Week 4 - Exploring Numerical Integration Approaches

Max Richards

2025-08-06

Summary of week 3

Last week was quite a busy week, as I worked with a real data set for the first time. The data set involved figures over 20+ years regarding exchange rates of many different currencies in comparison to the euro. I focussed only on the GBP and USD conversions and explored the bootstrap particle filter in this setting, to try and identify a hidden latent state over time for these measures.

I considered this model in some depth, exploring different ways of implementation, and ways it could be improved, such as dynamic noise scaling, mean-reverting processes, and how the re-sampling strategy could be done in different ways, spcefically looking at the ESS threshold idea, and multi-nomial re-sampling.

I also took some more time to develop my understanding of different measures used to track effectiveness of the filtering methods, with a particular focus on the covariance matrix determinant and trace measures.

After speaking to Adam, I was encouraged to explore two key ideas. The first was to look at a stochastic volatility model for that particular type of data set, as it was more suited to the nature of the problem, and the second was to explore the use of Numerical Integration Approaches, this was a key focus discussed right from the get go of the project and so was excited to begin researching this work.

Understanding Numerical Integration Approaches

In the study of state-space models, especially those involving nonlinear and non-Gaussian dynamics, particle filtering has become a widely used method for sequential inference. However, alternative techniques exist that avoid the randomness and resampling inherent in particle filters. Among these are numerical integration approaches, often referred to as grid-based filters, which provide a deterministic means of approximating the posterior distribution of the latent state.

Numerical integration approaches involve discretising the state space into a fixed grid and directly computing posterior densities using deterministic integration techniques, rather than relying on Monte Carlo sampling. These methods approximate the required integrals in the Bayesian filtering recursion by replacing them with finite sums over grid points, enabling precise computation of probability distributions.

Mathematical Interpretation

Consider some non-linear state space model:

$$x_t = f(x_{t-1}) + \epsilon_t \ , y_t = h(x_t) + \eta_t$$

At each time t the filtering task is to compute the posterior density $p(x_t|y_{1:t})$, which involves solving the following;

$$p(x_t|y_{1:t}) = \frac{p(y_t|x_t) \times \int p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1}) \ dx_{t-1}}{p(y_t|y_{1:t-1})}$$

Whereas, the gird-based methods compute integrals, compute only the integral,

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

The integral, which is the same as the integral in the numerator of the filtering methods, is evaluated using numerical quadrature, such as the trapezoidal rule or Gaussian quadrature, by summing over discretised values of x_{t-1}.

How do they work?

Grid-based filtering at each time step, have many different ways of being setup and ran, but below I provide a general overview for this family of methods.

1. **Discretisation:** The latent state space is divided into a grid $\{x_1, x_2, \ldots, x_n\}$.
2. **Prediction:** The prior density $p(x_t|y_{1:t-1})$ is computed via numerical integration of the transition density over the previous posterior.
3. **Update:** Bayes' rule is applied pointwise across the grid using the new observation y_t, yielding the posterior density $p(x_t|y_{1:t})$.
4. **Normalisation:** The posterior is normalised to ensure it integrates to 1.

Comparing with Particle Filtering methods

Feature	Particle Filter	Numerical Integration (Grid-Based)
Approximation method	Monte Carlo sampling (particles)	Deterministic numerical integration
Representation of posterior	Set of weighted particles	Discrete probability density over a grid
Randomness	Involves randomness and resampling	Fully deterministic
Dimensionality scalability	Scales better to high dimensions	Suffers from curse of dimensionality
Accuracy in low dimensions	May require many particles for accuracy	High accuracy possible with fine grids
Variance	Subject to sampling variance	No sampling variance
Resampling step	Required to avoid degeneracy	Not required

Why might we use these Grid-Based methods?

While particle filters offer significant flexibility and scalability, they can suffer from particle degeneracy and high variance, particularly when the number of particles is limited. In contrast, numerical integration approaches offer a more stable and accurate estimation procedure in low-dimensional settings by avoiding randomness and directly computing posterior densities. As such, these methods provide a useful benchmark or alternative to particle filtering when computational resources and model dimensionality permit.

In the context of this project, implementing and comparing grid-based filters alongside particle filters will provide deeper insight into the practical trade-offs between these approaches and highlight scenarios where each method may be preferable.

Working with the Univariate Linear Gaussian State-Space Model

To introduce and implement the principles of grid-based filtering, we begin with a foundational example, as advised by Adam, the univariate linear Gaussian state-space model (LGSSM). This class of models is not only mathematically tractable but also forms the basis for the well-known Kalman filter, which provides an exact recursive solution under linearity and Gaussianity assumptions. By focusing first on this simple model, we gain a controlled environment in which to compare the performance of our grid-based filtering algorithm against the analytical benchmark.

Model Specification

The LGSSM is defined by the following pair of equations:

$$x_t = \phi x_{t-1} + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \sigma_x^2), y_t = x_t + \eta_t, \quad \eta_t \sim \mathcal{N}(0, \sigma_y^2)$$

Here, $x_t \in \mathbb{R}$ represents the latent state at time t, and $y_t \in \mathbb{R}$ denotes the observed data. The system is governed by two independent white noise processes: process noise ϵ_t and observation noise η_t , both assumed to be Gaussian with variances σ_x^2 and σ_y^2, respectively. The parameter $\phi \in (-1, 1)$ controls the persistence of the latent state and reflects the dynamics of the hidden process.

This model serves as an ideal test bed for validating the grid-based approach to filtering for several reasons:

Analytical tractability: The Kalman filter provides the exact posterior distribution of the latent state, allowing direct comparison with the numerically approximated posteriors produced by the grid-based method.

Univariate simplicity: The one-dimensional nature of the latent state avoids the curse of dimensionality inherent in grid-based methods, ensuring computational feasibility.

Well-understood behavior: The linear and Gaussian assumptions provide predictable and interpretable behavior, helping isolate the effects of the approximation method itself.

The Simulated Example of Choice

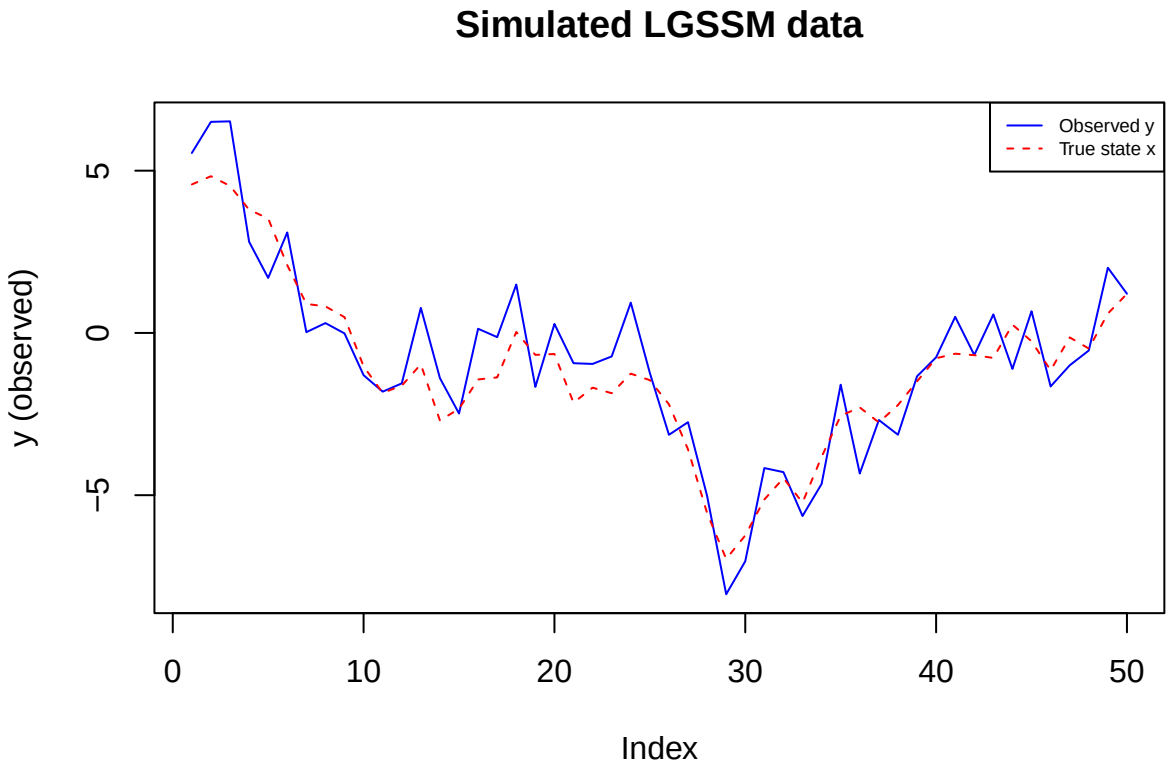
To implement and test the method, we simulate data from the LGSSM with predefined parameters. Specifically, we consider an autoregressive coefficient $\phi = 0.9$, process noise standard deviation $\sigma_x = 1$, and observation noise standard deviation $\sigma_y = 1$. A synthetic time series of length $T = 50$ is generated, with the initial state drawn from the stationary distribution of the process:

$$x_1 \sim \mathcal{N}\left(0, \frac{\sigma_x^2}{1 - \phi^2}\right)$$

Subsequent states are generated recursively via the state equation, and observations are obtained by adding Gaussian noise to each latent state.

This simulated dataset will serve as the basis for evaluating the accuracy and behavior of our grid-based filtering method.

See the results of the simulated under a fixed seed (for sake of reproducibility) below:


Simulated LGSSM data
Observed y
True state x
y (observed)
5
0
−5
0
10
20
30
40
50
Index

Implementing the Grid-Based Filter

Following some reading, I was prepared to implement the a full grid-based filtering procedure for approximating the posterior distribution $p(x_t|y_{1:t})$ over a fixed discretised state space. The outline of this method will be discussed in the literature below, and I will display the results of such a method as well as comparison of results obtained to other filters such as the Kalman, for which we know holds the exact solution. The method I implemented provides a deterministic, numerical alternative to sampling-based particle filters and exploits the structure of the LGSSM to compute the filtering denisty recursively.

Recall that our model is of the form

$$x_t = \phi x_{t-1} + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \sigma_x^2) \quad \text{(State Equation)} y_t = x_t + \eta_t, \quad \eta_t \sim \mathcal{N}(0, \sigma_y^2) \quad \text{(Observation Equation)}$$

And let $\chi = \{x^{(1)}, x^{(2)}, ..., x^{(N)} \subset \mathbb{R}$ denote the fixed grid on which the filtering distributions are approximated. Aty each time t, we maintain a discrete approximation of $p(x_t|y_{1:t})$, denoted as $\hat{p}_t(x^{(i)})$ for grid point $x^{(i)} \in \chi$.

STEP 1: Initialisation

We assume that the initial state x_1 has a Gaussian prior distribution, $x_1 \sim \mathcal{N}(0, \sigma_1^2)$. This is evaluated over the grid, which follows;

$$\hat{p}_1(x^{(i)}) \propto exp\left(-\frac{(x^{(i)})^2}{2\sigma_1^2}\right), \quad i = 1, ..., N$$

and then we normalise, such that;

$$\hat{p}_1(x^{(i)}) \leftarrow \frac{\hat{p}_1(x^{(i)})}{\sum_{j=1}^{N} \hat{p}_1(x^{(j)})\Delta x}$$

where Δx is our *grid spacing*, which for our initial case is assumed to be *Uniform* over the state space. This we have that $\Delta x = x^{(2)} - x^{(1)}$

STEP 2: Prediction (Time Update)

Given the posterior $\hat{p}_{t-1}(x_{t-1})$, we compute the predicted prior for x_t using the Chapman-Kolmogorov equation *(Recall from ST227)* :

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

This is approximated on our defined grid as:

$$\hat{p}_t^{prior}(x^{(i)}) = \sum_{j=1}^{N} p(x^{(i)}|x^{(j)})\hat{p}_{t-1}(x^{(j)})\Delta x$$

Where $p(x^{(i)}|x^{(j)}) = \mathcal{N}(x^{(i)}; \phi x^{(j)}, \sigma_x^2)$ is the transition density.

This step effectively convolves the previous posterior with the Gaussian transition Kernel.

STEP 3: Update (Measurement Step)

Given the observation y_t, we update the predicted density using Bayes' rule:

$$\hat{p}_t(x^{(i)}) \propto p(y_t|x^{(i)}) \times \hat{p}_t^{prior}(x^{(i)})$$

where the likelihood is $p(y_t|x^{(i)}) = \mathcal{N}(y_t; x^{(i)}, \sigma_y^2)$. Then we normalise:

$$\hat{p}_t(x^{(i)}) \leftarrow \frac{\hat{p}_t(x^{(i)})}{\sum_{j=1}^{N} \hat{p}_t(x^{(j)})\Delta x}$$

This gives us the approximate posterior at time t.

Summary and Overview

Let $\chi = \{x^{(1)}, x^{(2)}, ..., x^{(N)}$ be the grid with spacing Δx, which in our initial method is uniform.

For $t = 1, ..., T$:

1. Initialise at $t = 1$; $\hat{p}_1(x^{(i)}) \propto \mathcal{N}(x^{(i)}; 0, \sigma_1^2)$
2. Prediction for $t \geq 2$; $\hat{p}_t^{prior}(x^{(i)}) = \sum_{j=1}^{N} \mathcal{N}(x^{(i)}; \phi x^{(j)}, \sigma_x^2) \times \hat{p}_{t-1}(x^{(j)}) \times \Delta x$
3. Update ; $\hat{p}_t(x^{(i)}) \propto \mathcal{N}(y_t; x^{(i)}, \sigma_y^2) \times \hat{p}_t^{prior}(x^{(i)})$
4. Normalise ; $\hat{p}_t(x^{(i)}) \leftarrow \frac{\hat{p}_t(x^{(i)})}{\sum_{j=1}^{N} \hat{p}_t(x^{(j)})\Delta x}$

This method is effectively a recursive convolution–multiplication–normalisation process. All integrals are approximated as finite sums on a fixed grid. The update step is equivalent to element-wise multiplication of vectors (prior × likelihood). This approach is exact in the limit as $N \to \infty$, but practically limited by computational costs due to scaling with $\mathcal{O}(N^2)$.

The grid and the intuition behind it

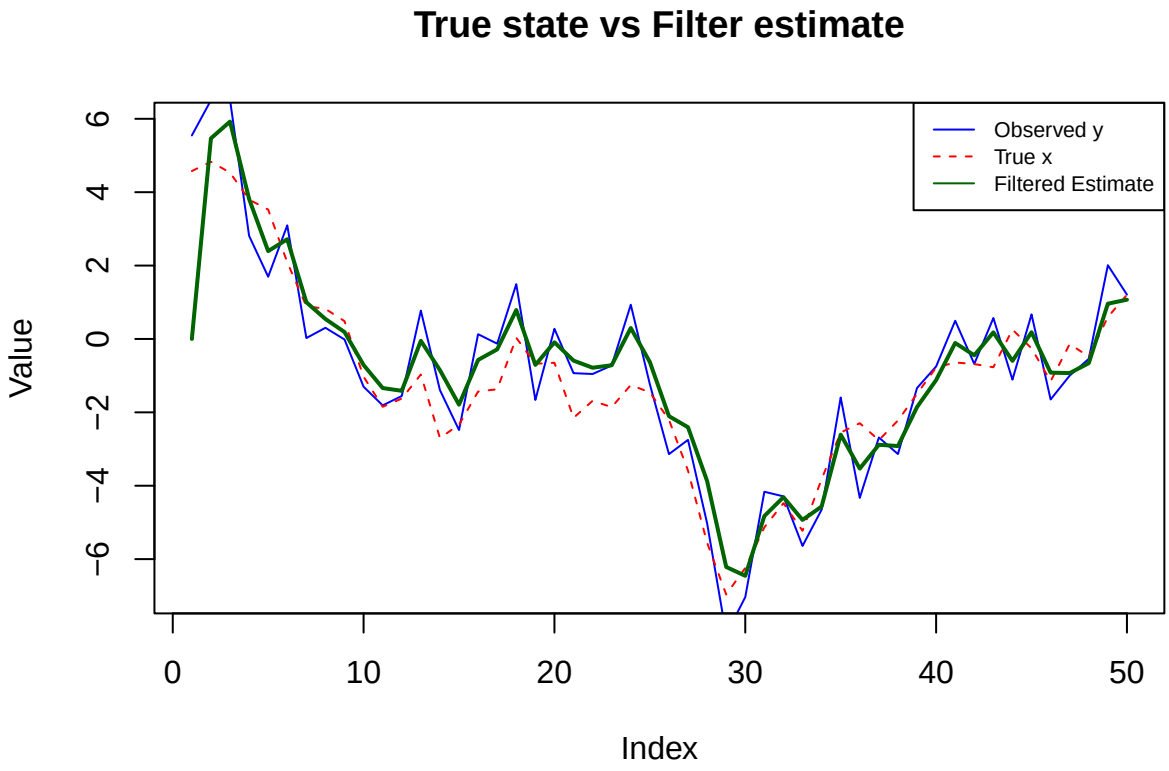
Think of the grid as:

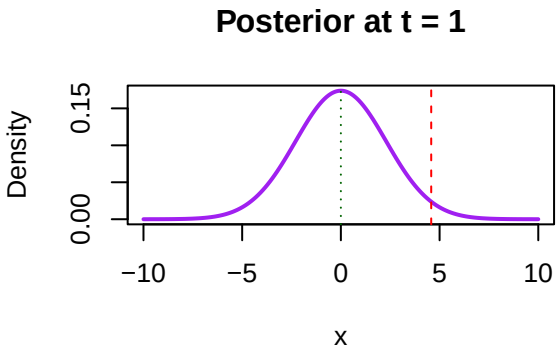
A discretised version of the possible state values, like a "ruler" laid out across the state space (in 1D case). On this ruler, you track probabilities, update them as new data comes in, and move probability mass from one grid point to another according to the dynamics of the model.

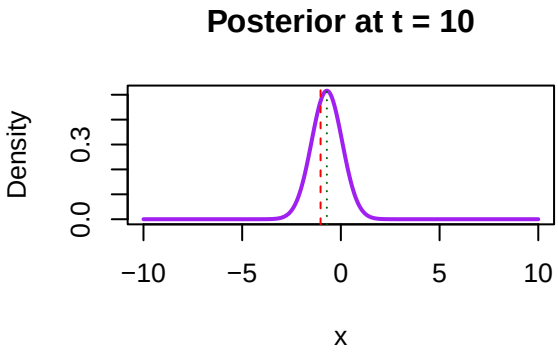
I used $N = 500$ in this code over an interval of $[-10, 10]$ to define my grid. Considering this first model followed a Uniform structure though, it meant simply that $\Delta x = \frac{20}{500}$.

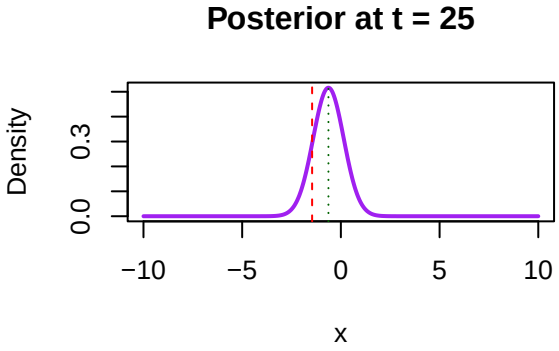
The output generated by the code implementing this strategy can be found below:

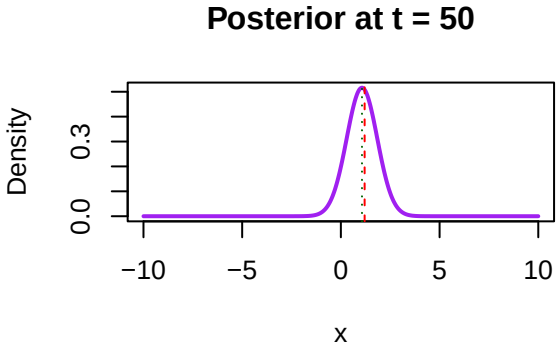
Results of Implementation





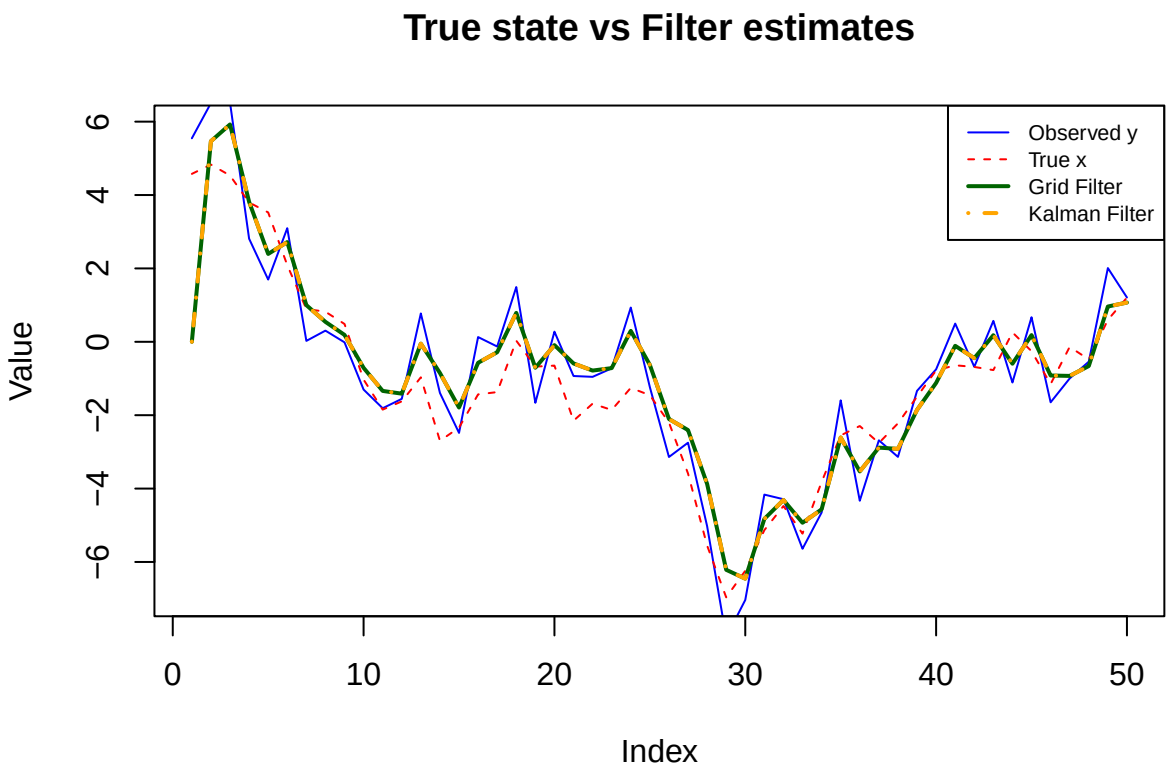






Implementing the Kalman Filter

As seen in the past, I know that given the linear nature of the state space defined, the Kalman filter will perform best as it gives the exact solution. Thus, I deemed it wise to implement this filter method and add it to the plot produced earlier, using same code and ideas from week 1 of this project, to validate my grid method implemented above.



Comparison of methods

Of course, I wanted to see how this grid-based filter matched up to the Kalman filter, and the most natural means of comparison at least in the 1D setting is the use of RMSE calculations, so that was what I performed next. The results are displayed in the following table:

Table 2: Comparison of RMSE for Grid-Based and Kalman Filters

Method	RMSE
Grid-based Filter	1.0146
Kalman Filter	1.0146

Evidently, the grid-based method implemented is performing very well, in fact it appears to be following the exact same trajectory as the Kalman filter, which is a very good sign in terms of our approximations of the posterior by numerical means.

Remember, I am working with a linear Gaussian state-space model (LGSSM) and this is exactly the model that the Kalman filter is optimally designed for. So the Kalman filter produces the exact Bayesian posterior at every step. My grid-based filter is approximating that same posterior numerically. And since I am using a fine enough grid (500 points across a reasonable support), the approximation is very accurate.

Hence, the filtered means from both methods match very closely, possibly to the point of being visually indistinguishable on the plot.

In fact, it is found that the maximum absolute error across all time points is 0.0001027 (4sf).

Comparing the Grid-Based method and the Kalman filter Implementation

Despite the very close approximations, there are of course differences between these two methods, and these are displayed below:

Table 3: Comparison of Kalman Filter vs. Grid-Based Filter in LGSSM

Feature	Kalman.Filter	Grid.Based.Filter
Assumptions	Requires linearity and Gaussian noise	No such assumption — works for any model if grid is fine enough
Representation	Tracks just mean and variance (parametric Gaussian)	Represents entire posterior over a fixed grid (non-parametric)
Computation	Very fast, closed-form recursive equations	Much slower: uses convolution + Bayes rule on a grid
Accuracy (LGSSM)	Optimal estimator (exact)	Approximates the same exact result — may suffer from grid error
Flexibility	Limited to linear-Gaussian models	Can be extended to nonlinear and non-Gaussian systems

Exploring Adaptive-Grids

In standard grid-based filtering, like the method I implemented above, the grid over the state space is fixed in range (e.g., from -10 to 10) with uniform in spacing (e.g., 500 points equally spaced). But this comes with problems. You may be wasting resolution in areas where the posterior is essentially zero and so may miss important detail in high-probability regions if your grid is too coarse.

Adaptive grid filtering dynamically refines or moves the grid to better represent the posterior distribution.

There are many types of strategies for creating and working with adaptive grids, but There is 4 that I want to consider at this moment in time. These are;

1. Center and truncate around posterior mass

At each time step, estimate the current posterior's mean and variance, then create a new grid centered at the mean and spanning, say, +/- 4 standard deviations. This keeps the grid focused where probability is non-negligible.

2. Adaptive refinement (non-uniform grids)

Use finer grid spacing where the posterior is sharply peaked, and coarser spacing elsewhere. Can be based on second derivatives of the posterior, or entropy, or local variance. This is generally more complex, and often uses spline or kernel interpolation.

3. Grid merging or splitting

When we maintain a set of intervals or "bins" that adapt over time. If posterior mass becomes concentrated in a region, split that bin into smaller ones. If a region becomes irrelevant, merge its bins to reduce computational cost. This is similar to adaptive mesh refinement used in numerical PDE solvers.

4. Logarithmic or quantile-based transforms

Transform the grid to work in a more natural scale: E.g., apply a log or logit transform if state space is bounded or skewed, or define grid points to be quantiles of the previous posterior, ensuring more points in high-density regions.

Table 4: Benefits of Adaptive grid Approaches

Benefit	Description
Efficiency	Use fewer grid points while maintaining accuracy
Accuracy	Better resolution where the posterior is complex
Scalability	Makes grid filtering possible for more complex models
Extends usability	Works better in non-Gaussian, nonlinear models

But, despite the benefits of these adaptive grids, there are of course considerations that come with implementation. These include;

1. Recomputing the grid at each time step.
2. Interpolating the previous posterior onto the new grid.
3. Carefully adjusting the convolution/prediction step for non-uniform grids.

Adaptive Grid Approach 1 - Center and truncate around posterior mass

So of course, the fundamental process here is the same as the grid-based method previously discussed, just with a different way of defining and calculating our grid. But what is actually mathematically different in this new setting?

1. Posterior Centering:

At each time t, after computing the posterior $p(x_t|y_{1:t})$, I get;

$$\mu_t = \int x \times p(x|y_{1:t})\, dx\ , \quad \sigma_t^2 = \int (x-\mu_t)^2 \times p(x|y_{1:t})\, dx$$

2. Grid re-definition:

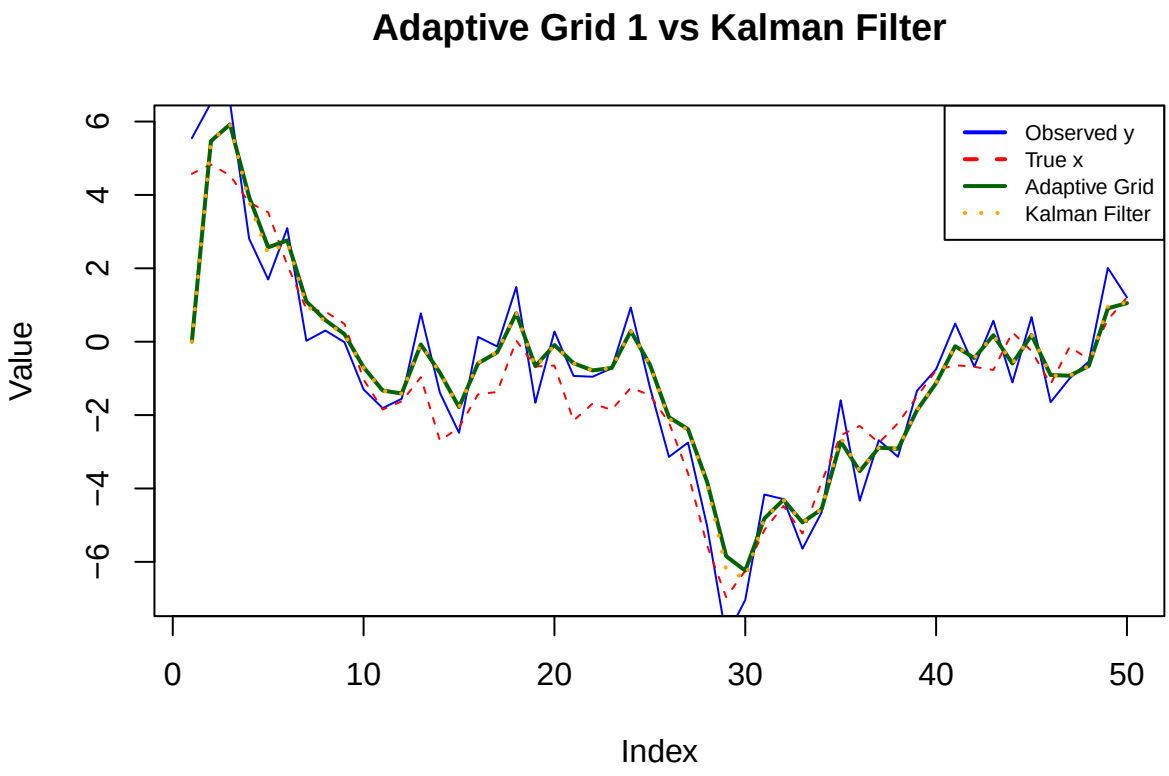
This step focuses the grid points around where the posterior mass is actually concentrated. So for instance, instead of a static $[-10, 10]$ grid, we redefine the grid at each time stamp to be:

$$x_{grid}^t = linspace(\mu_t - 4\sigma_t, \mu_t + 4\sigma_t, N)$$

3. Interpolation:

The posterior from the previous step (on the old grid) is interpolated onto the new adaptive grid. This keeps the filter focused, efficient, and accurate even in models where the posterior would otherwise drift outside a static grid.

Find below the results of implementing this adaptive grid-based strategy:



Adaptive Grid Approach 2 - Adaptive refinement (non-uniform grids)

Again, the fundamental process here is the same as the grid-based methods previously discussed, just with a different way of defining and calculating our grid. But what is actually mathematically different in this new setting?

This approach implements a non-uniform adaptive refinement grid filter, where grid spacing is dynamically adjusted to be finer in high-density regions of the posterior and coarser in the tails. This is a more sophisticated form of adaptive filtering than simply recentering the grid.

In previous (uniform) adaptive grid approach, the grid was uniformly spaced but moved (recentered) at each time step. Whereas, in the non-uniform adaptive refinement, grid spacing depends on the posterior density: More grid points are allocated in regions where posterior is high (steep slopes or peaks) and fewer points in tails where posterior is flat.

This is effectively using an inverse CDF transform to distribute grid points more efficiently.

Interpolation is used to transfer posterior to new grid $\rightarrow$ approximating continuous density.

This technique leads to better accuracy for the same number of grid points, especially in cases with skewed or multi-modal posteriors.

Mathematical Interpretation

As mentioned above, in non-uniform adaptive filtering, the grid points themselves are no longer equally spaced. Instead, they are concentrated in high-density regions of the posterior, and sparser in low-density regions (like tails).

1. Dynamic Grid Construction

Instead of fixed x_i, we generate a new grid at each time stamp using the posterior density $p_t(x)$. Let $CDF_t(x) = \int_{-\infty}^{x} p_t(u)\, du$. Then choose a uniform grid on the CDF scale: $u_1, u_2, ..., u_N \in [0, 1]$. Then invert the CDF numerically to get: $x_i^{(t)} = CDF_t^{-1}(u_i)$. This step concentrates grid points where the posterior is steep (high slope of CDF $\implies$ high density), and fewer pointswhere its flat (tails).

2. Non-Uniform Integration Weights

In uniform grids, integration is approximated via:

$$\int f(x)\, dx \approx \sum_i f(x_i)\, \Delta x$$

However, in non-uniform grids, we must use local-spacing:

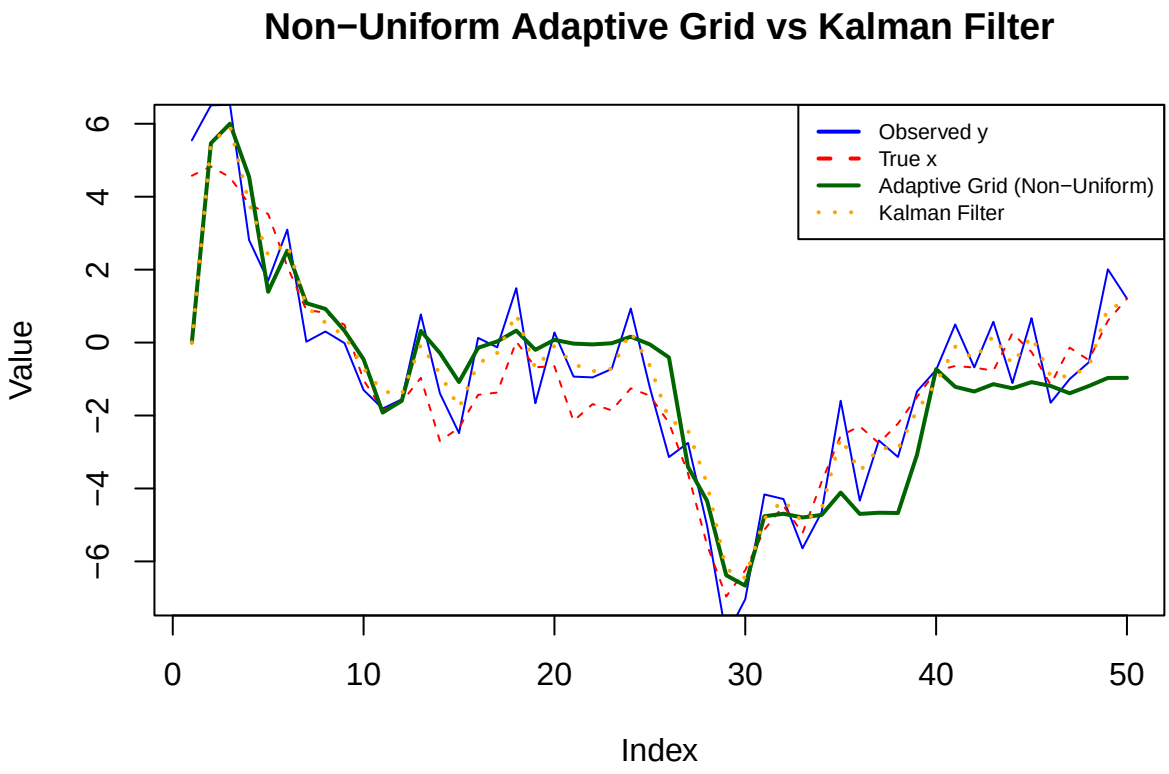
$$\int f(x)\, dx \approx \sum_i f(x_i)\, \Delta x_i, \text{ where } \Delta x_i = x_{i+1} - x_i$$

And so these weights now vary across the domain and are updated at every time stamp.

3. Interpolation and Re-Sampling

Because the grid is changing at each step, the posterior must be interpolated onto the new grid. This is akin to re-sampling in particle filters but done deterministically using interpolation.

Find below the results of implementing this adaptive grid-based strategy:



Adaptive Grid Approach 3 - Grid merging or splitting

This method, grid merging/splitting, dynamically refines or coarsens the grid based on the shape of the posterior distribution — allocating more resolution where the posterior is dense or changing rapidly, and less where it's flat or negligible.

How It Works — Intuitively

Start with a coarse grid (fewer intervals) over the domain. For each interval (or bin):

• If posterior mass is high → split that interval into finer subintervals.

• If posterior mass is low (i.e., region is uninformative) → merge neighboring bins to reduce resolution and computational load.

• The process is adaptive over time, meaning the grid structure evolves based on the data and state uncertainty.

Mathematical Interpretation

Suppose we start with an initial grid $\mathcal{G}_0 = \{x_1, x_2, ..., x_K\}$. Each interval (bin) is $[x_i, x_{i+1}]$ with midpoint $m_i = \frac{x_i + x_{i+1}}{2}$.

1. Compute Posterior Mass per Bin

Let $p_t(x)$ be the current posterior. Compute bin mass:

$$M_i = \int_{x_i}^{x_{i+1}} p_t(x)\, dx \approx p_t(m_i) \times (x_{i+1} - x_i)$$

2. Refinement (Split)

If $M_i > \mathcal{T}_{high}$, we split the bin into two sections:

$$[x_i, x_{i+1}] \to [x_i, m_i], [m_i, x_{i+1}]$$

3. Coarsening (Merge)

If $M_i < \mathcal{T}_{low}$, and neighboring bin also has low mass, we merge them:

$$[x_i, x_{i+1}], [x_{i+1}, x_{i+2}] \to [x_i, x_{i+2}]$$

4. Redistribute Posterior

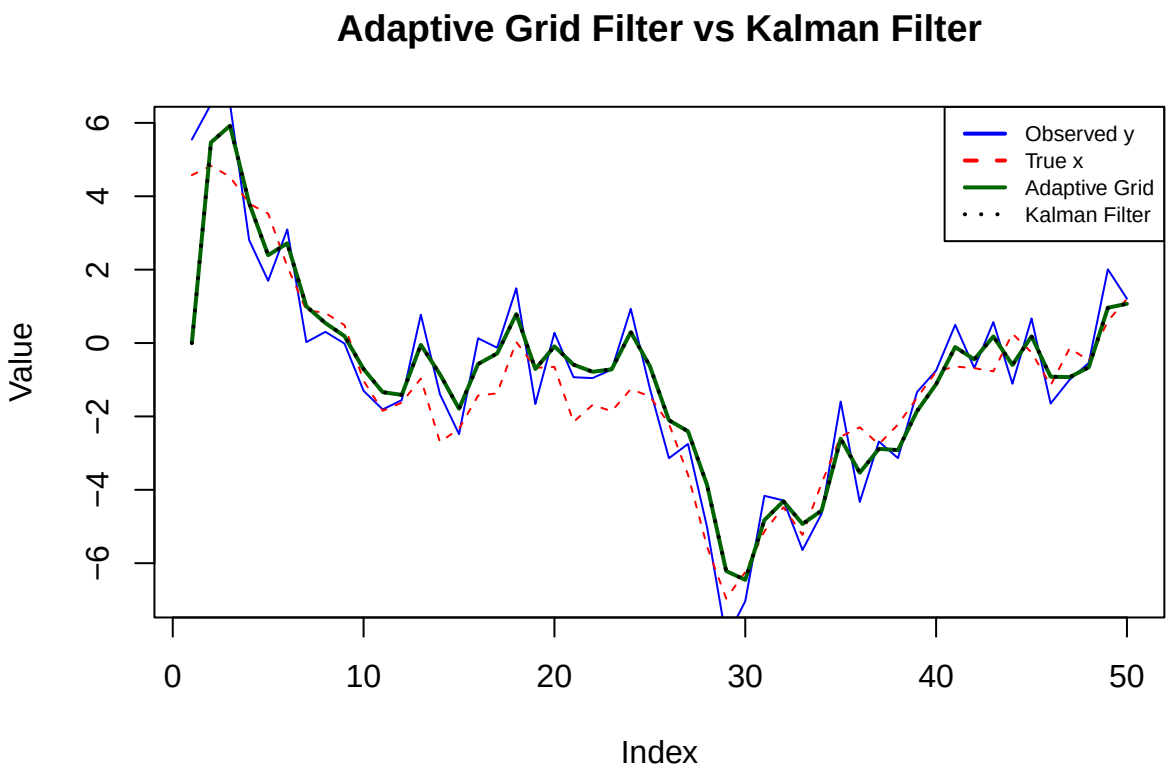
Interpolate or redistribute mass across new bins using previous density approximations.

5. Integration and Estimation

Now, all sums/integrals over the state space use adaptive bin widths:

$$\mathbb{E}[x_t] \approx \sum_i m_i \times p_t(m_i) \times (x_{i+1} - x_i)$$

Find below the results of implementing this adaptive grid-based strategy:



Comparison of methods

Summary of the 3 methods

Aspect	Approach 1: Recenter + Uniform Grid	Approach 2: Non-Uniform Grid via CDF Transform	Approach 3: Grid Splitting/Merging
Grid spacing	Uniform	Non-uniform, adaptive to posterior density	Adaptive bin widths via split/merge
Grid location	Centered at posterior mean	Grid points distributed via inverse CDF	Varies locally by posterior mass
Computational complexity	Low to moderate	Moderate (requires CDF inversion)	Higher (dynamic data structures)
Accuracy	Good if posterior near Gaussian	Better for skewed, multimodal posteriors	Efficient for highly variable posterior shapes
Integration weights	Equal Δx	Variable Δx_i	Variable bin widths
Posterior interpolation	Yes	Yes	Yes
Handling multimodality	Limited	Good	Very good
Suitable for	Posteriors near Gaussian, drifting	Complex posteriors, multimodal, skewed	Large dynamic posterior changes, efficiency-focused

Comparing summary statistics for the methods

Now following the implementation of these 3 adaptive grid approaches, I decided to compare the effectiveness of them all against one another. Once again, given the 1D nature of the state space I am dealing with, RMSE felt like the most natural and effective means of comparison.

Table 6: RMSE Comparison Across Filtering Approaches

Method	RMSE
Adaptive Grid Filter 1 (Centered Uniform Grid)	1.019458
Adaptive Grid Filter 2 (Non-Uniform Grid)	1.393978
Adaptive Grid Filter 3 (Grid Splitting/Merging)	1.014558
Kalman Filter	1.014560

Moving into 2 Dimensions

Now assume that the latent state is a 2D vector, $x_t = (x_t^{(1)}, x_t^{(2)})^{\mathbb{T}}$ Of course, in this new setting, the state equation and observation equation will change.

State Equation

Let the latent state be a 2-dimensional vector:

$$\mathbf{x}_t = \begin{bmatrix} x_t^{(1)} \\ x_t^{(2)} \end{bmatrix}.$$

The state evolves as:

$$\mathbf{x}_t = \Phi \mathbf{x}_{t-1} + \boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_x),$$

where

$$\Phi = \begin{bmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{bmatrix}$$

is a general 2×2 transition matrix that allows for coupling between components, and

$$\Sigma_x = \begin{bmatrix} \sigma_{x_{11}}^2 & \sigma_{x_{12}} \\ \sigma_{x_{12}} & \sigma_{x_{22}}^2 \end{bmatrix}$$

is a full covariance matrix allowing correlation between state noise components.

Observation Equation

Observations $\mathbf{y}_t \in \mathbb{R}^2$ are:

$$\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\eta}_t, \quad \boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_y),$$

with observation noise covariance matrix

$$\Sigma_y = \begin{bmatrix} \sigma_{y_{11}}^2 & \sigma_{y_{12}} \\ \sigma_{y_{12}} & \sigma_{y_{22}}^2 \end{bmatrix},$$

which can also have non-zero off-diagonal terms indicating correlation between observation noises.

Grid Filtering Procedure

Define a 2D grid of $N \times N$ points:

$$\chi = \{(x^{(i)}, x^{(j)}) : i, j = 1, \ldots, N\},$$

with uniform grid spacing $\Delta x, \Delta y$.

At each time t, approximate the posterior density over the grid:

$$\hat{p}_t(x^{(i)}, x^{(j)}) \approx p(\mathbf{x}_t = (x^{(i)}, x^{(j)}) \mid \mathbf{y}_{1:t}).$$

Prediction step:

$$\hat{p}_t^{prior}(x^{(i)}, x^{(j)}) = \sum_{m=1}^{N} \sum_{n=1}^{N} p(\mathbf{x}_t = (x^{(i)}, x^{(j)}) \mid \mathbf{x}_{t-1} = (x^{(m)}, x^{(n)})) \times \hat{p}_{t-1}(x^{(m)}, x^{(n)}) \times \Delta x \Delta y,$$

where the transition density is

$$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}\big(\mathbf{x}_t; \Phi \mathbf{x}_{t-1}, \Sigma_x\big),$$

a full bivariate normal allowing cross-coupling.

Update step:

$$\hat{p}_t(x^{(i)}, x^{(j)}) \propto p(\mathbf{y}_t \mid \mathbf{x}_t = (x^{(i)}, x^{(j)})) \times \hat{p}_t^{prior}(x^{(i)}, x^{(j)}),$$

with likelihood

$$p(\mathbf{y}_t \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{y}_t; \mathbf{x}_t, \Sigma_y).$$

Normalization:

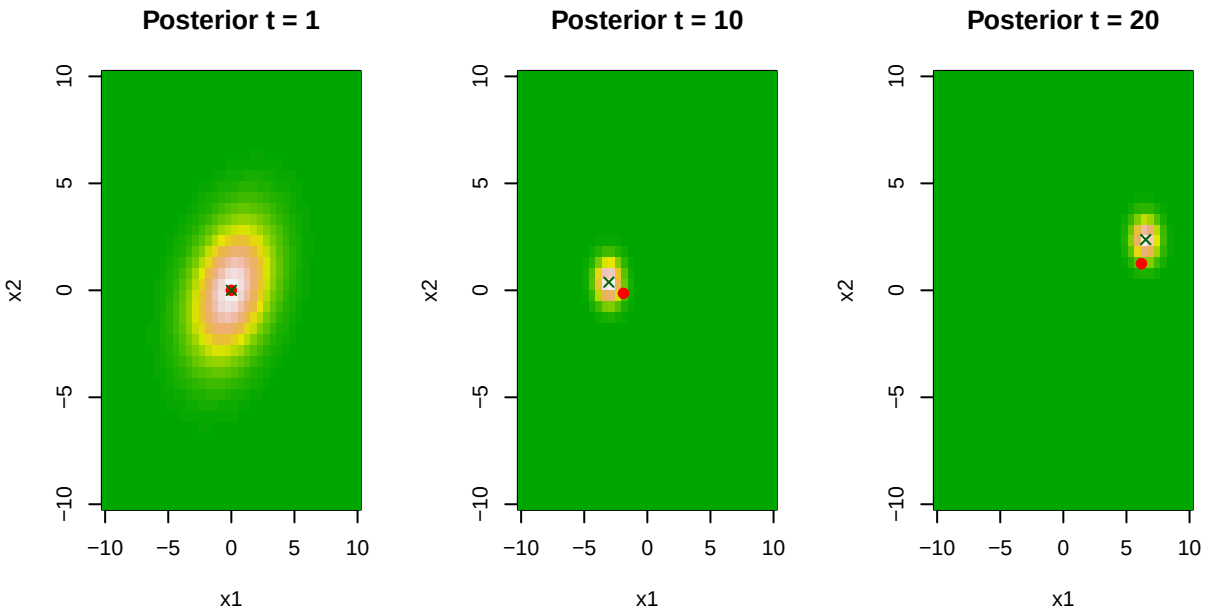
$$\hat{p}_t(x^{(i)}, x^{(j)}) \leftarrow \frac{\hat{p}_t(x^{(i)}, x^{(j)})}{\sum_{i=1}^{N} \sum_{j=1}^{N} \hat{p}_t(x^{(i)}, x^{(j)}) \Delta x \Delta y}.$$

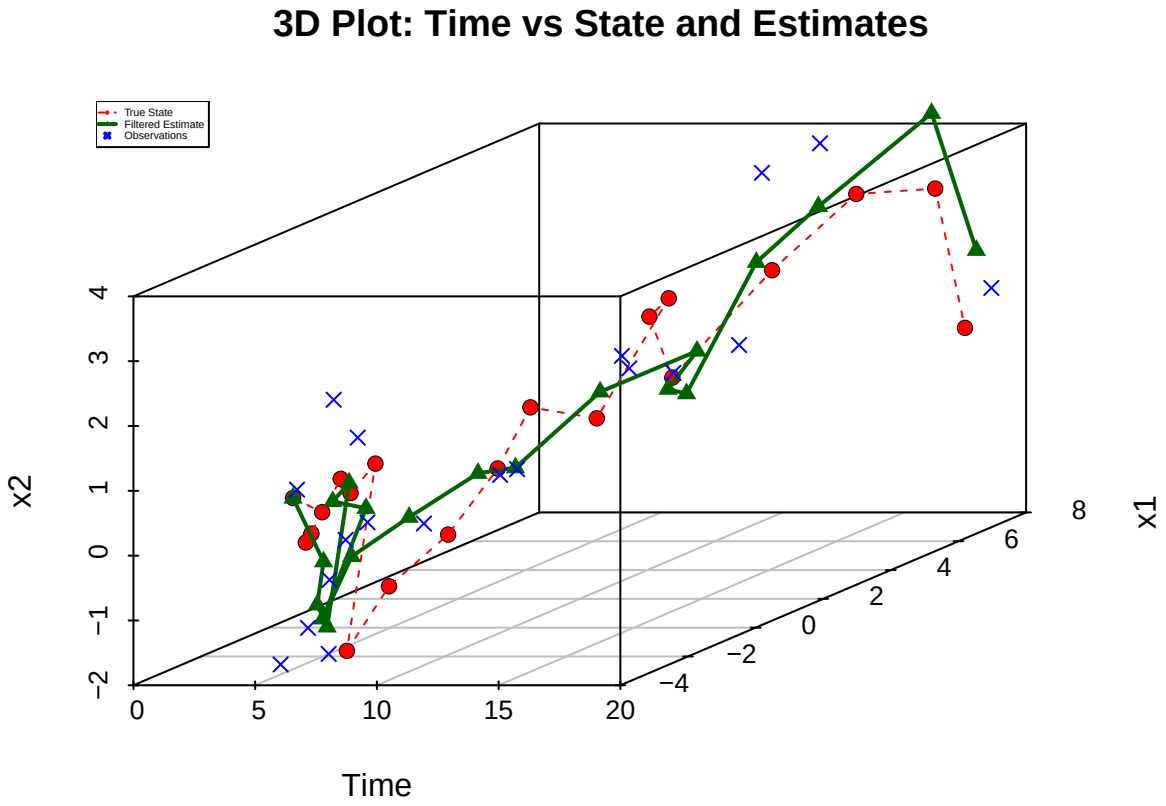
Intuition

The 2D grid discretizes the continuous bivariate state space. The transition matrix Φ allows interaction between state variables, while Σ_x models correlated process noise. The filtering proceeds by:

- **Prediction:** Convolve the previous posterior with the bivariate Gaussian transition kernel determined by Φ and Σ_x, shifting and spreading probability mass in the 2D state space.
- **Update:** Element-wise multiply the predicted distribution by the bivariate likelihood determined by the observed $\mathbf{y}_t$ and Σ_y, incorporating new information.
- **Normalization:** Normalize to ensure the posterior integrates to 1 over the 2D grid.

This general formulation captures dependencies and correlations in both the state dynamics and observation process.


Posterior t = 1
Posterior t = 10
Posterior t = 20
x1
x2
−10
−5
0
5
10


3D Plot: Time vs State and Estimates
True State
Filtered Estimate
Observations
Time
x1
x2