}$$

where $\mathcal{X}^{(i)}$ denotes the i-th sigma point, n is the state dimension, and λ is a scaling parameter. Then pass each sigma point through actual nonlinear function:

$$\mathcal{Y}^{(i)} = h(\mathcal{X}^{(i)}) \tag{42}$$

Compute weighted mean and covariance of transformed points:

$$\hat{\mathbf{y}} = \sum_{i=0}^{2n} W^{(i)} \mathcal{Y}^{(i)} \tag{43}$$
$$\mathbf{P}_y = \sum_{i=0}^{2n} W^{(i)} (\mathcal{Y}^{(i)} - \hat{\mathbf{y}})(\mathcal{Y}^{(i)} - \hat{\mathbf{y}})^T \tag{44}$$

where $W^{(i)}$ are predetermined weights. This approach still assumes both noises are Gaussian but it captures higher order patterns.

1.3 Use particles

Particle based methods are so far the most general approach for our problem, but it will be limited by computational complexity and efficiency.

Represent posterior distribution using N weighted particles (samples):

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx \sum_{i=1}^{N} w_t^{(i)} \delta(\mathbf{x}_t - \mathbf{x}_t^{(i)}) \tag{45}$$

where $\mathbf{x}_t^{(i)}$ is the i-th particle (sample state), $w_t^{(i)}$ is its weight, and $\delta(\cdot)$ is the Dirac delta function. This is essentially a coarse-grained optimal transport problem. Imagine we want to represent the probability distribution with particles, or data points. Then high probability region has higher particle density and low probability has lower particle density. If we have infinite number of particles, then particle density will approach the real probability distribution. However, if we have finite number of particles, we have to compromise, therefore, by not distinguish the particle that are close to each other, we add weights.

$$\sum_{j=1} \delta(\mathbf{x}_t - \mathbf{x}_t^{(j)})\Big|_{|\mathbf{x}_t^{(j)} - \mathbf{x}_t^{(i)}| < \epsilon} \approx w_t^{(i)} \delta(\mathbf{x}_t - \mathbf{x}_t^{(i)}) \tag{46}$$

Algorithm :

Initialize: Sample N particles from the initial state distribution and set uniform weights:

$$\mathbf{x}_0^{(i)} \sim p(\mathbf{x}_0), \quad w_0^{(i)} = \frac{1}{N}, \quad i = 1, \ldots, N \tag{47}$$

where $p(\mathbf{x}_0)$ is the initial state distribution. For example, if the initial state is Gaussian with mean $\mathbf{m}_0$ and covariance $\mathbf{P}_0$, then:

$$\mathbf{x}_0^{(i)} \sim \mathcal{N}(\mathbf{m}_0, \mathbf{P}_0), \quad w_0^{(i)} = \frac{1}{N} \tag{48}$$

1. **Prediction (stochastic):** For each particle $i = 1, \ldots, N$:

$$\mathbf{x}_t^{(i)} = f(\mathbf{x}_{t-1}^{(i)}) + \mathbf{w}_t^{(i)}, \quad \mathbf{w}_t^{(i)} \sim p_w \tag{49}$$

Particles evolve by random sampling from the process noise distribution p_w (can be non-Gaussian). Namely, we can actually write $f(\mathbf{x}_{t-1}^{(i)}) + \mathbf{w}_t^{(i)} \Rightarrow f(\mathbf{x}_{t-1}^{(i)}, \mathbf{w}_t^{(i)})$

2. **Update Weights:** Compute likelihood and normalize:

Unnormalized weights: For each particle $i = 1, \ldots, N$:

$$\tilde{w}_t^{(i)} = w_{t-1}^{(i)} \cdot p(\mathbf{y}_t|\mathbf{x}_t^{(i)}) \tag{50}$$

where $p(\mathbf{y}_t|\mathbf{x}_t^{(i)})$ is the measurement likelihood.

Normalization: Normalize weights to sum to unity:

$$w_t^{(i)} = \frac{\tilde{w}_t^{(i)}}{\sum_{j=1}^{N} \tilde{w}_t^{(j)}} = \frac{w_{t-1}^{(i)} \cdot p(\mathbf{y}_t|\mathbf{x}_t^{(i)})}{\sum_{j=1}^{N} w_{t-1}^{(j)} \cdot p(\mathbf{y}_t|\mathbf{x}_t^{(j)})} \tag{51}$$

Effective sample size: A measure of weight degeneracy:

$$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} (w_t^{(i)})^2} \tag{52}$$

When $N_{\text{eff}} \ll N$, the particle filter suffers from degeneracy and resampling is needed. To see why this gives effects sample size,

$$\frac{1}{N_{\text{eff}}} = \sum_{i=1}^{N} (w_t^{(i)})^2 = \bar{w} \tag{53}$$

where $\bar{w}$ is the average weight for each particle with $N_{\text{eff}} = \frac{1}{\bar{w}}$ number of particles.

3. **Resampling (stochastic):** Resample particles with replacement according to weights to avoid degeneracy (also called particle depletion or weight collapse). High-weight particles get duplicated; low-weight particles die out. This step is typically performed when $N_{\text{eff}} < N_{\text{threshold}}$ (e.g., $N_{\text{threshold}} = N/2$). The most straightforward resampling method is Multinomial resampling, namely, generate N independent samples from the categorical distribution:

$$\mathbf{x}_t^{(i)} \sim \text{Categorical}(\{w_t^{(j)}\}_{j=1}^{N}), \quad i = 1, \ldots, N \tag{54}$$

where particle $\mathbf{x}_t^{(j)}$ is selected with probability $w_t^{(j)}$. After resampling, set all weights to $w_t^{(i)} = 1/N$ for $i = 1, \ldots, N$. In the code, we will use systematic resampling, which is more efficient. For each sample we generate a single uniform random number $u \sim \mathcal{U}(0, 1/N)$ and define:

$$u^{(i)} = u + \frac{i-1}{N}, \quad i = 1, \ldots, N \tag{55}$$

Then select particle indices j_i such that:

$$\sum_{k=1}^{j_i - 1} w_t^{(k)} < u^{(i)} \leq \sum_{k=1}^{j_i} w_t^{(k)} \tag{56}$$

Set $\mathbf{x}_t^{(i)} = \mathbf{x}_t^{(j_i)}$ and $w_t^{(i)} = 1/N$ for all i. The particles relying on the duplicates or multiplicity to mimic the underlying distribution.

4. **Estimate:** Compute weighted average:

$$\hat{\mathbf{x}}_t = \sum_{i=1}^{N} w_t^{(i)} \mathbf{x}_t^{(i)} \tag{57}$$

Advantages: Handles arbitrary nonlinearity; handles non-Gaussian noise in **either or both** process and measurement; can represent multimodal distributions; asymptotically exact as $N \to \infty$.

Limitations: Computationally expensive ($O(Nn^2)$); curse of dimensionality for high-dimensional states ($n > 10$).

1.4 Particle flow

References:

- **Daum (10), Exact particle flow for nonlinear filters**
- **Daum (11), Particle Degeneracy: Root Cause and Solution**

Previously, we have discussed how after each resampling, the information of the distribution is concentrated on the multiplicity distribution of the particles, and the diversity of the particles have been lost. If we look at the particle methods, it's clearly that we either use weights or multiplicity distribution to mimic distribution. The culprit of the particle degeneracy is the consecutive multiplication of small numbers $p(y_t|x_t^{(i)})$ will make a large number of particles irrelevant after view steps. And the observation was done independently and there is no way to control the value of $p(y_t|x_t^{(i)})$. In Daum (10), and Daum (11), the author proposed to use the multiplicity to store the information of the distribution and invented an alternative evolution for the particles such that $p(y_t|x_t^{(i)})$ was never too small. The limitation of these two papers is that to solve the equation, which we will talk about later, they have to assume linear or nearly linearly observation model and dynamics with additive Gaussian noise. The evolution for the particles is complete different equation than underlying dynamics for the ground truth. Later, we will see the methods that generalize the equation for more general noise distribution. Let us first derive the evolution equations. First, we have the prediction step:

$$p(x_t|z_{1:t-1}) = g(x_t) \tag{58}$$

and the "observation" step,

$$p(z_t|x_t) = h(x_t) \tag{59}$$

combine both information, the updated probability for x_t should be,

$$p(x_t|z_{1:t}) \propto g(x_t)h(x_t) \tag{60}$$

The normalization will be handled implicitly by the particles. The difference between the paritlce flow and the particle filter is how you do the update. Since we are moving the particles directly, we demand,

$$\log p(x, \lambda) = \log g(x) + \lambda \log h(x) \tag{61}$$

we always start with $\lambda = 0$, and end with $\lambda = 1$, therefore we can assure the particle flows to the desired position. By definition, the evolution is driven entirely by the observation model. This is where the limitation of the flow model comes in.

Assuming particles evolve in pseudo-time λ according to the ordinary differential equation:

$$\frac{d\mathbf{x}}{d\lambda} = f(\mathbf{x}, \lambda) \tag{62}$$

where $f(\mathbf{x}, \lambda)$ is the flow field that transports particles from the prior distribution at $\lambda = 0$ to the posterior distribution at $\lambda = 1$. This deterministic evolution of particles leads to a corresponding evolution of the probability density $p(\mathbf{x}, \lambda)$.

The evolution of the probability density is governed by the Fokker-Planck equation with zero diffusion. Starting from the continuity equation for probability conservation, and noting that the flow is deterministic (no diffusion term), we obtain:

$$\frac{\partial p}{\partial \lambda} = -\nabla \cdot (pf) \tag{63}$$

This is the zero-diffusion Fokker-Planck equation, which expresses that the rate of change of probability density equals the negative divergence of the probability flux pf. To work with log-probabilities, we divide both sides by p and apply the product rule for divergence, $\nabla \cdot (pf) = p\nabla \cdot f + (\nabla p) \cdot f$, yielding: Recognizing that $\frac{1}{p}\frac{\partial p}{\partial \lambda} = \frac{\partial \log p}{\partial \lambda}$ and $\frac{\nabla p}{p} = \nabla \log p$, we obtain:

$$\frac{\partial \log p}{\partial \lambda} = -\nabla \cdot f - (\nabla \log p) \cdot f \tag{64}$$

From the log-homotopy definition, we have $\frac{\partial \log p}{\partial \lambda} = \log h(x)$, which leads to the fundamental PDE:

$$\boxed{\log h(x) = -\nabla \cdot f - (\nabla \log p) \cdot f} \tag{65}$$

To solve this equation, we assume an affine flow ansatz:

$$f(x, \lambda) = A(\lambda)x + b(\lambda) \tag{66}$$

where $A(\lambda)$ and $b(\lambda)$ are to be determined. The goal is to find expressions for these functions such that particles following this flow evolve from the prior distribution $g(x)$ at $\lambda = 0$ to the posterior distribution proportional to $g(x)h(x)$ at $\lambda = 1$.

We begin by computing the gradient of the log-homotopy. Using the Gaussian form for both $g(x)$ and $h(x)$, and substituting the explicit forms of the prior and likelihood into the log-homotopy:

$$\log p(x, \lambda) = -\frac{1}{2}(x - \bar{\eta}_0)^T P^{-1}(x - \bar{\eta}_0) - \frac{\lambda}{2}(z - Hx)^T R^{-1}(z - Hx) + \text{const} \tag{67}$$

Taking the gradient with respect to x:

$$\begin{aligned} \nabla_x \log p(x, \lambda) &= -P^{-1}(x - \bar{\eta}_0) + \lambda H^T R^{-1}(z - Hx) && (68) \\ &= -P^{-1}x + P^{-1}\bar{\eta}_0 + \lambda H^T R^{-1} z - \lambda H^T R^{-1} Hx && (69) \end{aligned}$$

For the affine flow, the divergence is simply the trace of the matrix A:

$$\nabla \cdot f = \nabla \cdot (Ax + b) = \text{Tr}(A) \tag{70}$$

The dot product $(\nabla \log p) \cdot f$ expands as:

$$\begin{aligned} (\nabla \log p) \cdot f &= [-P^{-1}(x - \bar{\eta}_0) + \lambda H^T R^{-1}(z - Hx)]^T [Ax + b] && (71) \\ &= -(x - \bar{\eta}_0)^T P^{-1} Ax - (x - \bar{\eta}_0)^T P^{-1} b && (72) \\ &\quad + \lambda(z - Hx)^T R^{-1} HAx + \lambda(z - Hx)^T R^{-1} Hb && (73) \end{aligned}$$

The log-likelihood for the linear-Gaussian observation model can be expanded as:

$$\begin{aligned} \log h(x) &= -\frac{1}{2}(z - Hx)^T R^{-1}(z - Hx) + \text{const} && (74) \\ &= -\frac{1}{2} z^T R^{-1} z + z^T R^{-1} Hx - \frac{1}{2} x^T H^T R^{-1} Hx + \text{const} && (75) \end{aligned}$$

A key insight for the EDH flow is that, under Gaussian assumptions, the distribution at any pseudo-time λ remains Gaussian. Specifically, the posterior at λ is:

$$p(x, \lambda) \sim \mathcal{N}(\bar{\eta}_\lambda, P_\lambda) \tag{76}$$

where the precision matrix and mean evolve according to:

$$\begin{aligned} P_\lambda^{-1} &= P^{-1} + \lambda H^T R^{-1} H && (77) \\ \bar{\eta}_\lambda &= P_\lambda [P^{-1}\bar{\eta}_0 + \lambda H^T R^{-1} z] && (78) \end{aligned}$$

This allows us to express the gradient of the log-probability in a compact form:

$$\nabla \log p(x, \lambda) = -P_\lambda^{-1}(x - \bar{\eta}_\lambda) \tag{79}$$

Substituting this into the fundamental PDE, we obtain an alternative formulation:

$$\log h(x) = -\mathrm{Tr}(A) + (x - \bar{\eta}_\lambda)^T P_\lambda^{-1}(Ax + b) \tag{80}$$

A more direct approach to determining $A(\lambda)$ and $b(\lambda)$ comes from requiring that particles starting from a Gaussian distribution remain Gaussian under the flow. For an affine flow $f = Ax + b$, the evolution of the mean and covariance must satisfy:

$$\frac{d\bar{\eta}_\lambda}{d\lambda} = A\bar{\eta}_\lambda + b, \quad \frac{dP_\lambda}{d\lambda} = AP_\lambda + P_\lambda A^T \tag{81}$$

From Kalman filter theory, we know that the precision matrix evolves as:

$$\frac{dP_\lambda^{-1}}{d\lambda} = H^T R^{-1} H \tag{82}$$

Using the identity $\frac{dP_\lambda}{d\lambda} = -P_\lambda \frac{dP_\lambda^{-1}}{d\lambda} P_\lambda$, we find:

$$\frac{dP_\lambda}{d\lambda} = -P_\lambda H^T R^{-1} H P_\lambda \tag{83}$$

Matching this with the covariance evolution equation $AP_\lambda + P_\lambda A^T = -P_\lambda H^T R^{-1} H P_\lambda$, and solving the resulting Riccati-type equation with boundary condition $P_0 = P$, we obtain:

$$\boxed{A(\lambda) = -\frac{1}{2} P_\lambda H^T R^{-1} H = -\frac{1}{2} P H^T (\lambda H P H^T + R)^{-1} H} \tag{84}$$

For the mean evolution, we require that $\frac{d\bar{\eta}_\lambda}{d\lambda} = H^T R^{-1}(z - H\bar{\eta}_\lambda)$ equals $A\bar{\eta}_\lambda + b$. Solving for b:

$$b(\lambda) = P_\lambda H^T R^{-1} z - A\bar{\eta}_\lambda \tag{85}$$

After algebraic manipulation, this simplifies to:

$$\boxed{b(\lambda) = (I + 2\lambda A)[(I + \lambda A) P H^T R^{-1} z + A\bar{\eta}_0]} \tag{86}$$

The EDH flow equations are derived by requiring that Gaussian distributions remain Gaussian under the flow, that the flow satisfies the Fokker-Planck equation, and that the boundary conditions $p(x, 0) = g(x)$ and $p(x, 1) \propto g(x)h(x)$ are satisfied. The resulting solution provides an **exact** method for transporting particles from the prior to the posterior in the linear-Gaussian case, avoiding the weight degeneracy issues that plague standard particle filters. As for the LEDH, the difference is that each particle has their own equation. That is because the expansion of the model are taken at their individual location instead of the average position.

1.5 (Invertible) Particle flow particle filter

References:

- **Particle Filtering with Invertible Particle Flow (Li, Coates, 17)**

The particle flow particle filter method is directly built up on the EDH and LEDH. The improvement is to add weight redistribution for each particle. This will introduce addition expressiveness to the framework. There is however, one ore subtle point, which is the correction of the weight.

Of course in principle, we can also track the effective sample size. This will be added later. Here we want to introduce a concept call the proposed distribution q. So far, we have not met this concept, but later for the particle flow filter kernel method, we will see that this is also important.

For any filtering algorithm at time step k, we must distinguish between:

- **Theoretical target**: The true Bayesian posterior

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{\int p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})d\mathbf{x}_k} \tag{87}$$

- **Procedural distribution**: What the algorithm actually produces

$$\hat{p}(\mathbf{x}_k|\mathbf{z}_{1:k}) \tag{88}$$

- **Correction mechanism**: How the algorithm accounts for discrepancies between target and procedure

The **proposal distribution** $q(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_k)$ is the distribution from which candidate states are drawn. The relationship to Bayes' rule determines the weight correction needed.

For the Kalman filter, there is not discrepancy between the theoretical target and the proposed candidate. While there is a discrepancy between these two in EKL and UKF, there is **no mechanism** within the EKF/UKF framework to detect or compensate for these errors. The algorithm blindly propagates Gaussian moments regardless of approximation quality.

1.5.1 Particle Flow Particle Filter (EDH/LEDH)

Setup: Combine prediction step (standard PF) with flow-based update.

Theoretical target: Same posterior $p(\mathbf{x}_k|\mathbf{z}_{1:k})$.

Exact Daum-Huang (EDH) Flow: Assumes additive Gaussian observation noise:

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k, \quad \mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k) \tag{89}$$

The log-likelihood is:

$$\log p(\mathbf{z}_k|\mathbf{x}_k) = -\frac{1}{2}(\mathbf{z}_k - h(\mathbf{x}_k))^T \mathbf{R}_k^{-1}(\mathbf{z}_k - h(\mathbf{x}_k)) + \text{const} \tag{90}$$

Gradient:

$$\nabla_{\mathbf{x}} \log p(\mathbf{z}_k|\mathbf{x}_k) = \mathbf{H}(\mathbf{x}_k)^T \mathbf{R}_k^{-1}(\mathbf{z}_k - h(\mathbf{x}_k)) \tag{91}$$

where $\mathbf{H}(\mathbf{x}_k) = \partial h/\partial \mathbf{x}$.

EDH flow velocity: Linearize $h(\mathbf{x})$ around ensemble mean $\bar{\mathbf{x}}(\lambda)$:

$$h(\mathbf{x}) \approx h(\bar{\mathbf{x}}) + \mathbf{H}(\bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}}) \tag{92}$$

The EDH flow is:

$$\mathbf{g}(\mathbf{x}, \lambda) = \mathbf{C}_{\mathbf{xz}}(\lambda)[\mathbf{C}_{\mathbf{zz}}(\lambda) + \mathbf{R}_k]^{-1}[\mathbf{z}_k - \bar{\mathbf{z}}(\lambda)] \tag{93}$$

where

$$\bar{\mathbf{z}}(\lambda) = \frac{1}{N}\sum_{i=1}^{N} h(\mathbf{x}_k^{(i)}(\lambda)) \tag{94}$$

$$\mathbf{C}_{\mathbf{xz}}(\lambda) = \frac{1}{N}\sum_{i=1}^{N} (\mathbf{x}_k^{(i)}(\lambda) - \bar{\mathbf{x}}(\lambda))(h(\mathbf{x}_k^{(i)}(\lambda)) - \bar{\mathbf{z}}(\lambda))^T \tag{95}$$

$$\mathbf{C}_{\mathbf{zz}}(\lambda) = \frac{1}{N}\sum_{i=1}^{N} (h(\mathbf{x}_k^{(i)}(\lambda)) - \bar{\mathbf{z}}(\lambda))(h(\mathbf{x}_k^{(i)}(\lambda)) - \bar{\mathbf{z}}(\lambda))^T \tag{96}$$

Key property of EDH: The flow velocity is **globally linear in state**:

$$\mathbf{g}(\mathbf{x}, \lambda) = \mathbf{K}(\lambda)[\mathbf{z}_k - \bar{\mathbf{z}}(\lambda) - \mathbf{H}(\bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})] \tag{97}$$

where $\mathbf{K}(\lambda)$ depends on ensemble statistics but not individual $\mathbf{x}$.

This means:

$$\frac{\partial \mathbf{g}}{\partial \mathbf{x}} = \text{constant across all particles} \tag{98}$$

Therefore, the Jacobian is **particle-independent**:

$$\mathbf{J}_i = \mathbf{J} \quad \forall i \tag{99}$$

Weight update for EDH:

$$\tilde{w}_k^{(i)} \propto w_{k|k-1}^{(i)} \cdot |\det \mathbf{J}| \tag{100}$$

After normalization:

$$w_k^{(i)} = \frac{w_{k|k-1}^{(i)} |\det \mathbf{J}|}{\sum_{j=1}^{N} w_{k|k-1}^{(j)} |\det \mathbf{J}|} = \frac{w_{k|k-1}^{(i)}}{\sum_{j=1}^{N} w_{k|k-1}^{(j)}} = w_{k|k-1}^{(i)} \tag{101}$$

The Jacobian cancels. If prior weights were uniform, posterior weights remain uniform.

Local Exact Daum-Huang (LEDH): Linearize around **each individual particle** instead of ensemble mean:

$$h(\mathbf{x}) \approx h(\mathbf{x}_k^{(i)}) + \mathbf{H}(\mathbf{x}_k^{(i)})(\mathbf{x} - \mathbf{x}_k^{(i)}) \tag{102}$$

The LEDH flow becomes:

$$\mathbf{f}^{(i)}(\mathbf{x}, \lambda) = \mathbf{C}_{\mathbf{xz}}^{(i)}(\lambda)[\mathbf{C}_{\mathbf{zz}}^{(i)}(\lambda) + \mathbf{R}_k]^{-1}[\mathbf{z}_k - h(\mathbf{x}_k^{(i)}(\lambda))] \tag{103}$$

Now the flow is **particle-specific**:

$$\mathbf{f}^{(i)} \neq \mathbf{f}^{(j)} \quad \text{for } i \neq j \tag{104}$$

Each particle has its own Jacobian:

$$\mathbf{J}_i = \left.\frac{\partial \Psi^{(i)}}{\partial \mathbf{x}}\right|_{\mathbf{x}_k^{(i)}(0)} \tag{105}$$

Weight update for LEDH:

$$\tilde{w}_k^{(i)} \propto w_{k|k-1}^{(i)} \cdot |\det \mathbf{J}_i| \tag{106}$$

Jacobians are **different**, so:

$$w_k^{(i)} = \frac{w_{k|k-1}^{(i)} |\det \mathbf{J}_i|}{\sum_{j=1}^{N} w_{k|k-1}^{(j)} |\det \mathbf{J}_j|} \tag{107}$$

Jacobians do NOT cancel. Weights become non-uniform even if they started uniform.

The particle flow ODE

$$\frac{d\mathbf{x}(\lambda)}{d\lambda} = \mathbf{f}(\mathbf{x}(\lambda), \lambda), \quad \lambda \in [0, 1] \tag{108}$$

is approximated by discretizing the pseudo-time interval into N_λ steps: $0 = \lambda_0 < \lambda_1 < \cdots < \lambda_{N_\lambda} = 1$.

At each discrete step j, the flow velocity is linearized:

$$\mathbf{f}(\mathbf{x}, \lambda_j) \approx \mathbf{A}_j \mathbf{x} + \mathbf{b}_j \tag{109}$$

The Euler integration step from λ_{j-1} to λ_j is:

$$\mathbf{x}(\lambda_j) = \mathbf{x}(\lambda_{j-1}) + \Delta\lambda_j \cdot [\mathbf{A}_j \mathbf{x}(\lambda_{j-1}) + \mathbf{b}_j] \tag{110}$$

where $\Delta\lambda_j = \lambda_j - \lambda_{j-1}$.

Rearranging as an affine map:

$$\mathbf{x}(\lambda_j) = (\mathbf{I} + \Delta\lambda_j \mathbf{A}_j)\mathbf{x}(\lambda_{j-1}) + \Delta\lambda_j \mathbf{b}_j \tag{111}$$

Define:

$$\mathbf{M}_j = \mathbf{I} + \Delta\lambda_j \mathbf{A}_j \quad \text{(linear part)} \tag{112}$$

$$\mathbf{c}_j = \Delta\lambda_j \mathbf{b}_j \quad \text{(translation)} \tag{113}$$

So each discrete step is:

$$\mathbf{x}(\lambda_j) = \mathbf{M}_j \mathbf{x}(\lambda_{j-1}) + \mathbf{c}_j \tag{114}$$

1.5.2 Jacobian of an Affine Map

For an affine map $\mathbf{y} = \mathbf{M}\mathbf{x} + \mathbf{c}$:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbf{M} \tag{115}$$

The translation $\mathbf{c}$ disappears because $\frac{\partial \mathbf{c}}{\partial \mathbf{x}} = \mathbf{0}$.

Intuition: The Jacobian measures how a small change in input $\mathbf{x}$ affects the output $\mathbf{y}$. Translations shift everything uniformly, so they don't affect the rate of change.

For composed functions $\mathbf{x}_{N_\lambda} = f_{N_\lambda} \circ f_{N_\lambda - 1} \circ \cdots \circ f_1(\mathbf{x}_0)$, the chain rule gives:

$$\frac{\partial \mathbf{x}_{N_\lambda}}{\partial \mathbf{x}_0} = \frac{\partial \mathbf{x}_{N_\lambda}}{\partial \mathbf{x}_{N_\lambda - 1}} \cdot \frac{\partial \mathbf{x}_{N_\lambda - 1}}{\partial \mathbf{x}_{N_\lambda - 2}} \cdots \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1} \cdot \frac{\partial \mathbf{x}_1}{\partial \mathbf{x}_0} \tag{116}$$

For our affine maps:

$$\frac{\partial \mathbf{x}_{N_\lambda}}{\partial \mathbf{x}_0} = \mathbf{M}_{N_\lambda} \cdot \mathbf{M}_{N_\lambda - 1} \cdots \mathbf{M}_2 \cdot \mathbf{M}_1 = \prod_{j=N_\lambda}^{1} \mathbf{M}_j \tag{117}$$

Substituting $\mathbf{M}_j = \mathbf{I} + \Delta\lambda_j \mathbf{A}_j$:

$$\mathbf{J} = \frac{\partial \mathbf{x}_{N_\lambda}}{\partial \mathbf{x}_0} = \prod_{j=N_\lambda}^{1} (\mathbf{I} + \Delta\lambda_j \mathbf{A}_j) \tag{118}$$

This is the **Jacobian matrix** of the flow map.

1.5.3 Determinant of the Jacobian

The Jacobian determinant is:

$$\det(\mathbf{J}) = \det\left(\prod_{j=N_\lambda}^{1} (\mathbf{I} + \Delta\lambda_j \mathbf{A}_j) \right) \tag{119}$$

Using the property $\det(\mathbf{AB}) = \det(\mathbf{A}) \det(\mathbf{B})$:

$$\det(\mathbf{J}) = \prod_{j=1}^{N_\lambda} \det(\mathbf{I} + \Delta\lambda_j \mathbf{A}_j) \tag{120}$$

Taking absolute value for the weight update:

$$|\det(\mathbf{J})| = \prod_{j=1}^{N_\lambda} |\det(\mathbf{I} + \Delta\lambda_j \mathbf{A}_j)| \tag{121}$$

1.5.4 Summary: Computational Recipe

To compute the Jacobian determinant for discretized particle flow:

1. At each pseudo-time step $j = 1, \ldots, N_\lambda$:
   - Compute flow parameters $\mathbf{A}_j$ and $\mathbf{b}_j$ via linearization
   - Form the matrix $\mathbf{M}_j = \mathbf{I} + \Delta\lambda_j \mathbf{A}_j$
   - Compute $\det(\mathbf{M}_j)$
2. Multiply all determinants:

$$|\det(\mathbf{J})| = \prod_{j=1}^{N_\lambda} |\det(\mathbf{M}_j)| \tag{122}$$

1.5.5 Connection to Weight Update

$$w_k^{(i)} \propto \frac{p(\eta_1^{(i)}|\mathbf{x}_{k-1}^{(i)}) \cdot p(\mathbf{z}_k|\eta_1^{(i)})}{p(\eta_0^{(i)}|\mathbf{x}_{k-1}^{(i)}) \cdot |\det \mathbf{J}_i|} \cdot w_{k-1}^{(i)} \tag{123}$$

where:

- $\mathbf{x}_{k-1}^{(i)}$ is the particle state at time $k-1$
- $\eta_0^{(i)}$ is the particle after propagation through dynamics (before flow)
- $\eta_1^{(i)}$ is the particle after particle flow (final state at time k)
- $\mathbf{z}_k$ is the observation at time k
- $|\det \mathbf{J}_i| = \prod_{j=1}^{N_\lambda} \left|\det(\mathbf{I} + \Delta\lambda_j \mathbf{A}_j^{(i)})\right|$ is the Jacobian determinant

What this does is two fold, what want to use the flow equation to move the particles, however, in a perfect world, we would have the the probability of the evolved sample equal to the initial sample times the Jacobian, however this is not the case in practice. I want to make sure we are incorporating the information about the numerator, therefore, we need to correct the discrepancy.

1.6 Particle flow filter (kernel method)

References:

- **Sequential Monte Carlo with kernel embedded mappings: The mapping particle filter (Pulidoa, Leeuwen, 19)**
- **A particle flow filter for high-dimensional system applications (Hu, Leeuwen, 21)**

Our problem is the same as particle filter particle flow, move the particles to the desired location, without touching the weights, so that we get the target probability distribution $p(x_t|y_{1:t}) \propto p(y_t|x_t)p(x_t|y_{1:t-1})$.

1.6.1 Optimal Transport via Gradient Flow

Define target posterior $\pi(x) := p(x_t|y_{1:t})$ and intermediate density $q_\lambda(x)$ at pseudo-time $\lambda \in [0,1]$. With $q_0 = p(x_t|y_{1:t-1})$ (prior), $q_1 = \pi$ (posterior).

$$\frac{dx_\lambda}{d\lambda} = v_\lambda(x_\lambda) \tag{124}$$

Choose velocity v_λ to minimize KL divergence as fast as possible:

$$D_{KL}(q_\lambda \| \pi) = \int q_\lambda(x) \log \frac{q_\lambda(x)}{\pi(x)} dx \tag{125}$$

we need to formulate the above equation into an optimization problem. But the intuition of this method is clear, you want to evolve the point along a "straight" direction to the desire point.

1.6.2 Deriving Optimal Velocity via Kernel Restriction

Under velocity field v, density evolves via Liouville equation:

$$\frac{\partial q_\lambda}{\partial \lambda} = -\nabla_x \cdot (q_\lambda v) \tag{126}$$

Time derivative of D_{KL}:

$$\frac{d}{d\lambda} D_{KL} = \int \frac{\partial q_\lambda}{\partial \lambda} \left[\log q_\lambda(x) - \log \pi(x) + 1\right] dx \tag{127}$$

$$= -\int \nabla_x \cdot (q_\lambda v) \left[\log q_\lambda(x) - \log \pi(x) + 1\right] dx \tag{128}$$

Using $\nabla_x q_\lambda = q_\lambda \nabla_x \log q_\lambda$ and integrating by parts:

$$\frac{d}{d\lambda} D_{KL} = -\int q_\lambda \left[\nabla_x \log \pi(x) \cdot v + \nabla_x \cdot v\right] dx \tag{129}$$

The derivative with respect to the evolution parameter gives the vector field along the trajectory. To make things computatble, we restrict v to functions generated by kernel $K(x, x')$. Any such function has form:

$$v(x) = \int \alpha(x') K(x', x) dx' \tag{130}$$

for some coefficient function $\alpha(x')$. Substitute this into directional derivative.

$$\frac{d}{d\lambda} D_{KL} = -\int q_\lambda(x) \left[\nabla_x \log \pi(x) \cdot \int \alpha(x') K(x', x) dx' + \nabla_x \cdot \int \alpha(x') K(x', x) dx'\right] dx \tag{131}$$

Interchange integrals and use integration by parts:

$$\frac{d}{d\lambda} D_{KL} = \int \alpha(x) \cdot \phi(x) dx \tag{132}$$

where

$$\phi(x) = -\int q_\lambda(x') \left[K(x, x') \nabla_{x'} \log \pi(x') + \nabla_{x'} K(x, x')\right] dx' \tag{133}$$

We have $\frac{d}{d\lambda} D_{KL} = \langle \alpha, \phi \rangle$ (inner product in function space).

To minimize this quantity (make KL decrease fastest), choose α antiparallel to ϕ. By Cauchy-Schwarz, the minimum is achieved when:

$$\alpha(x) = -\phi(x) \tag{134}$$

Therefore, optimal velocity field:

$$v_\lambda(x) = \int \alpha(x') K(x', x) dx' \tag{135}$$

$$= -\int \phi(x') K(x', x) dx' \tag{136}$$

By the reproducing property of the kernel, this simplifies to:

$$\boxed{v_\lambda(x) = \mathbb{E}_{x' \sim q_\lambda}\left[K(x, x')\nabla_{x'} \log \pi(x') + \nabla_{x'} K(x, x')\right]} \tag{137}$$

Monte Carlo approximation with particles $\{x_\lambda^{(i)}\}_{i=1}^{N_p} \sim q_\lambda$:

$$v_\lambda(x) \approx \frac{1}{N_p} \sum_{i=1}^{N_p} \left[K(x_\lambda^{(i)}, x)\nabla \log \pi(x_\lambda^{(i)}) + \nabla K(x_\lambda^{(i)}, x)\right] \tag{138}$$

Posterior gradient:

$$\nabla \log \pi(x) = \nabla \log p(y_t|x) + \nabla \log p(x|y_{1:t-1}) \tag{139}$$

First term = likelihood gradient (known). Second term = prior gradient (approximated from ensemble).

Update rule:

$$x_{\lambda+\epsilon}^{(j)} = x_\lambda^{(j)} + \frac{\epsilon}{N_p} \sum_{i=1}^{N_p} \left[K(x_\lambda^{(i)}, x_\lambda^{(j)})\nabla \log \pi(x_\lambda^{(i)}) + \nabla K(x_\lambda^{(i)}, x_\lambda^{(j)})\right] \tag{140}$$

Key advantage: All particles maintain equal weight $w^{(i)} = 1/N_p$ throughout. No resampling needed.

Generality: Works for any differentiable likelihood $p(y_t|x_t)$ - not limited to Gaussian noise.

1.6.3 Scalar Kernel

Standard choice: isotropic Gaussian

$$K(x, x') = \exp\left(-\frac{1}{2\alpha\sigma^2}\|x - x'\|^2\right), \quad \alpha \sim n_x \tag{141}$$

Gradient: $\nabla_x K(x, x') = -\frac{1}{\alpha\sigma^2}(x - x')K(x, x')$

Single scalar $K(x, x')$ weights all components identically.

Failure in high dimensions with sparse observations:

Consider 1000-D state, 250 observations (25% coverage):

1. Observed components converge rapidly (strong likelihood gradient)
2. Unobserved components remain dispersed (weak/no gradient)
3. Distance $\|x^{(i)} - x^{(j)}\|$ dominated by unobserved components
4. Even when observed components nearly coincide, $K(x^{(i)}, x^{(j)}) \approx 0$
5. Repulsion $\nabla K \propto K \approx 0$ in all directions
6. Observed components collapse to posterior mode

Global distance cannot distinguish heterogeneous convergence rates.

1.6.4 Matrix-Valued Kernel

Measure distance independently per component:

$$\mathbf{K}(x, x') = \text{diag}\left[K^{(1)}(x, x'), \ldots, K^{(n_x)}(x, x')\right] \tag{142}$$

where

$$K^{(d)}(x, x') = \exp\left(-\frac{(x^{(d)} - x'^{(d)})^2}{2\alpha(\sigma^{(d)})^2}\right), \quad \alpha = \frac{1}{N_p} \tag{143}$$

Component-wise repulsion:

$$\left[\nabla_x \cdot \mathbf{K}(x, x')\right]^{(d)} = -\frac{x^{(d)} - x'^{(d)}}{\alpha(\sigma^{(d)})^2} K^{(d)}(x, x') \tag{144}$$

- Observed component d: $x^{(d)} \approx x'^{(d)} \Rightarrow K^{(d)} \approx 1 \Rightarrow$ strong repulsion
- Unobserved component d': $x^{(d')} \not\approx x'^{(d')} \Rightarrow K^{(d')} \approx 0 \Rightarrow$ weak repulsion
- Repulsion in dimension d independent of distances in other dimensions

Observed dimensions maintain diversity regardless of unobserved dimension spread.

2 Part one

2.1 Kalman filter

2.1.1 1-d filter implementation

This is just to set the foundation for later experiments. This is a one-dimensional linear-Gaussian state-space model:

$$x_{t+1} = 0.9x_t + 0.5 + w_t, \quad w_t \sim \mathcal{N}(0,1) \tag{145}$$

$$y_t = 1.1x_t + 0.3v_t, \quad v_t \sim \mathcal{N}(0,1) \tag{146}$$

with initial state $x_0 \sim \mathcal{N}(0,1)$ and $T = 15,000$ time steps. The filter uses the standard covariance update form (not Joseph form).

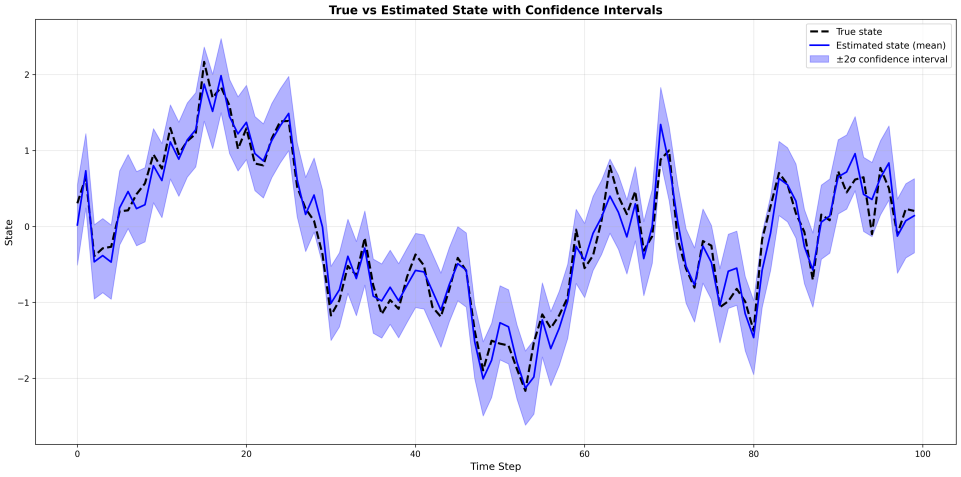

Figure 1:

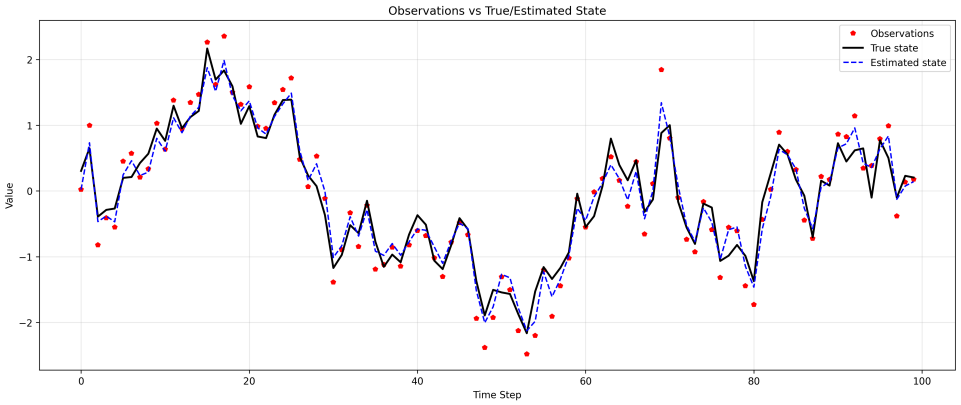

Figure 2:

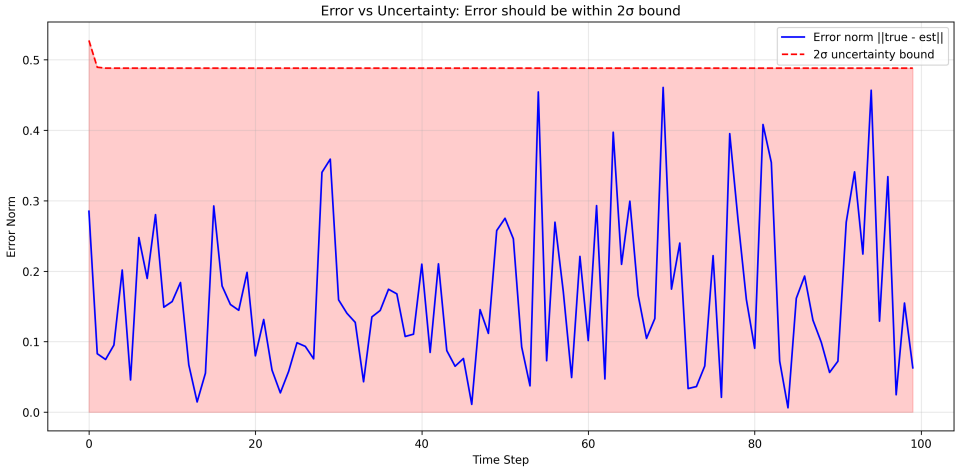

Figure 3:

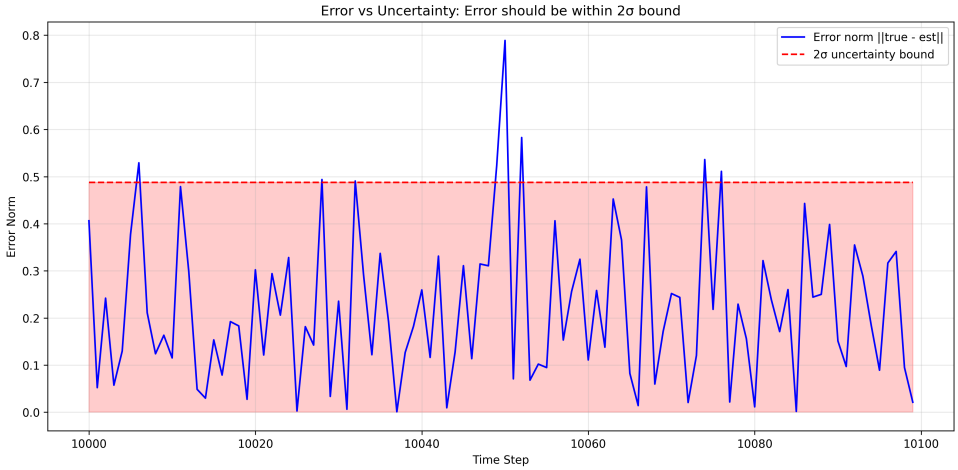

Figure 4:

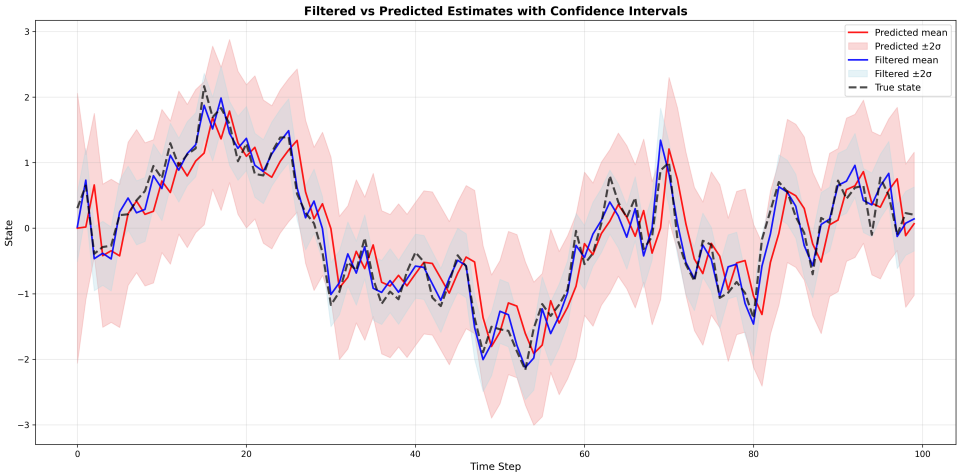

Figure 5:

We see that the filter worked very well.

2.1.2 Joseph form vs standard covariance update

Standard Covariance Update:

$$P_{t|t} = (I - K_t H)P_{t|t-1} \tag{147}$$

where $K_t = P_{t|t-1}H^T S_t^{-1}$ and $S_t = HP_{t|t-1}H^T + R$.

Tiny Observation Noise ($R \to 0$)

When $R = \epsilon I$ with $\epsilon \ll 1$:

$$S_t = HP_{t|t-1}H^T + \epsilon I \tag{148}$$

The condition number is:

$$\kappa(S_t) = \frac{\lambda_{\max}(HP_{t|t-1}H^T) + \epsilon}{\epsilon} \approx \frac{\lambda_{\max}}{\epsilon} \tag{149}$$

For $\epsilon = 10^{-10}$ and $\lambda_{\max} \sim 1$, we have $\kappa(S_t) \sim 10^{10}$.

Computing S_t^{-1} in finite precision with machine epsilon $\epsilon_{\text{mach}} \sim 10^{-16}$:

$$\text{Relative error in } S_t^{-1} \sim \kappa(S_t) \cdot \epsilon_{\text{mach}} \sim 10^{10} \times 10^{-16} = 10^{-6} \tag{150}$$

This propagates to K_t and then to:

$$P_{t|t} = P_{t|t-1} - K_t S_t K_t^T + \mathcal{O}(\text{error}) \tag{151}$$

Small errors in K_t destroy positive definiteness through the subtraction.

Why Joseph Form Fixes This

The Joseph form:

$$P_{t|t} = (I - KH)P_{t|t-1}(I - KH)^T + KRK^T \tag{152}$$

To demonstrate why this works, we go back to one dimension.

$$P^+ = (1 - KH)^2 P^- + K^2 R \tag{153}$$

This structure guarantees $P_{t|t} \geq 0$ regardless of numerical errors in K. Even if K is computed incorrectly due to ill-conditioning, the form preserves positive definiteness, whereas the standard form requires precise cancellation in $P - KHP$ which fails under poor conditioning. Result is that, only tiny Observation Noise will break it.

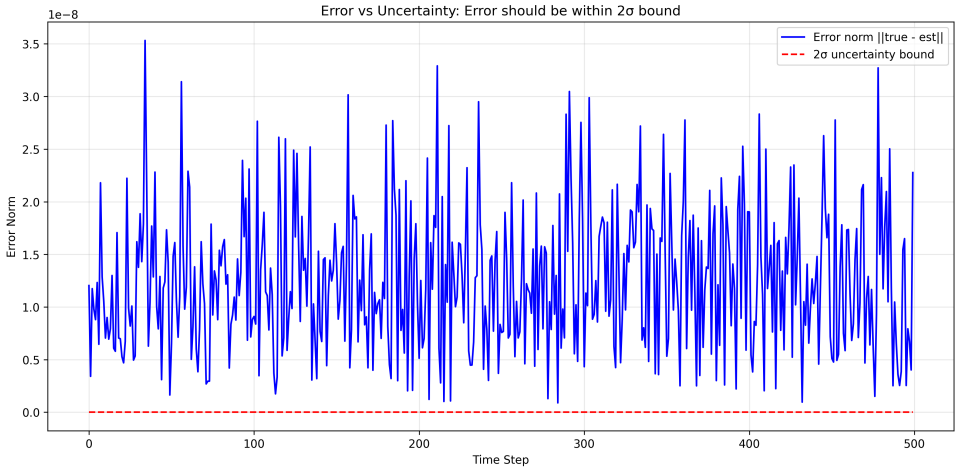

Figure 6: Standard covariance update with tiny observation noise ($R = 10^{-8}I$).

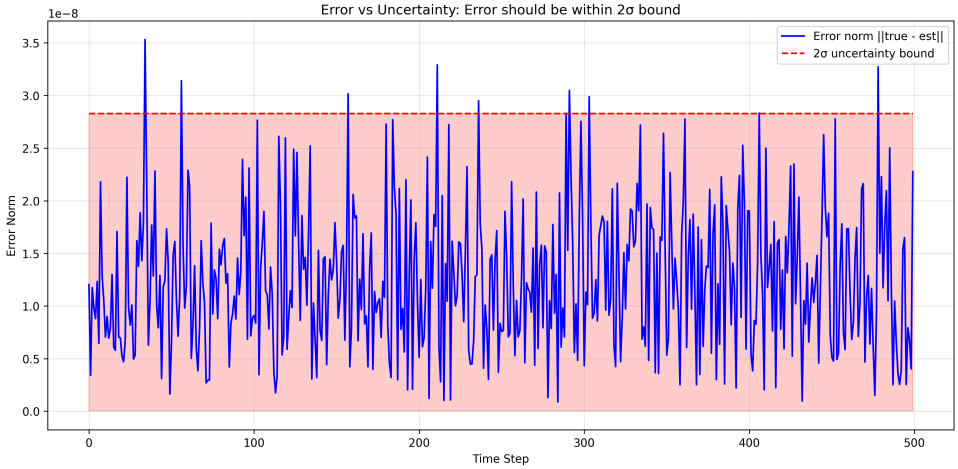

Figure 7: Joseph form covariance update with tiny observation noise ($R = 10^{-8}I$).

We clearly see that Joseph covariance update will solve this numerical instability. There are 3 more cases where I did the test but not included here, mostly due to the fact that original update worked fine.

2.2 EKF, UKF and particle filter

- Non-linearality completely broke EKF and UKF, but particle filter works.
- For the range-bearing, all three of them works but degeneracy problem is much worse due to high dimension. (Implemented in the code, did some preliminary test, did not have time to do comprehensive tests)

We use the following Stochastic Volatility model,

$$x_{t+1} = 0.91x_t + w_t, \quad w_t \sim \mathcal{N}(0,1) \tag{154}$$

$$y_t = 0.5\exp(x_t/2) \cdot v_t, \quad v_t \sim \mathcal{N}(0,1) \tag{155}$$

with the initial condition drawn from $N(0, \frac{\sigma^2}{1-\alpha^2})$, where $\alpha = 0.91, \sigma = 1$. $EKF : 0.1101 seconds, UKF : 0.1975 seconds, PF : 84.2930 seconds$

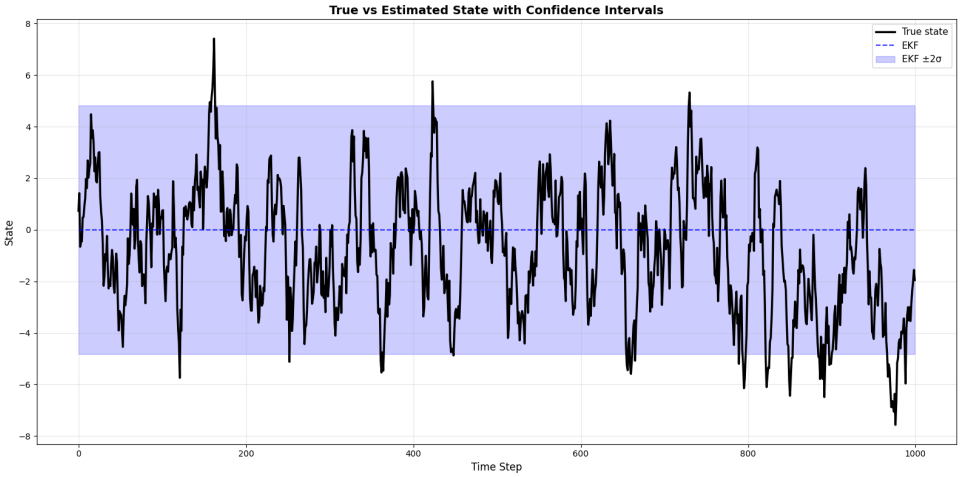

Figure 8: Extended Kalman Filter (EKF) results.

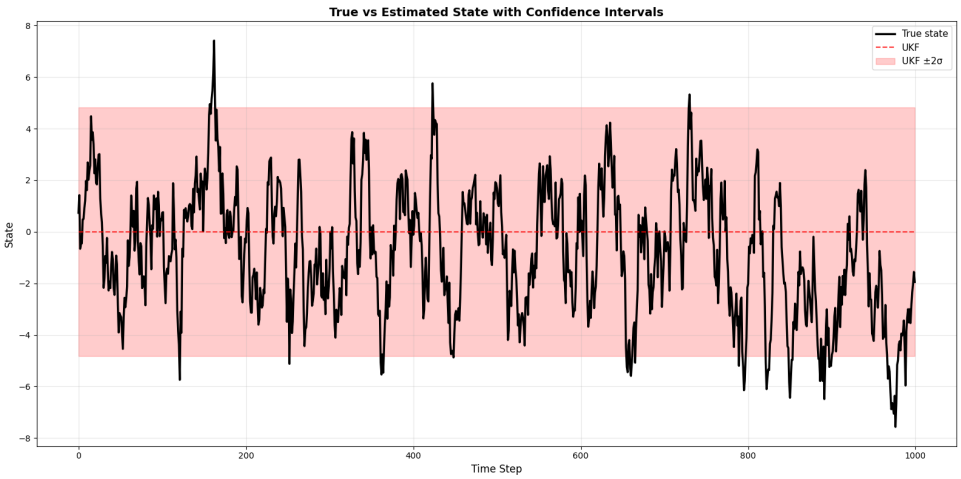

Figure 9: Unscented Kalman Filter (UKF) results.

Both Kalman-like methods failed completely. There is no update at all, as a matter of fact, the mean states at zero and variance stays constant. It's easy to see that these are just the initial condition of the filter. This is due to the multiplicative nature of the noise, the h of the process was mistaken as zero, which leads to zero Kalman gain.

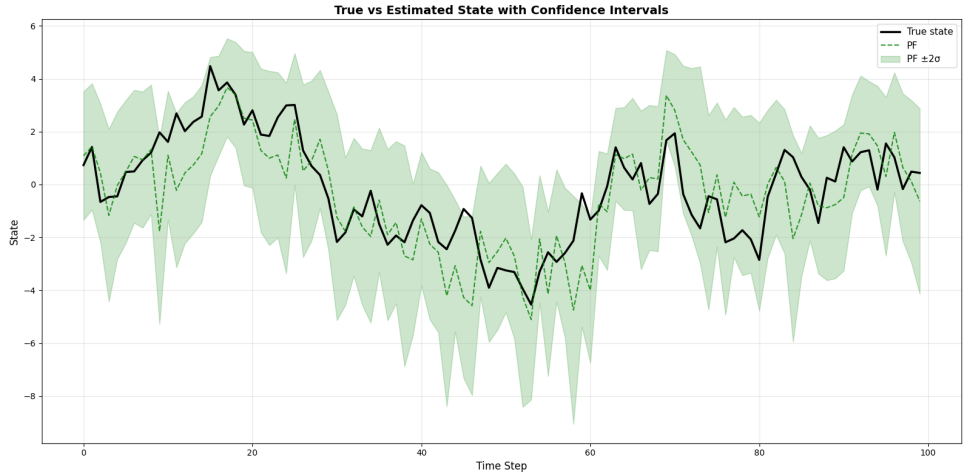

Figure 10: Particle Filter result compared with ground truth with uncertainty

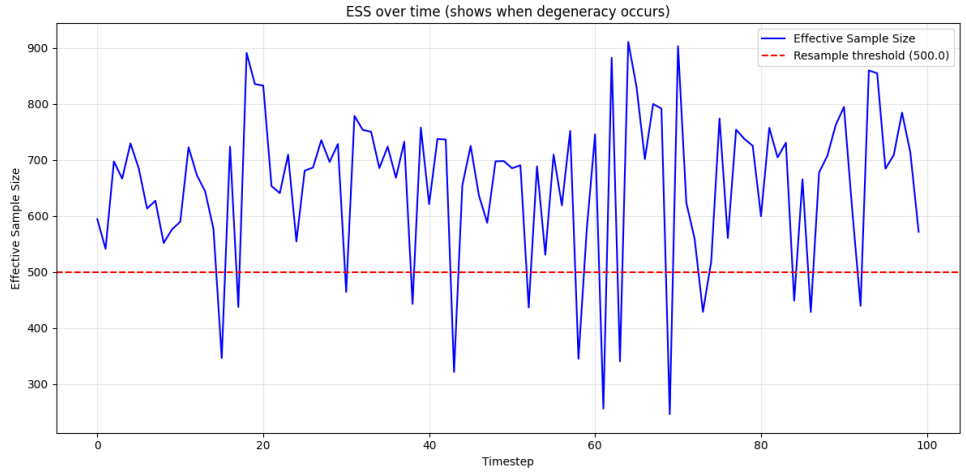

Figure 11: Particle Filter effective sample size. We see that ESS constantly drops below half of the total particle number which mean we need to resample

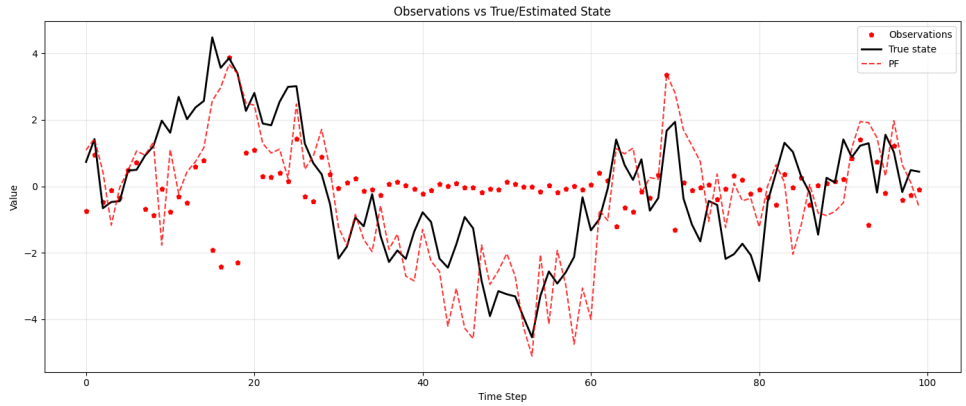

Figure 12: Particle Filter result compared with ground truth with observation

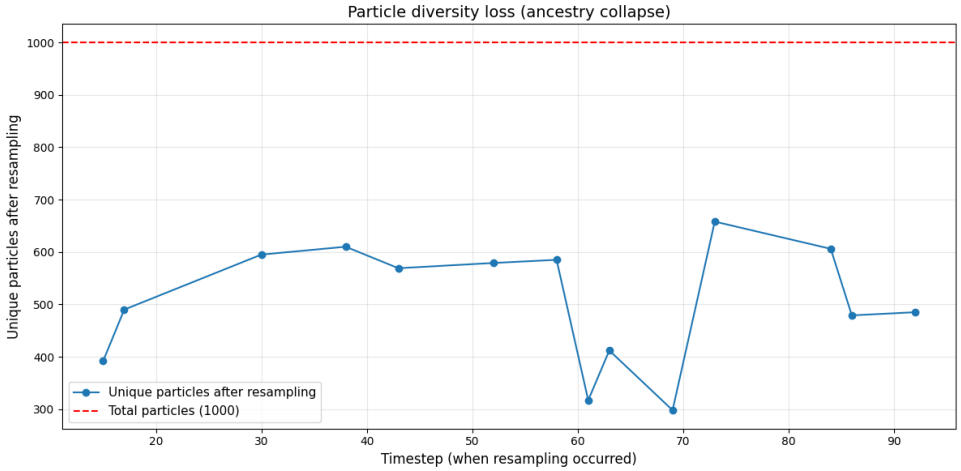

Figure 13: Number of unique particles as a function of time

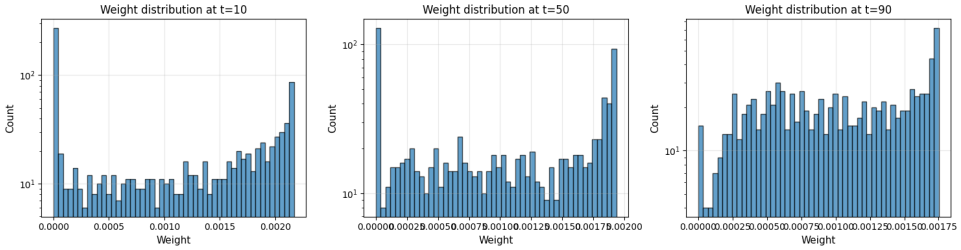

Figure 14: Particle weight distribution. We see large weight particles dominate

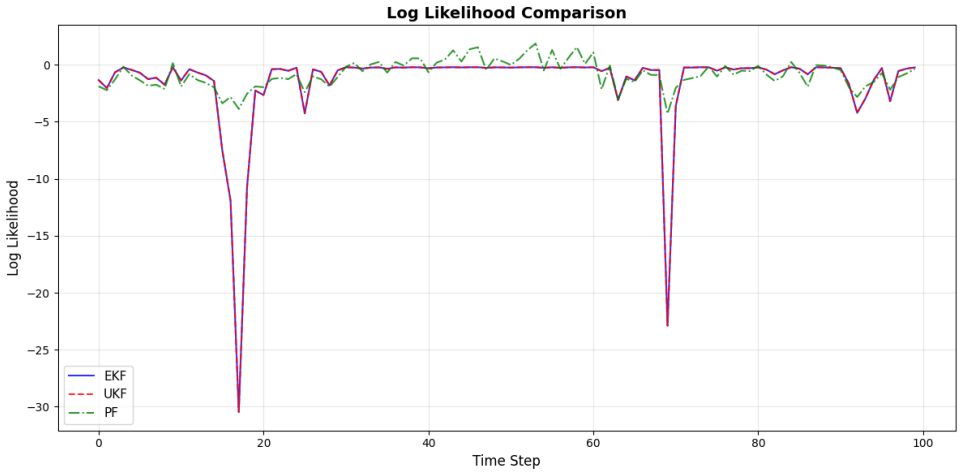

Figure 15: Log likelihood as a function of time. Comparison between all three filter

we see that for most of the time, particle filter log likelihood is above the other two filter. More experiments should be taken, for example, maybe the cross over point corresponds to resampling. Log likelihood is just the $\log p(y_t|y_{1:t-1})$, the better the filter, the higher the probability to predict the next observation.

2.3 All about particle flow

2.3.1 EDH, LEDH and PF-PF

2.3.2 PFF-kernel method

3 Part two

4 Bonues