Weeks 8 & 9 - Applying Grid-Based Filters to the Neuroscience Model

Max Richards

2025-09-16

Summary of Weeks 6 and 7

Over the past few weeks, I was away and therefore had limited direct involvement with the project. Prior to this period of absence, however, I dedicated time to exploring the neuroscience model, gaining valuable insight through discussions with Rocco, and conducting preliminary experiments with the model across multiple dimensions. This initial stage of work allowed me to develop a clearer understanding of the model's structure and potential applications within the broader scope of the project.

During this period, I also made an attempt to implement the ORCSMC algorithm as described in Adam's paper. This proved to be a highly challenging and technically demanding task, given the inherent complexity of the algorithm. While this provided me with useful exposure to the algorithmic framework and its underlying mechanics, I later came to realise that there had been a degree of misalignment in my focus. Specifically, my primary objective within this project is not to replicate or fully develop the ORCSMC algorithm, but rather to investigate the role and effectiveness of numerical, grid-based filters in this setting.

My efforts in the upcoming weeks will therefore be directed toward a systematic exploration of grid-based filtering methods. I aim to evaluate their performance and effectiveness, as well as to conduct comparisons with alternative filtering approaches. This work will be carried out across a range of different state-space models, ensuring that my assessment is not confined to a single example but instead reflects a variety of scenarios. In doing so, I will also revisit and build upon the work I have undertaken over the past eight weeks, consolidating earlier results into a more comprehensive framework.

Ultimately, the goal of this phase of the project is to draw well-supported conclusions regarding the practicality, strengths, and limitations of numerical grid-based filters across diverse settings. This analysis should contribute both to a deeper understanding of their theoretical properties and to an informed perspective on their potential applications in practice.

Recap of the Neuroscience Model (1D)

The model is was discussed in my most recent notes of work, and this is the nuroscience model estbalished in the ORCSMC paper. I will sumarrise the mathematical intuition once again below though, for ease of reading.

Model Specification

Let $\{x_t\}_{t\geq 0}$ denote the latent state process and $\{y_t\}_{t\geq 1}$ the corresponding observations.

In the univariate setting, each observation $y_t \in \{0, 1, \ldots, M\}$ represents the number of activated neurons observed in M repeated trials at time t, with $M = 50$. The observation model is defined as a Binomial distribution with a logistic link function:

$$y_t \mid x_t \sim \text{Binomial}(M, \kappa(x_t)), \tag{1}$$

$$\kappa(x_t) = \frac{1}{1 + \exp(-x_t)}, \tag{2}$$

where $\kappa(x_t)$ maps the latent state to the probability of neuronal activation.

The latent dynamics are specified as a linear Gaussian autoregressive process:

$$x_0 \sim \mathcal{N}(0, 1), \tag{3}$$

$$x_t \mid x_{t-1} \sim \mathcal{N}(\alpha x_{t-1}, \sigma^2), \tag{4}$$

where $\alpha = 0.99$ and $\sigma^2 = 0.11$. This formulation captures the temporal dependence of neuronal activity over consecutive trials.

Simulation and Filtering of a Binomial State Space Model

In this section, we describe the simulation of a simple state space model and the implementation of a uniform grid-based approach, and an adaptive grid-based filter, which adjusts the grid by centering and truncating around a constantly evolving posterior's mean and variance.

Simulating the State Space Model

We are interested in simulating a simple state space model consisting of a latent AR(1) process observed through a binomial likelihood with a logistic link. Specifically, the latent state evolves according to:

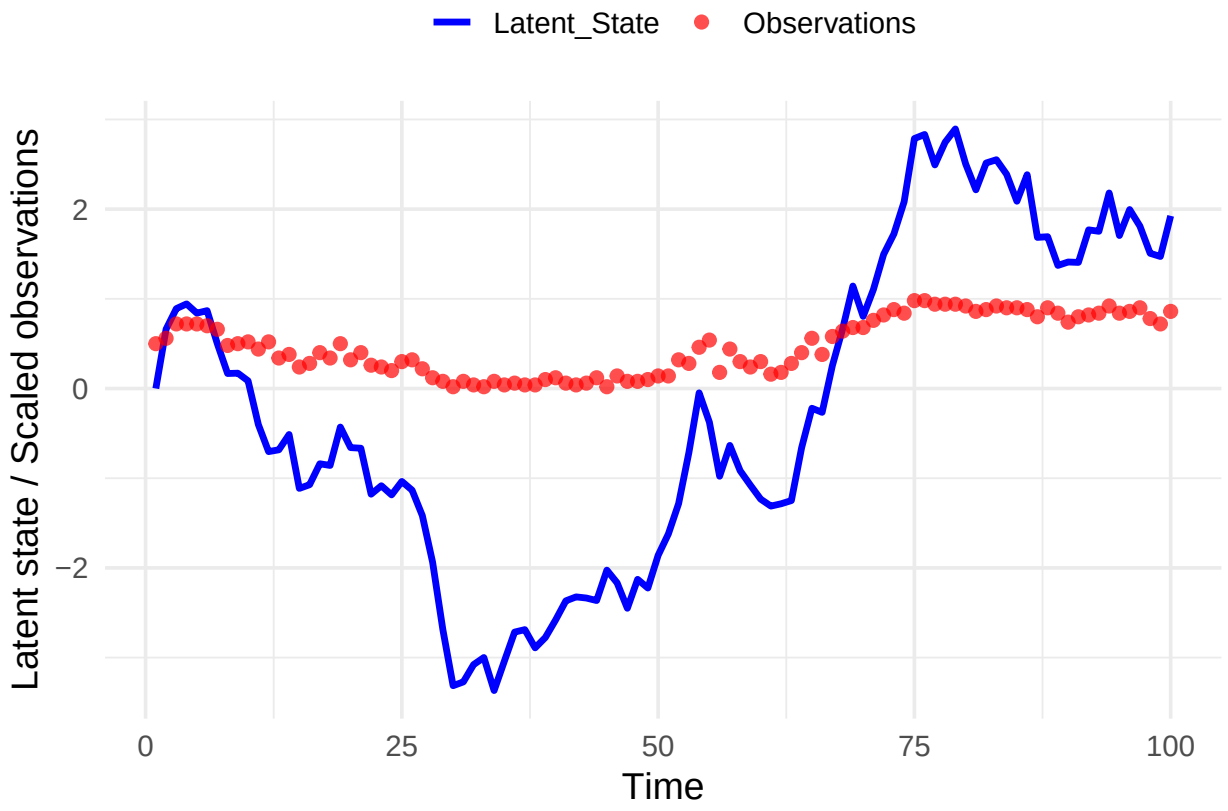
$$x_t = \alpha x_{t-1} + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \sigma^2),$$

where we choose $\alpha = 0.99$ and $\sigma^2 = 0.11$. This defines a smooth autoregressive process for the latent state x_t.

The observations are generated as binomial counts with success probability given by a logistic transformation of the latent state:

$$y_t \sim \text{Binomial}(M, p_t), \quad p_t = \frac{1}{1 + e^{-x_t}}.$$

Here, M is the number of trials at each time step, and p_t maps the real-valued latent state to the interval $[0, 1]$. In R, we will simulate both x_t and y_t over a chosen time horizon and visualize their behavior.



Implementing the Uniform grid-based filter in 1D

In order to estimate the latent state process of the binomial-logistic state space model, we consider a uniform grid-based filter. The key idea is to discretise the latent state space into a fixed grid of points, and then propagate and update probability weights on this grid at each time step. This approach replaces analytic or sampling-based methods (such as particle filters) with a deterministic numerical scheme that approximates the filtering distribution.

I have worked with these filters in the past few weeks extensively (particularly week 4)

Discussion

This grid-based approach provides a deterministic approximation to the filtering distribution, avoiding the sampling variance inherent in particle filters. Its accuracy depends on the choice of grid size K and interval $[a, b]$. While computationally more expensive in higher dimensions (due to the curse of dimensionality), it is particularly suitable for low-dimensional nonlinear state space models, such as the binomial-logistic model considered here.

Grid Construction

Let $\mathcal{X} = \{x^{(1)}, x^{(2)}, \ldots, x^{(K)}\}$ denote a uniform grid of K points over an interval $[a, b] \subset \mathbb{R}$, chosen to cover the region where the latent state is likely to lie. For instance, in the logistic-binomial model, $[a, b] = [-5, 5]$ is often sufficient due to the saturation of the logistic function.

Recursive Filtering Equations

The filtering distribution can be written recursively as

$$p(x_t \mid y_{1:t}) \propto p(y_t \mid x_t)\, p(x_t \mid y_{1:t-1}), \tag{5}$$

$$p(x_t \mid y_{1:t-1}) = \int p(x_t \mid x_{t-1})\, p(x_{t-1} \mid y_{1:t-1})\, dx_{t-1}. \tag{6}$$

Prediction step. On the grid, the predictive distribution~(6) is approximated numerically by

$$\pi_t^{\text{pred}}(x^{(i)}) = \sum_{j=1}^{K} p(x^{(i)} \mid x^{(j)})\, \pi_{t-1}(x^{(j)})\, \Delta x, \tag{7}$$

where $\pi_{t-1}(x^{(j)})$ denotes the posterior weight at grid point $x^{(j)}$ at time $t-1$, and Δx is the uniform grid spacing. The transition kernel is given by

$$p(x_t \mid x_{t-1}) = \mathcal{N}(x_t \mid \alpha x_{t-1}, \sigma^2).$$

Update step. The posterior distribution is obtained by weighting the predictive distribution with the likelihood of the new observation:

$$\pi_t(x^{(i)}) \propto p(y_t \mid x^{(i)})\, \pi_t^{\text{pred}}(x^{(i)}), \tag{8}$$

$$p(y_t \mid x^{(i)}) = \text{Binomial}\Big(y_t; M, \kappa(x^{(i)})\Big), \qquad \kappa(x) = \frac{1}{1+e^{-x}}. \tag{9}$$

The weights $\pi_t(x^{(i)})$ are normalised such that $\sum_{i=1}^{K} \pi_t(x^{(i)}) = 1$.

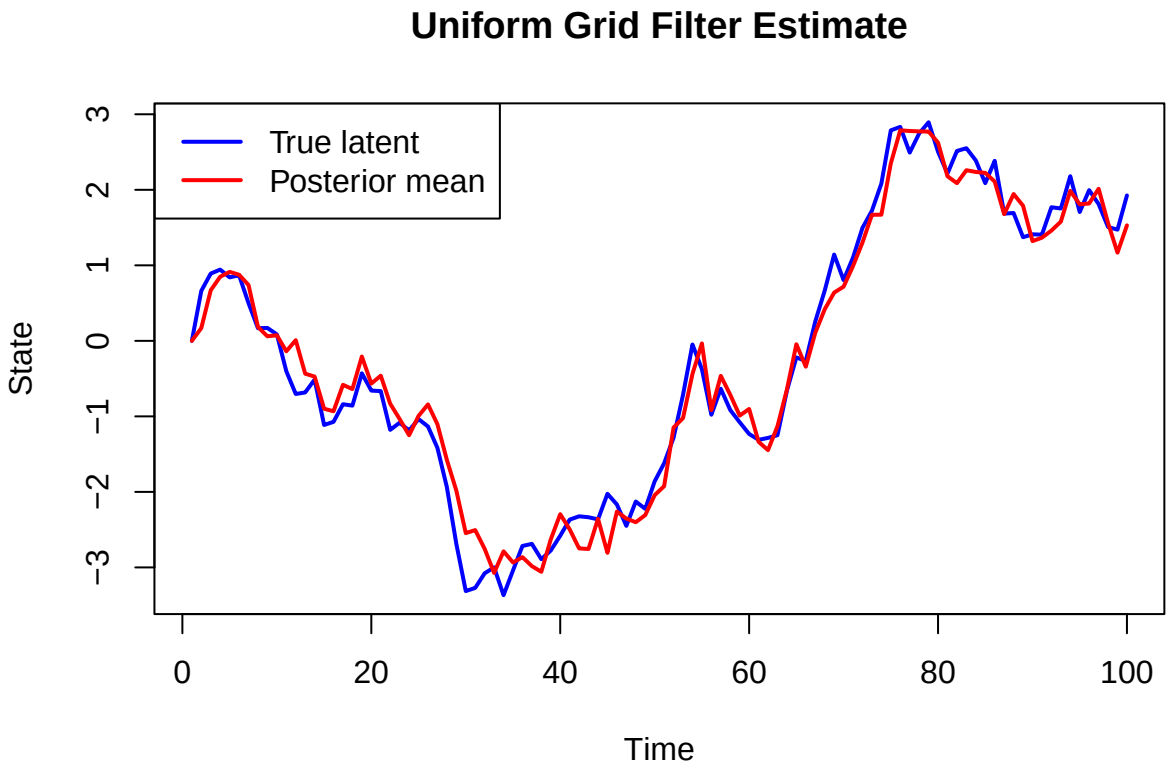
State Estimation

Once the posterior weights $\{\pi_t(x^{(i)})\}_{i=1}^K$ are obtained, filtered estimates of the latent state can be computed. For example, the posterior mean is given by

$$\hat{x}_t = \sum_{i=1}^{K} x^{(i)}\, \pi_t(x^{(i)}), \tag{10}$$

while the posterior mode (MAP estimate) is

$$\hat{x}_t^{\mathrm{MAP}} = \arg\max_{x^{(i)} \in \mathcal{X}} \pi_t(x^{(i)}). \tag{11}$$



Implementing an Adaptive Grid-Based Filter

As we saw previously the uniform grid filter described previously fixes the grid $\{x^{(i)}\}_{i=1}^K$ in advance, typically over a wide interval such as $[-L, L]$. While simple, this approach wastes resolution in regions where the posterior distribution assigns negligible probability mass. To address this, we implement an *adaptive grid-based filter*, which dynamically refocuses the grid around the current posterior at each time step.

Effectiveness of an Adaptive Grid-Based Filter

In the binomial-logistic state space model, the latent states evolve smoothly according to a Gaussian autoregressive process, while the observations are linked through a highly nonlinear logistic function. As a result, the posterior distribution at each time step typically remains concentrated around a narrow region of the state space, with probability mass rapidly decaying outside a few standard deviations of the posterior mean. A uniform fixed grid covering a large interval would therefore devote unnecessary resolution to regions of negligible posterior support, leading to computational inefficiency.

An adaptive grid-based filter mitigates this issue by re-centering and truncating the grid around the current posterior mean and variance. By allocating grid points only within a few posterior standard deviations (e.g., $\pm 6\sigma$), the method maintains high resolution where the posterior density is significant, while discarding regions that contribute negligible probability mass. This dynamic adjustment both improves numerical accuracy in representing the posterior distribution and reduces the number of grid points required, thereby increasing computational efficiency. Such an approach is particularly effective in this setting, where the logistic observation model saturates outside moderate ranges of the latent state, ensuring that informative likelihood contributions are always concentrated within a limited region of the sample space.

Posterior Representation on a Grid

At time t, the posterior distribution $\pi_t(x_t) = p(x_t \mid y_{1:t})$ is approximated on a grid of K equally spaced points $\{x_t^{(i)}\}_{i=1}^K$ with associated weights $\{w_t^{(i)}\}_{i=1}^K$. The weights are normalised so that

$$\sum_{i=1}^{K} w_t^{(i)} = 1,$$

and the posterior mean and variance can be computed as

$$\mu_t = \sum_{i=1}^{K} x_t^{(i)} w_t^{(i)}, \tag{12}$$

$$\sigma_t^2 = \sum_{i=1}^{K} (x_t^{(i)} - \mu_t)^2 w_t^{(i)}. \tag{13}$$

Adaptive Grid Construction

At each time step, the posterior mean μ_t and standard deviation σ_t are estimated. A new grid is then defined by centering it at the posterior mean and spanning $\pm c\sigma_t$, where $c > 0$ is a tuning constant (e.g. $c = 4$):

$$x_{t+1}^{(i)} = \mu_t - c\sigma_t + \frac{2c\sigma_t}{K-1}(i-1), \quad i = 1, \ldots, K.$$

This ensures the grid concentrates resolution where the posterior is non-negligible.

Propagation Step

Given the posterior weights $\{w_t^{(i)}\}$ on the current grid $\{x_t^{(i)}\}$, the predictive distribution at the next time step is approximated as

$$p(x_{t+1} \mid y_{1:t}) \approx \sum_{i=1}^{K} w_t^{(i)} \, \mathcal{N}\big(x_{t+1}; \alpha x_t^{(i)}, \sigma^2\big).$$

Since the grid changes at each step, this predictive density must be interpolated or projected onto the new adaptive grid $\{x_{t+1}^{(i)}\}$.

Update Step

The likelihood of the new observation y_{t+1} is evaluated on the adaptive grid:

$$\ell_{t+1}^{(i)} = p(y_{t+1} \mid x_{t+1}^{(i)}) = \binom{M}{y_{t+1}} \kappa(x_{t+1}^{(i)})^{y_{t+1}} \big(1 - \kappa(x_{t+1}^{(i)})\big)^{M - y_{t+1}},$$

with logistic link $\kappa(x) = \frac{1}{1+e^{-x}}$.
The posterior weights are updated via Bayes' rule:

$$w_{t+1}^{(i)} \propto p(x_{t+1}^{(i)} \mid y_{1:t}) \, \ell_{t+1}^{(i)},$$

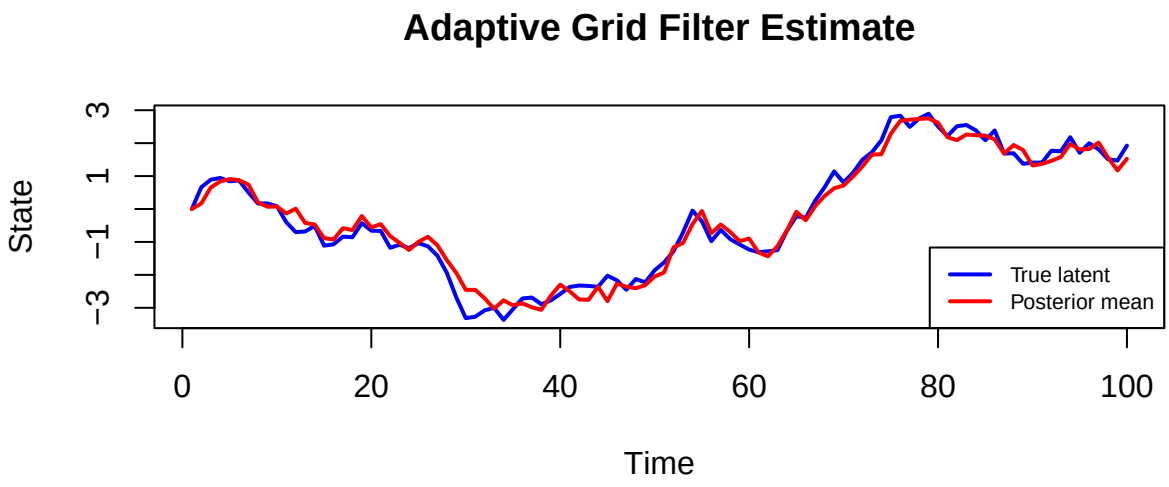
followed by normalisation.

Summary of Algorithm

The adaptive grid filter proceeds recursively as follows:

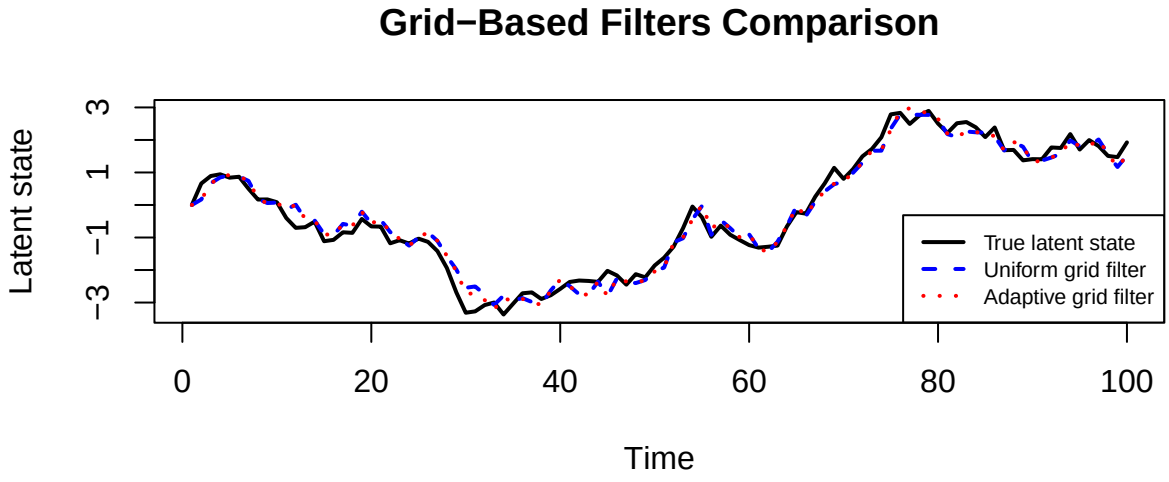
1. **Initialisation:** Choose an initial grid $\{x_0^{(i)}\}$ and weights proportional to the prior $p(x_0)$.
2. **Prediction:** Propagate the posterior $\{w_t^{(i)}\}$ through the state dynamics to approximate $p(x_{t+1} \mid y_{1:t})$.
3. **Adaptive re-centering:** Compute μ_t, σ_t^2 from $\{x_t^{(i)}, w_t^{(i)}\}$, construct new grid $\{x_{t+1}^{(i)}\}$ over $[\mu_t - c\sigma_t, \mu_t + c\sigma_t]$, and interpolate predictive density onto it.
4. **Update:** Multiply predictive weights by the likelihood $\ell_{t+1}^{(i)}$, and normalise to obtain $\{w_{t+1}^{(i)}\}$.

This method avoids wasting resolution on implausible regions of the state space, and allows accurate posterior approximation with relatively few grid points.



Comparing both approaches

Below, we present the results of both grid-based filters alongside the true latent state, and evaluate their performance using standard error metrics that we have introduced and discussed in previous weeks. This allows for a clear and quantitative comparison between the uniform and adaptive filtering approaches.



Normalised Root Mean Squared Error (NRMSE)

To quantify the point-wise accuracy of the posterior mean $\hat{x}_t$ relative to the true latent state x_t^{true}, we use the root mean squared error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(\hat{x}_t - x_t^{\text{true}})^2}.$$

Since the scale of the latent variable may vary, we normalise by the standard deviation of the true latent states to obtain the normalised RMSE:

$$\text{NRMSE} = \frac{\text{RMSE}}{\text{sd}(x^{\text{true}})}.$$

This provides a dimensionless measure of error that facilitates comparison across different models or parameter settings.

Table 1: NRMSE for Uniform and Adaptive Grid-Based Filters

Filter	NRMSE
Uniform Grid	0.1569247
Adaptive Grid	0.1521940

Computational Cost

To quantify the efficiency of each filter, we record the total wall-clock time required to run the filter over all T time steps. Let t_{filter} denote the elapsed time in seconds for a single run of the filter. We report these times as

$$\text{Cost}_{\text{uniform}} = t_{\text{uniform}}, \qquad \text{Cost}_{\text{adaptive}} = t_{\text{adaptive}}.$$

This provides a simple and transparent measure of computational efficiency, which is particularly relevant when comparing uniform versus adaptive grid-based filters with the same number of grid points.

Table 2: Computational Time for Uniform and Adaptive Grid-Based Filters

Filter	Computational Time (s)
Uniform Grid	0.6031780
Adaptive Grid	0.3533762

Stepping into multiple dimensions

Before we begin implementing any grid-based filtering approaches, we have to adjust our model specification. Very little is different, as it is the same fundamental idea, but we must make some reasonable adjustments for the higher-dimensional settings.

For higher-dimensional extensions, we consider a multivariate state $x_t = (x_{t,1}, \ldots, x_{t,d}) \in \mathbb{R}^d$ and corresponding observations $y_t = (y_{t,1}, \ldots, y_{t,d})$. The observation likelihood factorizes across dimensions:

$$y_{t,j} \mid x_{t,j} \sim \text{Binomial}(M, \kappa(x_{t,j})), \quad j = 1, \ldots, d, \tag{14}$$

while the latent dynamics remain independent across dimensions:

$$x_{t,j} \mid x_{t-1,j} \sim \mathcal{N}(\alpha x_{t-1,j}, \sigma^2). \tag{15}$$

As I am approaching the final stages of this research project, I want to go into higher dimensions, specifically beginning with 4.

4D Binomial-Logistic State Space

Mathematical setup

Below we present the multivariate extension of the univariate binomial-logistic state space model for $d = 4$. The latent state at time t is the vector

$$x_t = (x_{t,1}, x_{t,2}, x_{t,3}, x_{t,4})^\top \in \mathbb{R}^4,$$

and observations are likewise a vector of binomial counts

$$y_t = (y_{t,1}, y_{t,2}, y_{t,3}, y_{t,4})^\top, \qquad y_{t,j} \in \{0, 1, \ldots, M\}.$$

The observation model factorises across dimensions (independent observation channels) with a logistic link:

$$y_{t,j} \mid x_{t,j} \sim \text{Binomial}\big(M,\ \kappa(x_{t,j})\big), \qquad \kappa(x) = \frac{1}{1 + e^{-x}}, \quad j = 1, \ldots, 4.$$

The latent dynamics are independent AR(1) processes across components:

$$x_{0,j} \sim \mathcal{N}(0, 1), \qquad x_{t,j} \mid x_{t-1,j} \sim \mathcal{N}\big(\alpha x_{t-1,j},\ \sigma^2\big), \quad j = 1, \ldots, 4.$$

Because both the transition and observation models factorise across dimensions, the joint prior and likelihood factorise, and the joint posterior is the product of the marginal posteriors (when the components are independent). This independence allows us to simulate and — if desired — filter each component separately using univariate methods without constructing an intractable tensor-product grid.

R code: simulate a 4-dimensional latent state and Binomial observations

The following R script simulates a length-T trajectory of the 4D latent process and corresponding Binomial observations. It returns the latent matrix X (T-by-4) and the observation matrix Y (T-by-4). No plotting is performed.

```r
# 4D Binomial-Logistic State Space Simulation
simulate_4d_binomial_logistic <- function(T = 100L,
                                          d = 4L,
                                          M = 50L,
                                          alpha = 0.99,
                                          sigma2 = 0.11,
                                          seed = 47) {
  stopifnot(d == 4L)
  set.seed(seed)

  sigma <- sqrt(sigma2)

  # Preallocate matrices:
  # rows = time 0..T (we'll return times 1..T for observations)
  X <- matrix(0, nrow = T, ncol = d)
  Y <- matrix(0L, nrow = T, ncol = d)

  # Initial state x_0 ~ N(0,1) for each component
  x_prev <- rnorm(d, mean = 0, sd = 1)

  for (t in seq_len(T)) {
    # Transition: x_t = alpha * x_{t-1} + epsilon_t
    x_curr <- alpha * x_prev + rnorm(d, mean = 0, sd = sigma)

    # Observation probability via logistic link
    p_t <- 1 / (1 + exp(-x_curr))
    # Numerical safety: clamp p away from exact 0 or 1
    eps <- 1e-12
    p_t <- pmin(pmax(p_t, eps), 1 - eps)

    # Binomial observations (vectorised across components)
    y_t <- rbinom(d, size = M, prob = p_t)

    # Store
    X[t, ] <- x_curr
    Y[t, ] <- y_t

    # advance
    x_prev <- x_curr
  }

  colnames(X) <- paste0("x", 1:d)
  colnames(Y) <- paste0("y", 1:d)

  list(X = X, Y = Y, params = list(T = T, d = d, M = M, alpha = alpha,
                                   sigma2 = sigma2))
}
```

```r
# Example usage (runs simulation but does not plot):
res <- simulate_4d_binomial_logistic(T = 200, d = 4, M = 50, alpha = 0.99,
                                     sigma2 = 0.11, seed = 47)
```

Notes

- The function is written for $d = 4$ but can be generalised to any d by removing the `stopifnot(d==4L)` guard. Because components are independent, the code fully vectorises across dimensions.
- For reproducibility, `seed` is an input; seed(47) has been used in all my work so far
- The script clamps the logistic probabilities away from exact 0/1 to avoid numerical issues in downstream likelihood computations.

Uniform Grid-Based Filtering in 4D

While a naive approach to filtering in four dimensions would require constructing a tensor-product grid over $\mathbb{R}^4$, such a method is computationally infeasible due to the exponential scaling in the number of grid points. Fortunately, in the present model, both the transition and observation models factorise across the four dimensions: each latent component evolves as an independent AR(1) process, and each observation depends only on the corresponding latent state. This independence implies that the joint posterior factorises as a product of univariate marginals. Hence, it suffices to implement a one-dimensional uniform grid filter for each dimension separately, which can be run in parallel. This approach yields exact marginals for each component without incurring the curse of dimensionality.

```r
# 1D uniform grid filter for Binomial-logistic state space
grid_filter_1d <- function(y, M, alpha = 0.99, sigma2 = 0.11,
                           grid_min = -6, grid_max = 6, grid_size = 200) {
  T <- length(y)
  sigma <- sqrt(sigma2)

  # Grid
  grid <- seq(grid_min, grid_max, length.out = grid_size)
  dx <- grid[2] - grid[1]

  # Transition kernel matrix: f(x_j | x_i)
  trans_mat <- outer(grid, grid, function(x, xp) {
    dnorm(x, mean = alpha * xp, sd = sigma)
  }) * dx

  # Initial prior density: N(0,1) on the grid
  prior <- dnorm(grid, mean = 0, sd = 1)
  prior <- prior / sum(prior)

  # Storage
  post <- matrix(NA, nrow = T, ncol = grid_size)
  mean_est <- numeric(T)

  filt <- prior
  for (t in seq_len(T)) {
    # Prediction: integrate over transition
    pred <- as.vector(trans_mat %*% filt)

    # Likelihood at grid points
```

```r
    p <- 1 / (1 + exp(-grid))
    like <- dbinom(y[t], size = M, prob = p)

    # Update
    unnorm <- pred * like
    filt <- unnorm / sum(unnorm)

    # Save
    post[t, ] <- filt
    mean_est[t] <- sum(grid * filt)
  }

  list(grid = grid, post = post, mean_est = mean_est)
}

# 4D wrapper: apply independent 1D filters to each dimension
grid_filter_4d <- function(Y, M = 50, alpha = 0.99, sigma2 = 0.11,
                           grid_min = -6, grid_max = 6, grid_size = 200) {
  d <- ncol(Y)
  T <- nrow(Y)

  res_list <- vector("list", d)
  mean_est <- matrix(NA, nrow = T, ncol = d)

  for (j in seq_len(d)) {
    res_list[[j]] <- grid_filter_1d(Y[, j], M, alpha, sigma2,
                                    grid_min, grid_max, grid_size)
    mean_est[, j] <- res_list[[j]]$mean_est
  }

  colnames(mean_est) <- paste0("x", 1:d, "_mean")
  list(filters = res_list, mean_est = mean_est)
}

# Example: simulate data then filter
sim <- simulate_4d_binomial_logistic(T = 50, d = 4, M = 50)
Y <- sim$Y

res <- grid_filter_4d(Y, M = 50, alpha = 0.99, sigma2 = 0.11,
                      grid_min = -6, grid_max = 6, grid_size = 200)
```

Evaluation via Normalized Root Mean Squared Error (NRMSE)

To assess the performance of the grid-based filter, we compare its posterior mean estimates of the latent states to the true simulated latent trajectories. For each latent dimension $j = 1, \ldots, 4$, we compute the root mean squared error (RMSE) between the estimated state sequence $\{\hat{x}_{t,j}\}_{t=1}^{T}$ and the true state sequence $\{x_{t,j}\}_{t=1}^{T}$:

$$\mathrm{RMSE}_j = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\big(\hat{x}_{t,j} - x_{t,j}\big)^2}.$$

Since the scale of the latent states may differ across components, we normalize by the dynamic range of the true latent process in each dimension, given by $\max_t x_{t,j} - \min_t x_{t,j}$. This yields the normalized root mean squared error (NRMSE):

$$\mathrm{NRMSE}_j = \frac{\mathrm{RMSE}_j}{\max_t x_{t,j} - \min_t x_{t,j}}.$$

This normalization ensures comparability across dimensions by scaling the error relative to the amplitude of the underlying latent trajectory. A smaller NRMSE value indicates that the filter tracks the true latent process more accurately in that dimension. For the four-dimensional model, we report the NRMSE separately for each latent component, thereby quantifying the accuracy of the grid-based filter along each independent axis of the state space.

Table 3: NRMSE of Grid-Based Filter vs True Latent States (4D)

Dimension	NRMSE
x1	0.0920
x2	0.0536
x3	0.0428
x4	0.0436

From Component-Wise NRMSE to an Overall Measure

For each latent dimension $j = 1, \ldots, 4$, the normalized root mean squared error (NRMSE) was originally defined as

$$\mathrm{NRMSE}_j \;=\; \frac{\sqrt{\frac{1}{T}\sum_{t=1}^{T}\big(\hat{x}_{t,j} - x_{t,j}\big)^2}}{\max_t x_{t,j} - \min_t x_{t,j}},$$

where $\hat{x}_{t,j}$ is the posterior mean estimate from the filter and $x_{t,j}$ the true simulated state.
This yields four separate values, one for each latent component.

To obtain a single summary error measure across all dimensions, we pool the trajectories together. Instead of averaging the four NRMSE values, we compute a global root mean squared error (RMSE) across all time steps and all dimensions simultaneously:

$$\mathrm{RMSE}_{\mathrm{overall}} \;=\; \sqrt{\frac{1}{4T}\sum_{j=1}^{4}\sum_{t=1}^{T}\big(\hat{x}_{t,j} - x_{t,j}\big)^2}.$$

This quantity is then normalized by the dynamic range of the entire latent trajectory, taken over all dimensions:

$$\text{NRMSE}_{\text{overall}} \;=\; \frac{\text{RMSE}_{\text{overall}}}{\max_{t,j} x_{t,j} - \min_{t,j} x_{t,j}}.$$

In this way, the four-dimensional error is condensed into a single unit-free value that reflects the average discrepancy between the filter estimates and the true latent states, relative to the global scale of variation of the latent process.

Table 4: Overall NRMSE of Grid-Based Filter vs True Latent States (4D)

Metric	Value
Overall NRMSE	0.0335

Computational Time of the Uniform Grid Filter

For comparison, it is also useful to quantify the computational cost of the uniform grid-based filter. Unlike the adaptive grid filter, the uniform filter maintains a fixed grid across all time steps, covering the entire plausible range of the latent state. The execution time includes both the prediction step (matrix multiplication with the transition kernel) and the measurement update step for each time step and all four latent dimensions. Measuring and comparing the run time of the uniform versus adaptive filters highlights the efficiency gains of the adaptive grid approach.

```
## Unit: milliseconds
##            expr      min       lq     mean median       uq      max neval
##  uniform_filter 111.0744 114.1502 118.8355 120.92 123.8841 124.1489     5
```

Adaptive Grid-Based Filtering (centering & truncation)

We now want to also implement this adaptive grid-based idea we discussed earlier, in the 4 Dimensional setting.

Mathematical idea

Consider the four-dimensional state vector $x_t = (x_{t,1}, \ldots, x_{t,4})^\top$ whose components evolve independently as

$$x_{t,j} \mid x_{t-1,j} \sim \mathcal{N}(\alpha x_{t-1,j}, \sigma^2), \qquad j = 1, \ldots, 4,$$

and are observed via independent binomial channels

$$y_{t,j} \mid x_{t,j} \sim \text{Binomial}(M, \kappa(x_{t,j})), \qquad \kappa(x) = \frac{1}{1+e^{-x}}.$$

Because the transition and observation models factorise across components, the joint posterior factorises and the marginal posterior for each component can be obtained independently.

At each time t we maintain a discrete approximation to the marginal posterior density $\pi_t(x_j)$ on a finite grid. The adaptive grid strategy proceeds as follows: given a posterior π_{t-1} on a grid G_{t-1}, (i) compute the predictive density $\pi_{t|t-1}$ by convolving π_{t-1} with the Gaussian transition kernel, (ii) compute the predictive mean $\mu_{t|t-1}$ and variance $\sigma^2_{t|t-1}$, (iii) choose a new grid G_t centred at $\mu_{t|t-1}$ and truncated to $\mu_{t|t-1} \pm k\sqrt{\sigma^2_{t|t-1}}$ (with k large enough, e.g. $k \in [4, 8]$), (iv) interpolate $\pi_{t|t-1}$ onto G_t, and (v) form the posterior π_t by multiplying the interpolated predictive by the likelihood and re-normalising.

Intuition and benefits

The latent AR(1) dynamics are smooth and the logistic observation saturates outside moderate ranges, so the posterior mass at each time is typically concentrated in a small region of the real line. A fixed wide uniform grid wastes resolution in irrelevant regions; an adaptive grid places resolution where it matters by re-centering and truncating based on the predictive mean and variance. This yields higher local accuracy for a fixed number of grid points and reduces computational cost. Because we treat each of the four dimensions separately, this approach avoids the exponential blow-up of a tensor-product grid while still producing exact marginal posteriors for each dimension under the independence assumption.

```r
## R: Adaptive grid-based filter (1D) and 4D wrapper

adaptive_grid_filter_1d <- function(y, M = 50, alpha = 0.99, sigma2 = 0.11,
                                    grid_size = 300, k = 6, min_width = 1.0,
                                    prior_mean = 0, prior_sd = 1,
                                    tol_mass = 1e-8) {
  # y: vector of observations length T
  # returns posterior means & optionally posterior distributions on evolving grids
  T <- length(y)
  sigma <- sqrt(sigma2)

  # --- initialize grid from prior ---
  init_sd <- prior_sd
  init_mean <- prior_mean
  half_width <- max(k * init_sd, min_width / 2)
  grid <- seq(init_mean - half_width, init_mean + half_width,
              length.out = grid_size)
  dx <- grid[2] - grid[1]

  # initial prior density (on grid) and normalise
  prior <- dnorm(grid, mean = prior_mean, sd = prior_sd)
  prior <- prior / sum(prior)

  # storage
  post_means <- numeric(T)
  post_sds   <- numeric(T)

  filt <- prior
  old_grid <- grid

  for (t in seq_len(T)) {
    # --- Prediction on old grid: pred = \int f(x|x') filt(x') dx' ---
    # Compute transition matrix from old_grid -> old_grid
    dx_old <- old_grid[2] - old_grid[1]
    trans_mat <- outer(old_grid, old_grid, function(x, xp) {
      dnorm(x, mean = alpha * xp, sd = sigma)
    }) * dx_old
    pred <- as.vector(trans_mat %*% filt)  # predictive density on old_grid
    # numerical guard
    pred[pred < 0] <- 0
    if (sum(pred) <= 0) pred <- rep(1 / length(pred), length(pred))
    pred <- pred / sum(pred)

    # --- compute predictive mean/var on old grid ---
```

```
  mu_pred  <- sum(old_grid * pred)
  var_pred <- sum((old_grid - mu_pred)^2 * pred)
  sd_pred  <- sqrt(var_pred)
  # ensure min width
  half_width_new <- max(k * sd_pred, min_width / 2)
  new_grid <- seq(mu_pred - half_width_new, mu_pred + half_width_new,
                  length.out = grid_size)

  # --- interpolate predictive onto new grid ---
  interp_pred <- approx(x = old_grid, y = pred, xout = new_grid, rule = 2)$y
  # set negatives to zero, renormalise
  interp_pred[interp_pred < 0] <- 0
  total_mass <- sum(interp_pred)
  if (total_mass < (1 - tol_mass)) {
    # try expanding bounds once if unacceptable mass lost
    expand_factor <- 1.5
    half_wider <- expand_factor * half_width_new
    new_grid <- seq(mu_pred - half_wider, mu_pred + half_wider,
                    length.out = grid_size)
    interp_pred <- approx(x = old_grid, y = pred, xout = new_grid,
                          rule = 2)$y
    interp_pred[interp_pred < 0] <- 0
    total_mass <- sum(interp_pred)
  }
  if (total_mass <= 0) interp_pred <- rep(1/length(interp_pred),
                                          length(interp_pred))
  pred_on_new <- interp_pred / sum(interp_pred)

  # --- Measurement update on new grid ---
  p_vals <- 1 / (1 + exp(-new_grid))
  # clamp numerically
  eps <- 1e-12
  p_vals <- pmin(pmax(p_vals, eps), 1 - eps)
  like <- dbinom(y[t], size = M, prob = p_vals)
  unnorm_post <- pred_on_new * like
  unnorm_post[unnorm_post < 0] <- 0
  if (sum(unnorm_post) <= 0) {
    # avoid degenerate posterior: revert to predictive
    post <- pred_on_new
  } else {
    post <- unnorm_post / sum(unnorm_post)
  }

  # --- record posterior summary ---
  post_mean <- sum(new_grid * post)
  post_var  <- sum((new_grid - post_mean)^2 * post)
  post_means[t] <- post_mean
  post_sds[t]   <- sqrt(post_var)
  #grids_list[[t]] <- new_grid
  #post_list[[t]] <- post

  # prepare for next iteration
  old_grid <- new_grid
```

```
    filt <- post
  }

  list(mean = post_means, sd = post_sds
       #, grids = grids_list, posts = post_list
       )
}

# Wrapper: apply adaptive 1D filter to each of 4 dimensions independently
adaptive_grid_filter_4d <- function(Y, M = 50, alpha = 0.99, sigma2 = 0.11,
                                    grid_size = 300, k = 6, min_width = 1.0,
                                    prior_mean = 0, prior_sd = 1) {
  # Y: T x d matrix (d expected to be 4)
  T <- nrow(Y)
  d <- ncol(Y)
  results_means <- matrix(NA, nrow = T, ncol = d)
  results_sds   <- matrix(NA, nrow = T, ncol = d)
  for (j in seq_len(d)) {
    res_j <- adaptive_grid_filter_1d(y = Y[, j], M = M, alpha = alpha,
                                     sigma2 = sigma2,
                                    grid_size = grid_size, k = k,
                                    min_width = min_width,
                                    prior_mean = prior_mean,
                                    prior_sd = prior_sd)
    results_means[, j] <- res_j$mean
    results_sds[, j]   <- res_j$sd
  }
  colnames(results_means) <- paste0("x", 1:d, "_mean")
  colnames(results_sds)   <- paste0("x", 1:d, "_sd")
  list(mean = results_means, sd = results_sds)
}

# Example usage: simulate and run adaptive filter
# (requires simulate_4d_binomial_logistic() from earlier)
set.seed(42)
sim <- simulate_4d_binomial_logistic(T = 200, d = 4, M = 50, alpha = 0.99,
                                     sigma2 = 0.11, seed = 47)
Y <- sim$Y
true_X <- sim$X

# run adaptive filter (per-dimension)
ad_res <- adaptive_grid_filter_4d(Y, M = 50, alpha = 0.99, sigma2 = 0.11,
                                  grid_size = 300, k = 6, min_width = 1.0)

# optional: compute overall NRMSE comparing ad_res$mean to true_X
calc_nrmse_overall <- function(true_X, est_X) {
  mse <- mean((est_X - true_X)^2)
  rmse <- sqrt(mse)
  denom <- max(true_X) - min(true_X)
  rmse / denom
}
overall_nrmse_adaptive <- calc_nrmse_overall(true_X, ad_res$mean)
```

Evaluation of the Adaptive Grid Filter via NRMSE

To evaluate the performance of the adaptive grid-based filter, we compare the posterior mean estimates of the latent states to the true simulated trajectories. As in the uniform grid case, we use the normalized root mean squared error (NRMSE) as a unit-free metric.

The key difference is that here the posterior estimates are obtained from the adaptive grid filter, which re-centers and truncates the grid at each time step based on the predictive mean and variance. Despite this difference in filtering methodology, the overall NRMSE is computed in the same way: the root mean squared error is calculated across all time steps and all four latent dimensions, and is then normalized by the dynamic range of the true latent trajectories. This yields a single summary value quantifying the average deviation between the adaptive filter estimates and the true latent states.

Table 5: NRMSE of Adaptive Grid Filter vs True Latent States (4D)

Metric	Value
Overall NRMSE (Adaptive Grid Filter)	0.0342

Computational Time of the Adaptive Grid Filter

While the adaptive grid filter improves accuracy by concentrating grid points around regions of high posterior probability, it is also important to quantify the computational cost of the algorithm. We measure the total execution time required to process an entire sequence of observations for all four latent dimensions. This timing includes both the prediction (convolution with the transition kernel) and the measurement update steps for each time step, across all dimensions. Comparing execution times under different grid sizes or truncation parameters provides practical insight into the trade-off between accuracy and efficiency.

```
## Unit: seconds
##             expr      min       lq    mean   median       uq      max neval
##  adaptive_filter 9.289865 9.294196 9.40551 9.396251 9.397165 9.650074     5
```

Bootstrap (Sequential Importance Resampling) Particle Filter

We implement the standard bootstrap particle filter (also known as Sequential Importance Resampling, SIR), which is the simplest and most commonly-used SMC method and is the baseline used in the ORCSMC paper. In the bootstrap particle filter the transition prior is used as the importance proposal; hence particle weights are proportional to the observation likelihood.

Model (reminder)

We recall the 4-dimensional Binomial–logistic state space model (components independent).

$$x_t = \alpha x_{t-1} + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, \sigma^2 I_4), \tag{16}$$

$$y_{t,j} \mid x_{t,j} \sim \text{Binomial}\big(M,\ \kappa(x_{t,j})\big), \quad \kappa(x) = \frac{1}{1+e^{-x}}, \quad j = 1, \ldots, 4. \tag{17}$$

We denote observations by $Y_{1:T} \in \mathbb{Z}^{T\times 4}$.

Bootstrap particle filter: mathematics

Let N be the number of particles. At time $t-1$ we have particles $\{x_{t-1}^{(i)}\}_{i=1}^N$ and normalized weights $\{w_{t-1}^{(i)}\}_{i=1}^N$. The bootstrap filter uses the transition prior as proposal:

$$x_t^{(i)} \sim f(x_t \mid x_{t-1}^{(i)}) = \mathcal{N}(\alpha x_{t-1}^{(i)}, \sigma^2 I_4).$$

Because the proposal equals the prior, the incremental importance weights simplify to the observation likelihood:

$$\tilde{w}_t^{(i)} = p(y_t \mid x_t^{(i)}), \qquad w_t^{(i)} = \frac{\tilde{w}_t^{(i)}}{\sum_{k=1}^N \tilde{w}_t^{(k)}}.$$

Effective sample size (ESS) is monitored as

$$\text{ESS}_t = \frac{1}{\sum_{i=1}^N (w_t^{(i)})^2}.$$

When ESS_t falls below a threshold (e.g. $N/2$), we resample the particle set (systematic or stratified resampling) and reset weights to $1/N$. The particle approximation of the filtering mean is

$$\hat{x}_t = \sum_{i=1}^N w_t^{(i)} x_t^{(i)}.$$

Algorithm (summary)

1. **Initialise:** Sample $x_0^{(i)} \sim p(x_0) = \mathcal{N}(0, I_4)$, set $w_0^{(i)} = 1/N$.
2. **For** $t = 1, \ldots, T$:
   (a) *Propagate:* $x_t^{(i)} \sim \mathcal{N}(\alpha x_{t-1}^{(i)}, \sigma^2 I_4)$.
   (b) *Weight:* $\tilde{w}_t^{(i)} = p(y_t \mid x_t^{(i)})$ (product over 4 independent channels; compute in log-space).
   (c) *Normalise:* $w_t^{(i)} \propto \tilde{w}_t^{(i)}$.
   (d) *Estimate:* $\hat{x}_t = \sum_i w_t^{(i)} x_t^{(i)}$.
   (e) *Resample (optional):* If $\text{ESS}_t < \tau N$ (e.g. $\tau = 0.5$), draw N new particles by systematic resampling and set weights to $1/N$.

The bootstrap filter is simple to implement, numerically stable if likelihoods are computed in log-space, and provides the baseline Monte Carlo scaling RMSE $\sim \mathcal{O}(N^{-1/2})$ (CLT for SMC at fixed t).

Convergence Study: Grid-based Filters vs Particle Filter

We empirically compare the performance of three filtering approaches—*Uniform Grid Filter*, *Adaptive Grid Filter*, and *Bootstrap Particle Filter*—on a simulated four-dimensional binomial logistic state-space model. The latent state $\mathbf{X}_t \in \mathbb{R}^4$ evolves according to an autoregressive process, and observations $\mathbf{Y}_t$ are generated via a binomial logistic likelihood. This setup allows direct evaluation of estimation error against the true latent states.

Metrics and evaluation

For each method, we record at each replicate:

- **Overall Normalised Root Mean Squared Error (NRMSE):**

$$\text{NRMSE}_{\text{overall}} = \frac{\sqrt{\frac{1}{4T}\sum_{j=1}^{4}\sum_{t=1}^{T}(\hat{x}_{t,j} - x_{t,j})^2}}{\max_{t,j} x_{t,j} - \min_{t,j} x_{t,j}}.$$

- **CPU time per run.**

These quantities are aggregated over R replicate simulations to compute mean and standard error of the NRMSE and mean computation time. The results are summarised in the table `summary_df`, which reports, for each method and parameter setting, the mean and standard error of NRMSE, mean and standard error of CPU time, the number of replicates, and the relevant filter parameters.

Experimental setup

- **Grid filters (Uniform / Adaptive):** Grid sizes $K \in \{50, 100, 200\}$. For the Adaptive Grid, we fix the refinement parameter $k = 6$ in the pilot study.
- **Particle filter:** Number of particles $N \in \{250, 500, 1000\}$.
- **Replicates:** Each configuration is repeated over R independent simulated trajectories.
- **Shared parameters:** $T = 100$, $M = 50$, auto-regressive parameter $\alpha = 0.99$, innovation variance $\sigma^2 = 0.11$.

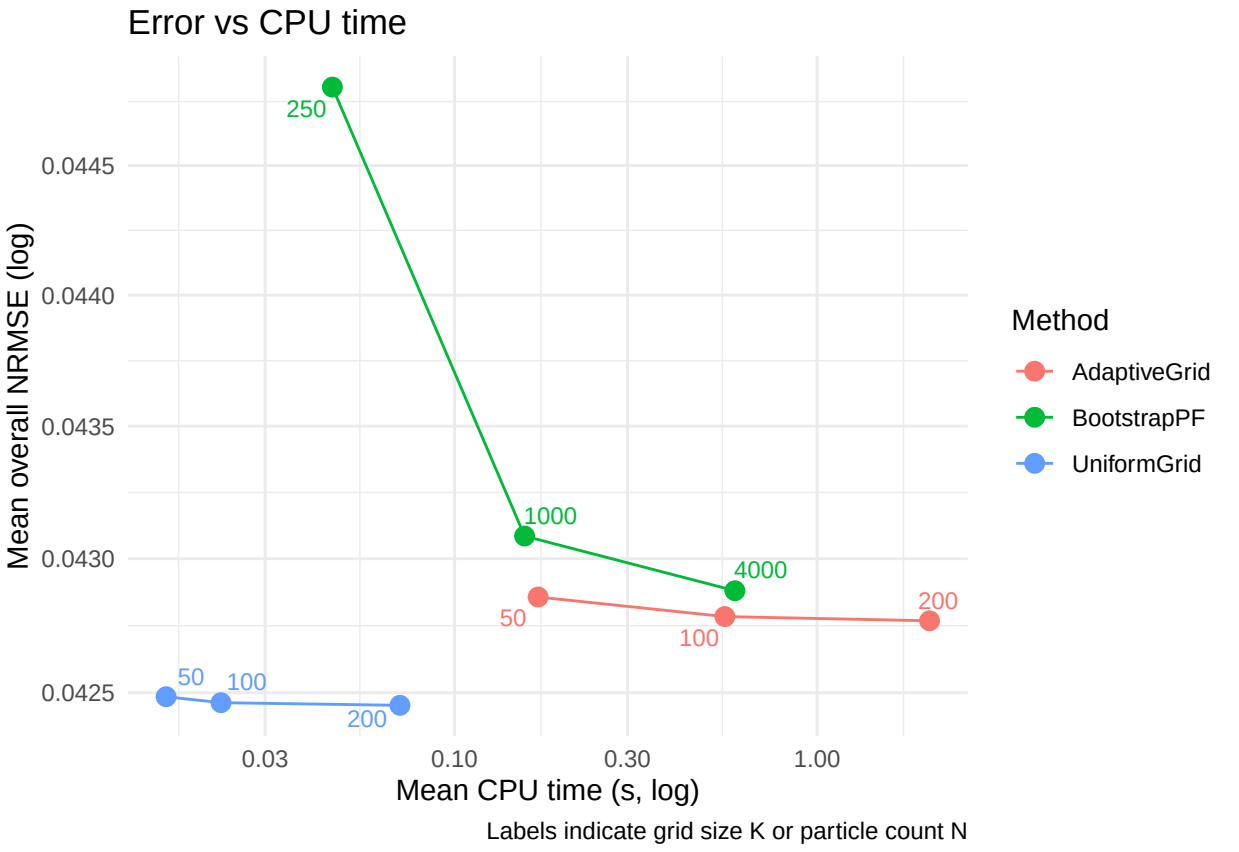
Analysis

- We see a table produced displaying the mean NRMSE and mean time for each method, with the different parameter values, as wellas the standard error for each of these measurements.
- Log–log plots of mean NRMSE versus CPU time are produced for all three methods, allowing a direct comparison of estimation accuracy per unit computational cost.

This analysis quantifies the trade-offs between accuracy and computational effort across methods, highlighting regimes where grid-based filters are efficient and where particle filters are preferable.

Table 6: Summary of NRMSE and CPU time across methods and parameters

method	param	param_type	mean_nrmse	se_nrmse	mean_time	se_time	n_reps
AdaptiveGrid	50	K	0.042857	0.001851	0.170000	0.001952	15
AdaptiveGrid	100	K	0.042783	0.001842	0.556667	0.009241	15
AdaptiveGrid	200	K	0.042768	0.001840	2.044000	0.015700	15
BootstrapPF	250	N	0.044806	0.001792	0.046000	0.002726	15
BootstrapPF	1000	N	0.043085	0.001785	0.156000	0.001902	15
BootstrapPF	4000	N	0.042880	0.001834	0.593333	0.004102	15
UniformGrid	50	K	0.042485	0.001789	0.016000	0.001902	15
UniformGrid	100	K	0.042463	0.001785	0.022667	0.002482	15
UniformGrid	200	K	0.042454	0.001783	0.070667	0.002667	15



Discussion of Filtering Methods

The plot comparing different filtering methods for estimating the latent state of our 4-dimensional state-space model reveals clear differences in both accuracy and computational efficiency. The uniform grid filter consistently achieves the lowest mean NRMSE across all grid sizes ($K = 50, 100, 200$) while maintaining very low CPU times. Increasing the grid size slightly improves accuracy, but the computational cost remains almost unchanged, suggesting that for this low-dimensional and well-behaved posterior, uniform grids are both highly efficient and accurate. The adaptive grid filter, on the other hand, incurs higher computational cost for the same grid sizes, with only a modest improvement in accuracy as K increases. This indicates that the adaptive refinement overhead is not justified in this context, and such an approach may be more appropriate for highly skewed or multi-modal posteriors. The bootstrap particle filter shows the expected Monte Carlo behavior: NRMSE decreases as the number of particles N increases, but even with $N = 4000$, accuracy remains slightly worse than the grid-based methods, and CPU time is orders of magnitude larger. This highlights that particle filtering is less efficient for low-dimensional, smooth posteriors, becoming competitive only at very high particle counts at a substantial computational cost. Overall, these results suggest that for low-dimensional and smooth latent processes, deterministic grid-based filters dominate particle methods, providing the best trade-off between accuracy and efficiency in this model.

Temporal evolution of NRMSE

In addition to summary statistics, we examine the time-resolved performance of each filter. For a single simulated trajectory, we compute the NRMSE at each time step:

$$\mathrm{NRMSE}_t = \frac{\sqrt{\frac{1}{4}\sum_{j=1}^{4}(\hat{x}_{t,j} - x_{t,j})^2}}{\max_j x_{t,j} - \min_j x_{t,j}}, \quad t = 1, \ldots, T.$$

These per-time-step NRMSE curves are plotted for multiple parameter settings of each filter, producing a visualization of estimation error dynamics over time. This allows us to compare:

- How quickly each method tracks the latent state.
- Differences in stability and variability of the estimates across time.
- The impact of resolution parameters (K for grid filters, N for the particle filter) on the temporal accuracy of state estimation.

These plots complement the log–log summary by providing intuition on transient behavior and the evolution of estimation error, which is particularly relevant in settings with time-varying latent states or non-linear observation models.