Particle Flow Filter and Differentiable Particle Filter

Zhongchangfei LI
zlihc@connect.ust.hk

Abstract

This report summarizes classic algorithms for filtering task in state-space model (SSM), where the noisy observations are determined by unobservable hidden states and random observation noise, while the hidden states transition according to some stochastic process. The report covers contents ranging from the most basic Kalman filter for linear Gaussian state-space model (LGSSM) and its classic extensions, to particle filters and its advanced extensions particle flow filters. The classic Kalman filter requires linear model and additive Gaussian noise, while the extend Kalman filter (EKF) and unscent Kalman filter (UKF) generalize the classic one to deal with nonlinear models by local linearization (EKF) or statistic moment matching (UKF), which still fail for highly nonlinear tasks. Particle Filter do not make assumptions on the model but use a large number of particles to approximate the model distributions, which can be applicable to nonlinear model but will fail when the task is a high dimensional problem, in which case a large number of particles will be placed in the region with only tiny probability distribution. Resampling can solve the particle degeneracy problem but will lead to sample impoverishment problem, where the particle collapse with a small effective sample size. Thus, particle flow filters are proposed to avoid using explicit weights and resampling, but approach the posterior distribution by dynamics. The details can be found in the report and the codebase: `https://github.com/FlyingBeyondUp/JPM-MLCOE-2026`.

.

1 State-Space Model

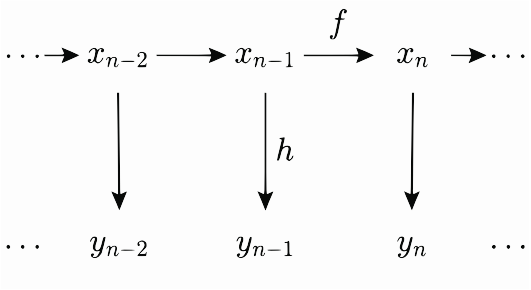


Figure 1: Schematic Figure for State-Space Model

Fig. 1 is a schematic figure of SSM, where the unobservable hidden state x_n at time step n transition according to the dynamics defined by function f and the observations y_n is given by the hidden states according to function h. Denote the transition noise by q_n, and observation noise by r_n, the SSM can be depicted by

$$\begin{aligned} x_n &= f(x_{n-1}, q_{n-1}) \\ y_n &= h(x_{n-1}, r_{n-1}) \end{aligned} \tag{1}$$

Therefore, the probability density $p(x_n|x_{n-1})$ is also called transition density while $p(y_n|x_n)$ is called emission density. The filter task can be modeled as a Bayesian inference problem, if at time step n, we have an approximate posterior probability $p(x_n|y_n)$, then the hidden state can be estimated by $\hat{x}_n \approx \mathrm{E}[x_n|y_n]$. The hidden state at the next step can be predicted by the probability distribution

$$p(x_{n+1}|y_n, ..., y_1) = \int_{\Omega_x} p(x_{n+1}|x_n)p(x_n|y_n, ..., y_1)dx_n, \tag{2}$$

where Ω_x is the whole space of the hidden state. This prediction can be updated once we observe the y_{n+1}

$$p(x_{n+1}|y_{n+1}, y_n, ..., y_1) = \frac{p(y_{n+1}|x_{n+1})p(x_{n+1}|y_n, ..., y_1)}{p(y_{n+1}|y_n, ..., y_1)}, \tag{3}$$

where $p(x_{n+1}|y_n, ..., y_1)$ plays the role of prior distribution and the emission density $p(y_{n+1}|x_{n+1})$ is the likelihood that update our estimate of the hidden state at time step $n + 1$. In the remaining of this section, we explain the one-dimensional SSM used in the report to visualize the SSM and the filtered results.

1.1 Stochastic Volatility Model

The Stochastic Volatility Model is widely used in financial econometrics to model time-varying variance in asset returns. In the codebase, this is implemented via the function `get1DStochasticVolModel`.

Let x_t represent the latent log-volatility and y_t represent the observed asset return (or mean-adjusted return).

The log-volatility follows an autoregressive AR(1) process:

$$x_{t+1} = \alpha x_t + \eta_t, \quad \eta_t \sim \mathcal{D}(0, \sigma^2) \tag{4}$$

where x_t is the latent state (log-volatility), α is the persistence parameter ($|\alpha| < 1$ for stationarity), σ is the volatility of volatility (vol-of-vol) and $\mathcal{D}$ is the process noise distribution. The code supports both Gaussian $\mathcal{N}(0, \sigma^2)$ and heavy-tailed Student-t distributions.

The observations depend non-linearly on the state via an exponential function:

$$y_t = \beta \exp\left(\frac{x_t}{2}\right) \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, 1) \tag{5}$$

where β is the scaling factor for the observation noise (related to baseline volatility) and ϵ_t is the observation noise, which is assumed to be standard normal. The term $\exp(x_t/2)$ represents the instantaneous standard deviation derived from the log-variance x_t.

1.2 Log-Squared Stochastic Volatility Model

To utilize linear filtering techniques (like the Kalman Filter) or to simplify the observation density, the SVM is often transformed by squaring and taking the logarithm of the observations. This is implemented in `get1DLogSquaredSVM` and approximated in `get_LogSVM_LGSSM`.

Starting from the observation equation $y_t = \beta \exp(x_t/2)\epsilon_t$, we square both sides:

$$y_t^2 = \beta^2 \exp(x_t)\epsilon_t^2 \tag{6}$$

Taking the logarithm yields a linear observation equation with non-Gaussian noise:

$$\log(y_t^2) = x_t + \log(\beta^2) + \log(\epsilon_t^2) \tag{7}$$

where $\log(\epsilon_t^2)$ follows a log-χ_1^2 distribution (logarithm of a Chi-square variable with 1 degree of freedom), with Mean: $\mathrm{E}[\log(\epsilon_t^2)] = \psi(0.5) - \log(0.5) \approx -1.27$, where ψ is the digamma function and textbfVariance: $\mathrm{Var}(\log(\epsilon_t^2)) = \frac{\pi^2}{2} \approx 4.93$. In the codebase, the log-χ_1^2 distribution is approximated by a normal distribution with the same mean and variance for the filtering algorithms explicitly requiring an additive Gaussian noise in the model.

The code constructs an LGSSM approximation by treating the non-Gaussian noise as Gaussian with the moments derived above.

$$x_{t+1} = \alpha x_t + \eta_t, \quad \eta_t \sim \mathcal{N}(0, \sigma^2) \tag{8}$$

$$z_t = x_t + \mu_{\text{bias}} + v_t, \quad v_t \sim \mathcal{N}(0, \frac{\pi^2}{2}) \tag{9}$$

where $z_t = \log(y_t^2)$ and the bias term is $\mu_{\text{bias}} = \log(\beta^2) + \mathbb{E}[\log(\epsilon_t^2)]$. This allows the use of a standard Kalman Filter as a robust baseline.

1.3 Vasicek Bond Pricing Model

This model combines the Vasicek short-rate process with the affine term structure of interest rates to price zero-coupon bonds. It is implemented in `getVasicekBondPriceModel`.

The instantaneous short rate r_t follows a mean-reverting Ornstein-Uhlenbeck SDE:

$$dr_t = \kappa(\theta - r_t)dt + \sigma dW_t \tag{10}$$

where κ is the mean reversion rate, θ is the long-term mean level of the interest rate, and σ is the volatility of the short rate. To implement this model numerically, the code defines the latent state as the centered rate $x_t = r_t - \theta$. The exact discretization over time step Δt is:

$$x_{t+1} = e^{-\kappa \Delta t} x_t + \eta_t, \quad \eta_t \sim \mathcal{N}\left(0, \frac{\sigma^2}{2\kappa}(1 - e^{-2\kappa \Delta t})\right) \tag{11}$$

For small Δt, the code approximates the variance as $\sigma^2 \Delta t$.

The observation y_t is the price of a Zero-Coupon Bond with time to maturity τ. Under the Vasicek model, this price is given by the affine formula:

$$P(r_t, \tau) = A(\tau) e^{-B(\tau) r_t} \tag{12}$$

Substituting $r_t = x_t + \theta$:

$$y_t = P(x_t + \theta, \tau) + \nu_t = A(\tau) e^{-B(\tau)(x_t + \theta)} + \nu_t \tag{13}$$

The coefficients are:

$$B(\tau) = \frac{1 - e^{-\kappa \tau}}{\kappa} \tag{14}$$

$$A(\tau) = \exp\left(\left(\theta - \frac{\sigma^2}{2\kappa^2}\right)(B(\tau) - \tau) - \frac{\sigma^2}{4\kappa} B(\tau)^2\right) \tag{15}$$

This constitutes a non-linear observation function $h(x_t)$ suited for EKF/UKF testing.

2 Classical Filters

Classical filters enable analytical solutions for state estimation but require linear transition and observation models, as well as Gaussian process and observation noise. We define a time-invariant Linear Gaussian State Space Model (LGSSM) as

$$\begin{aligned} x_t &= A x_{t-1} + q_t, & q_t &\sim \mathcal{N}(0, Q) \\ y_t &= C x_t + r_t, & r_t &\sim \mathcal{N}(0, R) \end{aligned} \tag{16}$$

where $x_t \in \mathbb{R}^{D_x}$ is the latent state, $y_t \in \mathbb{R}^{D_y}$ is the observation, A is the state transition matrix, and C is the observation matrix. The initial state is distributed as $x_0 \sim \mathcal{N}(\mu_0, P_0)$.

2.1 The Kalman Filter

The Kalman Filter performs exact Bayesian inference to estimate the posterior distribution $p(x_t | y_{1:t})$. In the codebase, this is implemented within the `KalmanFilter` class. Given the posterior at time

$t-1$, denoted by $x_{t-1|t-1} \sim \mathcal{N}(\hat{x}_{t-1|t-1}, P_{t-1|t-1})$, the predictive distribution is derived by taking expectations of the state transition equation

$$\hat{x}_{t|t-1} = A\hat{x}_{t-1|t-1} \tag{17}$$

$$P_{t|t-1} = AP_{t-1|t-1}A^\top + Q \tag{18}$$

Upon observing y_t, the estimate is corrected by computing the innovation $\nu_t = y_t - C\hat{x}_{t|t-1}$ and the innovation covariance $S_t = CP_{t|t-1}C^\top + R$. The optimal Kalman Gain K_t, derived by minimizing the mean squared error, is given by $K_t = P_{t|t-1}C^\top S_t^{-1}$. Consequently, the state estimate and covariance are updated as

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t\nu_t \tag{19}$$

The standard covariance update $P_{t|t} = (I - K_tC)P_{t|t-1}$ is prone to numerical errors, potentially resulting in non-symmetric or indefinite matrices. To ensure positive semi-definiteness, the implementation utilizes the Joseph Form, given by $P_{t|t} = (I - K_tC)P_{t|t-1}(I - K_tC)^\top + K_tRK_t^\top$, whose effect is shown in Fig. 3, where the Kalman filter is tested for a linear Gaussian SSM with ten state and observation dimensions. It can be found in the checkKF.py file in the codebase. This formulation arises from expanding the error covariance definition $\mathbb{E}[(x_t - \hat{x}_{t|t})(x_t - \hat{x}_{t|t})^\top]$.

2.2 Rauch-Tung-Striebel (RTS) Smoother

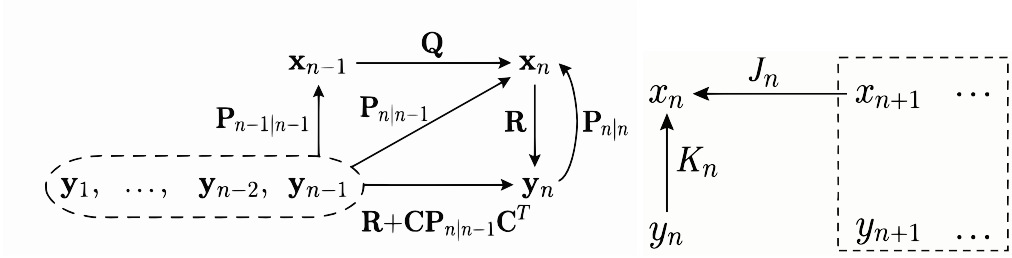
The RTS smoother computes the full posterior marginals $p(x_t|y_{1:T})$ utilizing the entire observation sequence. It operates as a backward pass starting from $t = T - 1$ down to 0. Using the Markov property of the SSM, the smoothing gain J_t is derived as $J_t = P_{t|t}A^\top P_{t+1|t}^{-1}$. The smoothed mean and covariance are then propagated backward according to

$$\hat{x}_{t|T} = \hat{x}_{t|t} + J_t(\hat{x}_{t+1|T} - \hat{x}_{t+1|t}) \tag{20}$$

$$P_{t|T} = P_{t|t} + J_t(P_{t+1|T} - P_{t+1|t})J_t^\top \tag{21}$$

Additionally, the cross-covariance $P_{t,t-1|T}$, which is required for parameter learning via EM, is approximated by $P_{t,t-1|T} \approx P_{t|T}J_{t-1}^\top$.

The schematic figures for filter and smoother can be found in Fig. 2.



(a) (b)

Figure 2: Schematic Figure for the relations between covariance matrices (a) and information flow in filter and smoother (b).

2.3 EM Algorithm for LGSSM

To estimate the model parameters $\theta = \{A, C, Q, R, \mu_0, P_0\}$, we maximize the log-likelihood $\mathcal{L}(\theta) = \log p(y_{1:T}|\theta)$. The Expectation-Maximization (EM) algorithm maximizes the expected complete log-likelihood $\mathcal{Q}(\theta, \theta^{old}) = \mathbb{E}_{X|Y}[\log p(x_{0:T}, y_{1:T}|\theta)]$. By taking derivatives of $\mathcal{Q}$ with respect to the parameters and setting them to zero, we obtain analytic solutions implemented in `EM_solver`. Letting the sufficient statistics be $\Sigma_{xx} = \sum_{t=0}^{T-1}\mathbb{E}[x_tx_t^\top]$, $\Sigma_{x1x1} = \sum_{t=1}^{T}\mathbb{E}[x_tx_t^\top]$,

$\Sigma_{xx1} = \sum_{t=1}^{T} \mathbb{E}[x_t x_{t-1}^\top]$, and $\Gamma_{yx} = \sum_{t=0}^{T-1} y_t \mathbb{E}[x_t]^\top$, the update rules for the M-step are

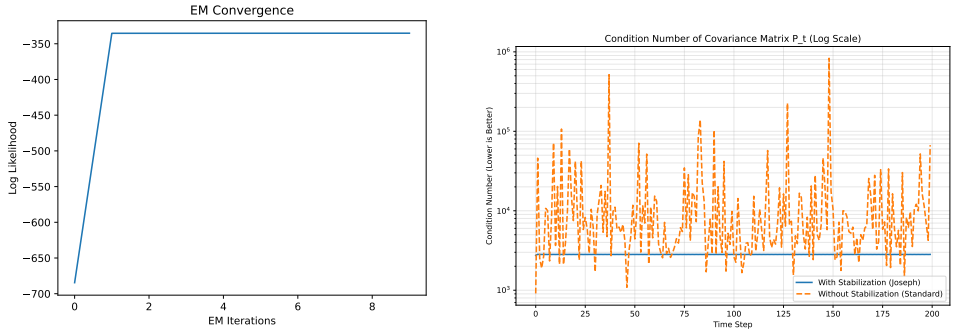
$$A_{new} = \Sigma_{xx1}\Sigma_{xx}^{-1} \tag{22}$$

$$C_{new} = \Gamma_{yx}\Sigma_{xx}^{-1} \tag{23}$$

$$Q_{new} = \frac{1}{T-1}\left(\Sigma_{x1x1} - A_{new}\Sigma_{xx1}^\top - \Sigma_{xx1}A_{new}^\top + A_{new}\Sigma_{xx}A_{new}^\top\right) \tag{24}$$

$$R_{new} = \frac{1}{T}\left(\sum y_t y_t^\top - C_{new}\Gamma_{yx}^\top - \Gamma_{yx}C_{new}^\top + C_{new}\Sigma_{xx}C_{new}^\top\right) \tag{25}$$

Fig. 3 shows an example of the convergence of the EM algorithm, where the model is a linear Gaussian state-space model with three state and observation dimensions, and the model parameters are drawn from independent normal distributions



(a) (b)

Figure 3: (a) EM algorithm convergence exmaple for Kalman filter. (b) Effect of the Joseph stabilization for the posterior covariance update

2.4 Extended Kalman Filter (EKF)

For non-linear systems, the linearity assumption of the Kalman Filter does not hold. The EKF addresses this by linearizing the dynamics around the current estimate. This is implemented in the `ExtendedKalmanFilter` class. The system is described by non-linear transition and observation functions

$$\begin{aligned} x_t &= f(x_{t-1}, q_t) \\ y_t &= h(x_t, r_t) \end{aligned} \tag{26}$$

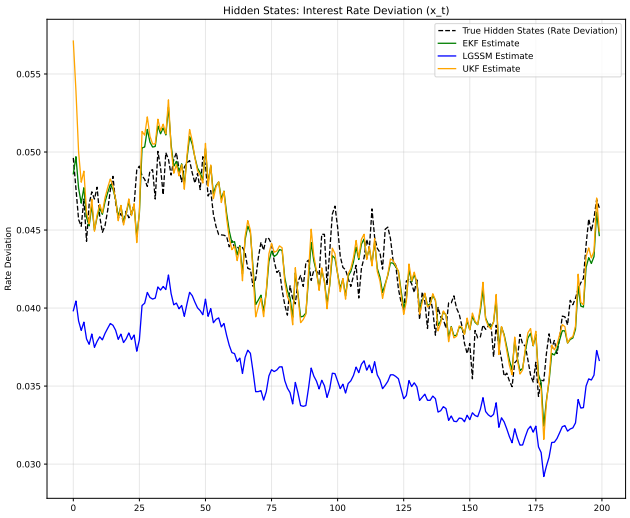
The EKF approximates these functions via a first-order Taylor expansion. The implementation utilizes Automatic Differentiation to compute the Jacobians $F_t = \frac{\partial f}{\partial x}\big|_{\hat{x}_{t-1|t-1}}$ and $H_t = \frac{\partial h}{\partial x}\big|_{\hat{x}_{t|t-1}}$. The standard Kalman filter prediction and update equations are then applied, substituting F_t for A and H_t for C.

Since the exact posterior is intractable in non-linear models, the implementation uses an approximate EM approach. In the E-Step, we run the EKF Smoother (linearized RTS) to obtain approximations for $\hat{x}_{t|T}$ and $P_{t|T}$. In the M-Step, we update Q and R based on the residuals of the non-linear functions. For example, the observation noise covariance is updated as

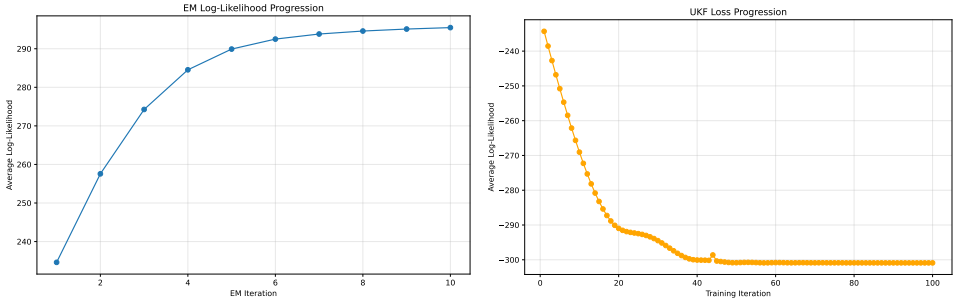
$$R_{new} \approx \frac{1}{T}\sum_t \left((y_t - h(\hat{x}_{t|T}))(y_t - h(\hat{x}_{t|T}))^\top + H_t P_{t|T} H_t^\top\right) \tag{27}$$

2.5 Unscented Kalman Filter (UKF)

The EKF can perform poorly when non-linearities are severe. The Unscented Kalman Filter (UKF) captures higher-order moments by propagating a set of deterministic Sigma Points through the non-linear functions. This is implemented in `UnscentedKalmanFilter`. For an n-dimensional state



(a)



(b) (c)

Figure 4: (a) Comparison between Kalman filter, Extend Kalman filter (EKF) and Unscent Kalman filter (UKF) on the Vasicek model. (b) and (c) example for the convergence of EM algorithm for EKF and gradient descent for UKF.

with mean $\bar{x}$ and covariance P, we generate $2n+1$ sigma points. Defining a scaling parameter $\lambda = \alpha^2(n+\kappa) - n$, the points are given by

$$\begin{aligned} \mathcal{X}_0 &= \bar{x} \\ \mathcal{X}_i &= \bar{x} + \left(\sqrt{(n+\lambda)P}\right)_i, \quad i = 1 \dots n \\ \mathcal{X}_{i+n} &= \bar{x} - \left(\sqrt{(n+\lambda)P}\right)_i, \quad i = 1 \dots n \end{aligned} \tag{28}$$

The weights for recovering the mean (W_m) and covariance (W_c) are defined such that $W_m^{(0)} = \frac{\lambda}{n+\lambda}$ and $W_c^{(0)} = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta)$, while for $i > 0$, $W_m^{(i)} = W_c^{(i)} = \frac{1}{2(n+\lambda)}$. Here β incorporates prior knowledge of the distribution.

In the prediction step, the sigma points are propagated through the transition function $f(\cdot)$ to yield $\mathcal{Y}_i = f(\mathcal{X}_i)$, and the predicted mean is $\hat{x}_{pred} = \sum W_m^{(i)} \mathcal{Y}_i$. In the update step, these points are mapped to observation space via $h(\cdot)$, yielding $\mathcal{Z}_i = h(\mathcal{Y}_i)$ and $\hat{y}_{pred} = \sum W_m^{(i)} \mathcal{Z}_i$. The Kalman Gain is computed utilizing the cross-covariance P_{xy} and innovation covariance S as $K = P_{xy} S^{-1}$.

Unlike the EM algorithm used for LGSSM, the UKF implementation employs direct gradient descent to optimize parameters, specifically the log-scales of process and observation noise. The objective function is the negative log-likelihood of the observations $\mathcal{L} = -\sum_{t=1}^{T} \log p(y_t | y_{1:t-1})$, where

$p(y_t|y_{1:t-1}) \approx \mathcal{N}(y_t; \hat{y}_{t|t-1}, S_t)$. This approach facilitates learning stationary parameters in highly non-linear settings where analytic M-steps are intractable.

The comparison between KF, EKF and UKF can be found in Fig. 4 and the `nonlinearSSM.py` in the codebase.

3 Particle Filtering and Flow-Based Methods

Nonlinear filtering aims to estimate the posterior distribution $p(x_k|y_{1:k})$ of a state x_k given observations $y_{1:k}$. Standard sequential Monte Carlo (SMC) methods often suffer from weight degeneracy in high-dimensional spaces or when measurements are highly informative. In such cases, the proposal distribution, which is often chosen to be the prior, has little overlap with the likelihood, causing most particle weights to collapse to zero. To address this, the implemented algorithms utilize Particle Flow, which migrates particles from the prior to the posterior via a pseudo-time process before the weight update step.

3.1 Standard Particle Filter (Bootstrap)

The Bootstrap Particle Filter (BPF) serves as the standard baseline implementation in `particle_filter.py`. It utilizes the system transition density as the proposal distribution, given by $q(x_k|x_{k-1}, y_k) = p(x_k|x_{k-1})$. Given a set of particles $\{x^i_{k-1}, w^i_{k-1}\}^N_{i=1}$ approximating $p(x_{k-1}|y_{1:k-1})$, the algorithm proceeds by first propagating samples $x^i_k \sim p(x_k|x^i_{k-1})$. Subsequently, the unnormalized importance weights are updated via the likelihood function according to

$$\tilde{w}^i_k \propto w^i_{k-1} \frac{p(y_k|x^i_k)p(x^i_k|x^i_{k-1})}{q(x^i_k|x^i_{k-1}, y_k)} = w^i_{k-1} p(y_k|x^i_k) \tag{29}$$

To mitigate degeneracy, the particles are resampled using Multinomial or Systematic resampling when the Effective Sample Size (ESS) drops below a defined threshold. The example for particle filter and its weight degeneracy problem for tasks with high measurement precision can be found in Fig. 5 and Fig. 6.

3.2 Exact Daum-Huang (EDH) and Local EDH Flows

The Exact Daum-Huang (EDH) Particle Flow, implemented in `particle_flow.py`, migrates particles along a pseudo-time $\lambda \in [0, 1]$ according to a stochastic differential equation $d\eta_\lambda = \zeta(\eta_\lambda, \lambda)d\lambda$, where η_0 follows the predictive prior and η_1 approximates the posterior. The drift ζ is derived to satisfy the Fokker-Planck equation such that the distribution of η_λ matches a log-homotopy between the prior and the posterior densities. Assuming a linear-Gaussian approximation, the EDH filter derives an analytic solution for the drift $\zeta(\eta, \lambda) = A(\lambda)\eta + b(\lambda)$. The flow parameters are computed as

$$A(\lambda) = -\frac{1}{2} PH(\lambda)^T (\lambda H(\lambda) PH(\lambda)^T + R)^{-1} H(\lambda) \tag{30}$$

and

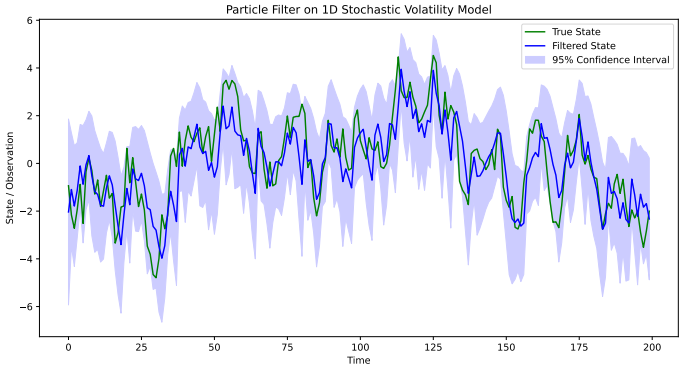
$$b(\lambda) = (I + 2\lambda A(\lambda)) \left[(I + \lambda A(\lambda)) PH(\lambda)^T R^{-1} (y_k - e(\lambda)) + A(\lambda)\bar{\eta}_0 \right] \tag{31}$$

where P is the predictive covariance estimated via EKF or UKF, R is the measurement noise covariance, $H(\lambda)$ is the Jacobian of the measurement function linearized at the mean particle $\bar{\eta}_\lambda$, and $e(\lambda)$ is the linearization error correction.

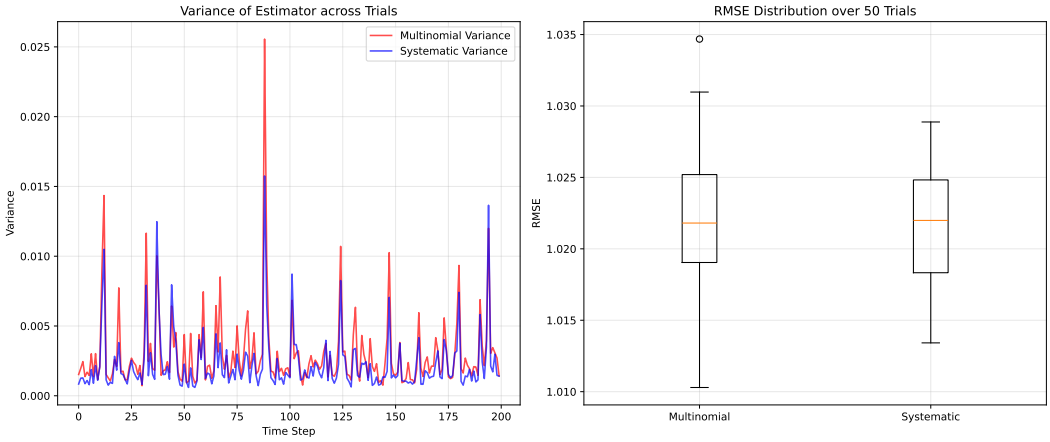
The standard EDH flow linearizes the measurement model at the global mean, which may be insufficient for highly non-linear functions. The Local Exact Daum-Huang (LEDH) flow addresses this by linearizing the model locally for each particle. Consequently, the drift parameters $A^i(\lambda)$ and $b^i(\lambda)$ become particle-specific, computed using the Jacobian H^i evaluated at each particle's location. While this significantly increases computational complexity, it provides a more accurate flow in non-linear manifolds.

3.3 Invertible Particle Flow Particle Filter (PF-PF)

While EDH and LEDH migrate particles to high-likelihood regions, accumulated discretization errors and Gaussian approximations mean the particles may not exactly sample the true posterior. The PF-PF



(a)



(b)

Figure 5: (a) Particle filter for 1D stochastic volatility model. (b) Comparison between multinomial resampling and systematic resampling.

framework treats the flow as a sophisticated proposal distribution q within a standard Importance Sampling framework. Computing the weights requires evaluating the proposal density $q(\eta_1^i)$, which in turn implies the flow mapping T must be invertible. The flow is discretized into steps of size ϵ_j, creating a composition of affine maps $\eta_{\lambda_j}^i = (I + \epsilon_j A_j^i)\eta_{\lambda_{j-1}}^i + \epsilon_j b_j^i$. For sufficiently small ϵ_j, the matrix $(I + \epsilon_j A_j^i)$ is full rank, ensuring invertibility. The Jacobian determinant of the total flow is given by $|\det \dot{T}(\eta_0^i)| = \prod_{j=1}^{N_\lambda} |\det(I + \epsilon_j A_j^i)|$. Utilizing the change of variables formula, the final weight update rule becomes

$$w_k^i \propto w_{k-1}^i \frac{p(y_k|\eta_1^i)p(\eta_1^i|x_{k-1}^i)|\det \dot{T}(\eta_0^i)|}{p(\eta_0^i|x_{k-1}^i)} \tag{32}$$

where $p(y_k|\eta_1^i)$ is the likelihood at the migrated position, $p(\eta_1^i|x_{k-1}^i)$ is the transition density of the migrated particle, and $p(\eta_0^i|x_{k-1}^i)$ is the transition density of the particle prior to flow.

The comparison of the EDH, LEDH and PF-PF, etc. algorithms is shown in the following table, which aims at replicating the experiment B in .

3.4 Kernel Particle Flow Filter

An alternative approach to deriving the flow is to minimize the rate of change of the Kullback-Leibler (KL) divergence between the particle density q_s and the target log-homotopy density p_s. The Kernel Particle Flow Filter (KPFF) formulates this optimization within a vector-valued Reproducing Kernel

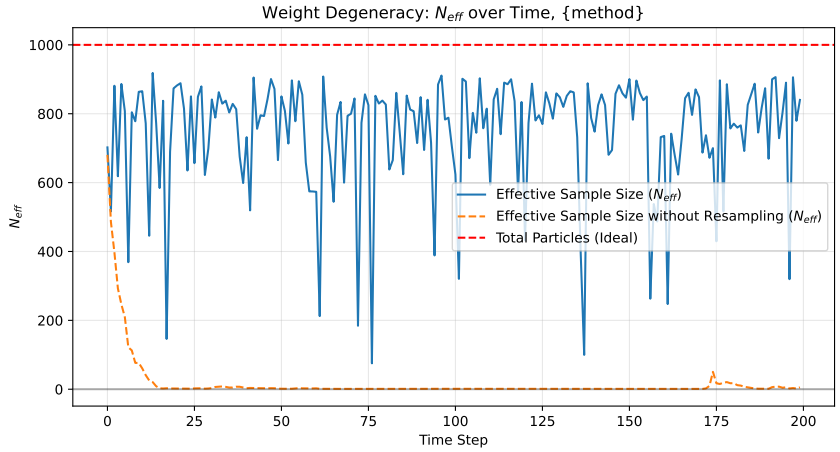


Figure 6: Weight degeneracy problem of particle filter without resampling

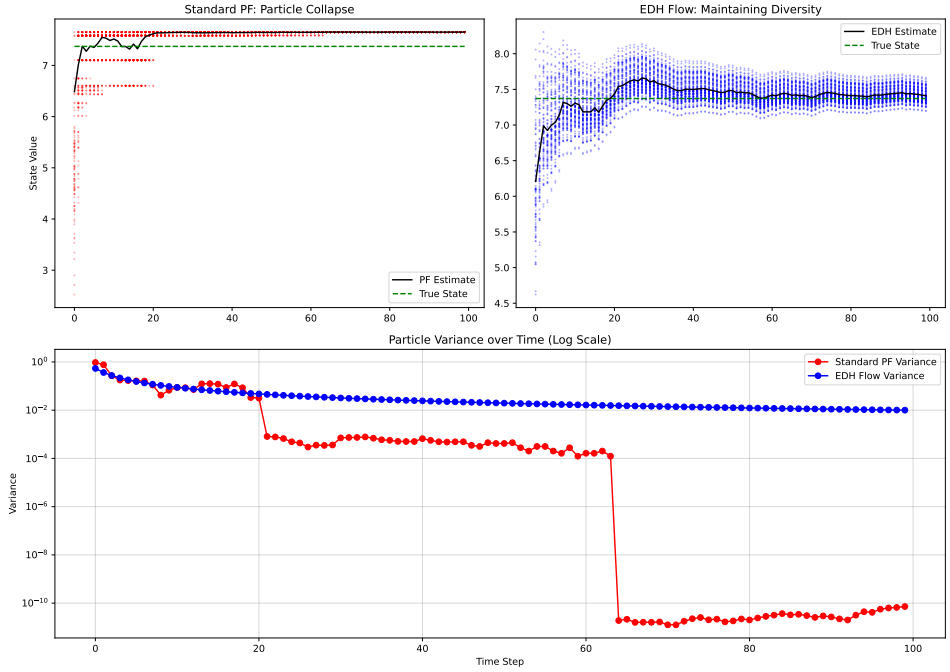


Figure 7: Sample Impoverishment problem for particle filter with resampling

Hilbert Space (RKHS), yielding the solution

$$f_s(x) = \mathbb{E}_{x' \sim q_s} \left[K(x, x') \nabla_{x'} \log p_s(x') + \nabla_{x'} \cdot K(x, x') \right] \tag{33}$$

This drift has a clear physical interpretation. The first term, involving the score function $\nabla \log p_s$, acts as an attraction force pulling particles toward high-probability modes, smoothed by the kernel K. The second term, the divergence of the kernel, acts as a repulsive force that prevents particle collapse and maintains the distribution's spread.

The choice of kernel $K(x, x')$ is critical. A scalar-valued kernel, $K(x, x') = k(x, x')\mathbf{I}$, imposes an isotropic geometry, which is often inefficient for high-dimensional states where the probability mass concentrates on thin manifolds. A matrix-valued kernel $K(x, x') = [K_1(x_1, x_1'), ..., K_D(x_D, x_D')]$ incorporates geometric information, typically using the covariance matrix $\mathbf{B}$ as a preconditioner and enable spatial scales to be different along different directions. This allows the flow to account for correlations and varying scales of state variables, significantly improving efficiency in high-

Table 1: Algorithm Performance under Different Noise Levels

Algorithm	Particle num.	$\sigma_z = 2$		$\sigma_z = 1$		$\sigma_z = 0.5$	
		Avg. MSE	Avg. ESS	Avg. MSE	Avg. ESS	Avg. MSE	Avg. ESS
PF-PF (LEDH)	200	0.6113	27.98	0.2507	23.25	0.1049	17.40
PF-PF (EDH)	200	0.6024	28.42	0.2539	23.08	0.1060	17.15
PF-PF (EDH)	10^4	0.5224	1035.44	0.2162	904.76	0.0924	698.08
EDH	200	0.4917	N/A	0.1847	N/A	0.0717	N/A
KF	N/A	0.4924	N/A	0.1843	N/A	0.0714	N/A
UKF	N/A	0.4924	N/A	0.1843	N/A	0.0714	N/A
BPF	200	1.4649	1.89	1.3894	1.18	1.3613	1.04
BPF	10^5	0.5672	38.72	0.3929	3.05	0.3632	1.34

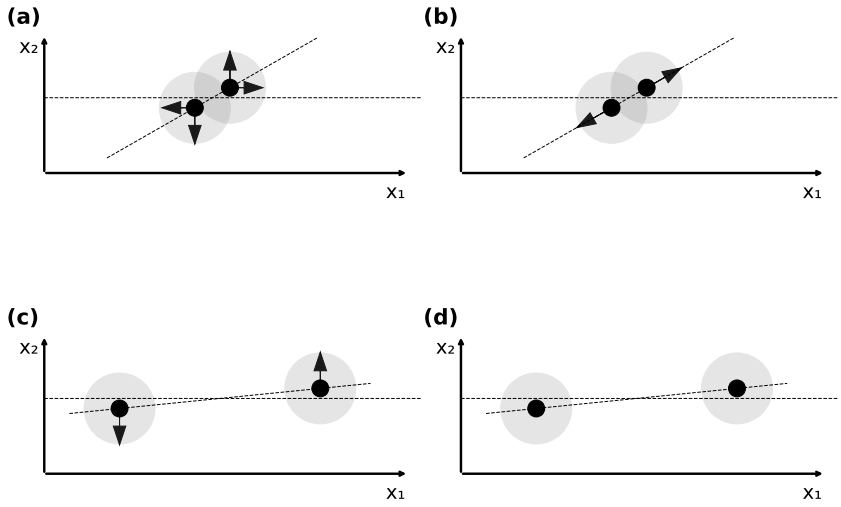


Figure 8: Comparison between scalar and matrix kernel

dimensional applications, where scalar kernel can give abnormally small repulsive force even if the particles are already close in some dimensions [Fig. 8], and thus lead to particle collapsing problem, as shown in Fig. 9.

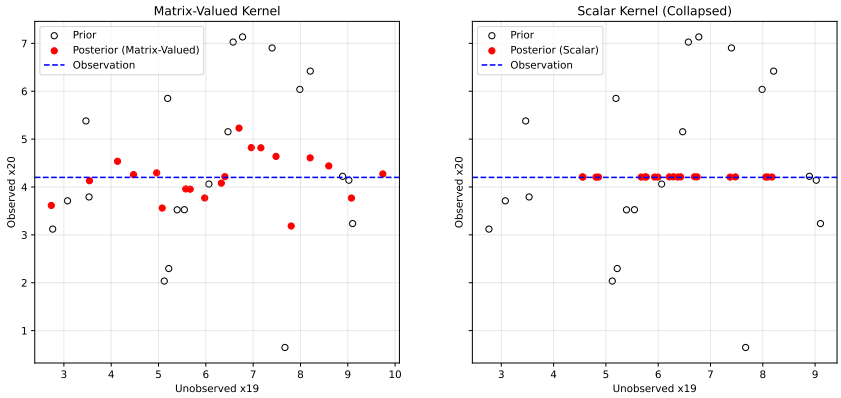


Figure 9: Comparison between the posterior given by matrix kernel and scalar kernel