Week 2 progressions

Max Richards

2025-07-24

Summary of Week 1

In the first week of this project I explored the use of a simple linear Gaussian state space model, implementing a simulation of this, and then estimating the simulations trajectory, using the bootstrap particle filter, the Kalman filter, and a Numerical grid filter.

All 3 methods followed the trajectory of the model well, however this of course was a very simple case, in fact, likely the most simple possible case, and so this week was all about building on from this beginning, and exploring new state space models, as well as opening up to more than 1 dimension.

Updated Model 1: Non-Gaussian Noise

An initial idea I had was to keep the state space model, rather similar, however instead of simulating our noise/errors using a simple Gaussian model, I decided to use the students t-distribution.

So our modified model became;

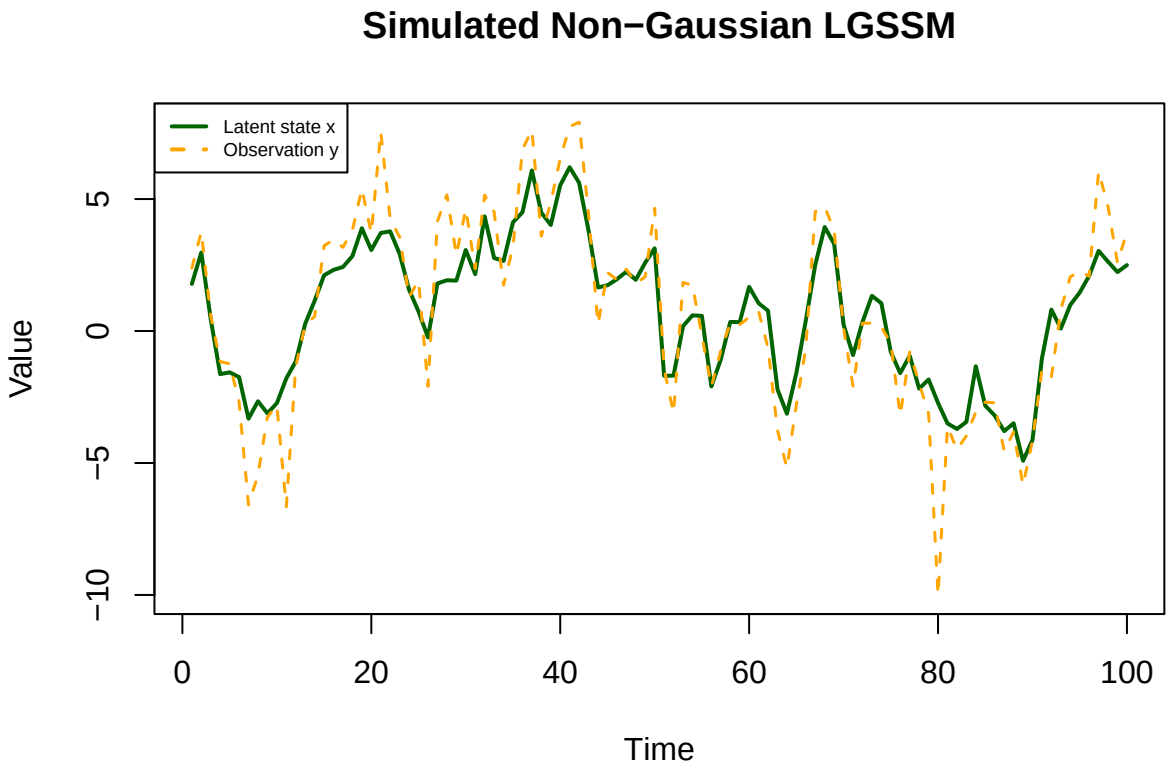
Consider the uni variate linear model of the form;

$$y_t = bx_t + \epsilon_t, \epsilon_t \sim t_{v_1}(0, \sigma_\epsilon^2)$$

Let us take the state equation;

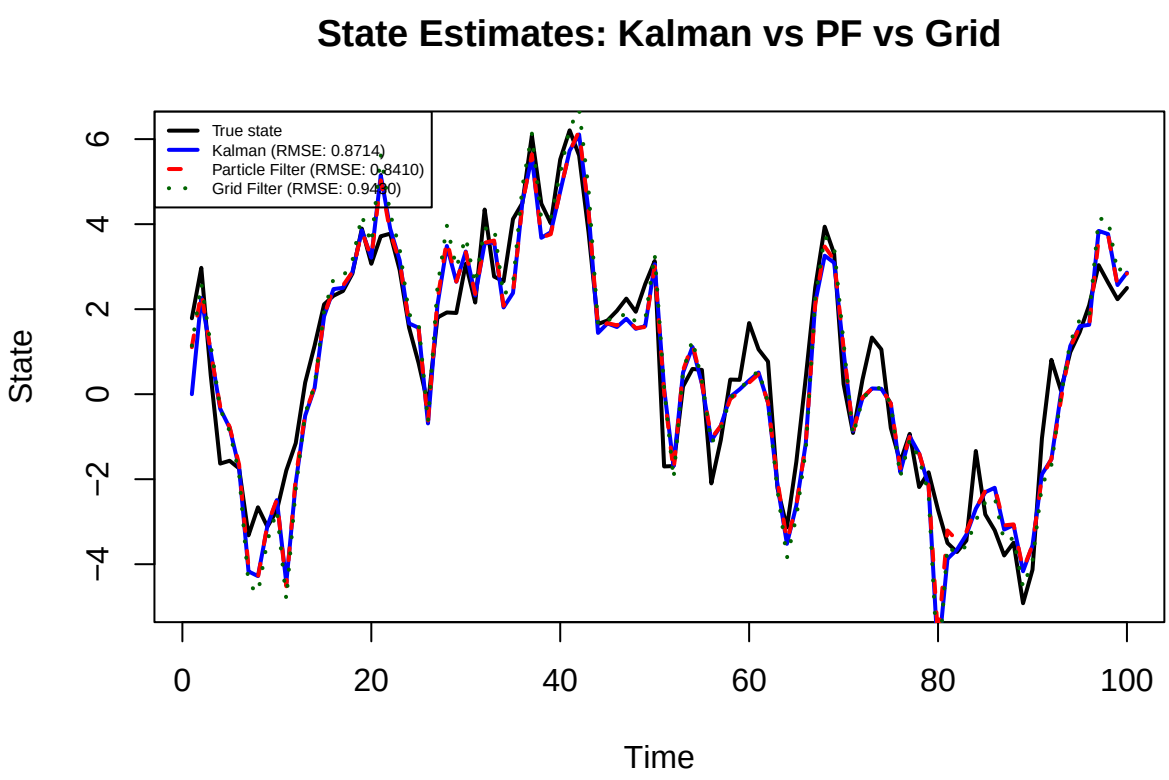
$$x_t = ax_{t-1} + \eta_t, \eta_t \sim t_{v_2}(0, \sigma_\eta^2)$$

The model produces the following plot;



REMINDER: The latent state x_t is the true underlying varuiable we care about, and it is hidden, i.e. not directly observed, the Observation y_t is the noisy measurement of the latent state, and is observed, i.e. we have this data

Next I wanted to use the Kalman filter, Bootstrap Particle filter and numerical grid approach again, but of course we had to update these models with the new state space required.



This produced a plot, very similar to taht of what we have previously seen, and really was not all that interesting to us, despite its ability to show that these filters can be applied in different contexts when we simply change the distribution of the noise component.

Going into 2 Dimensions

For this, I returned to the state space where we dealt with Gaussian distributed Noise components, but went on to dealing with 2 Dimensions.

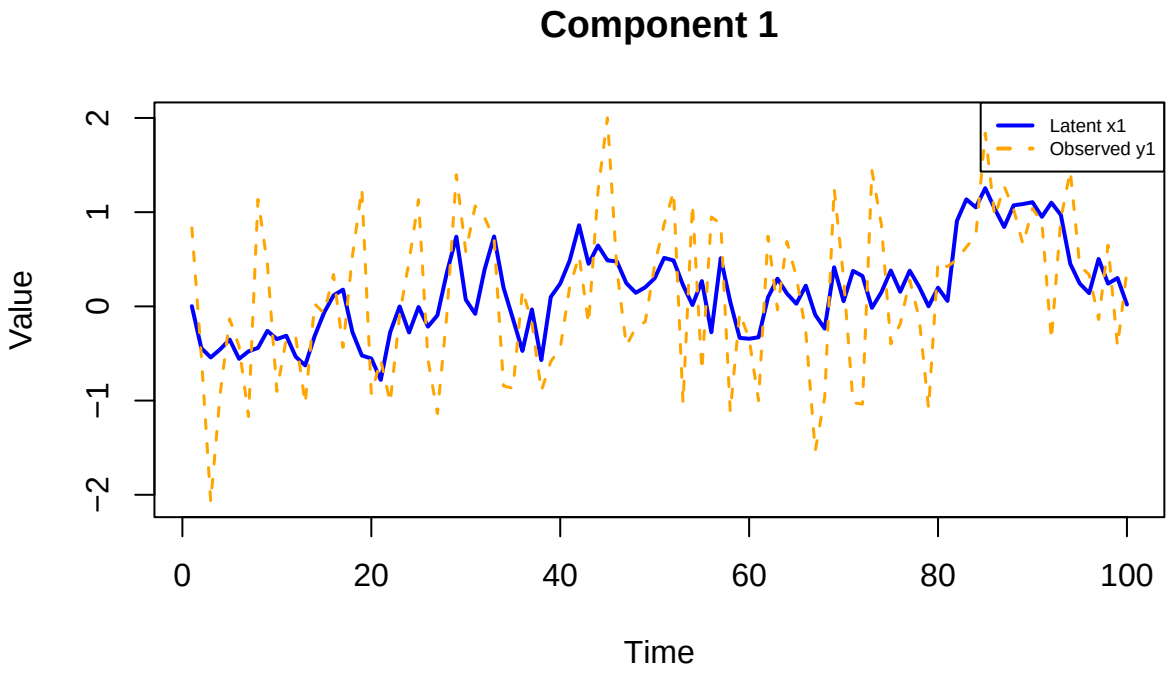
The model I constructed was defined as follows;

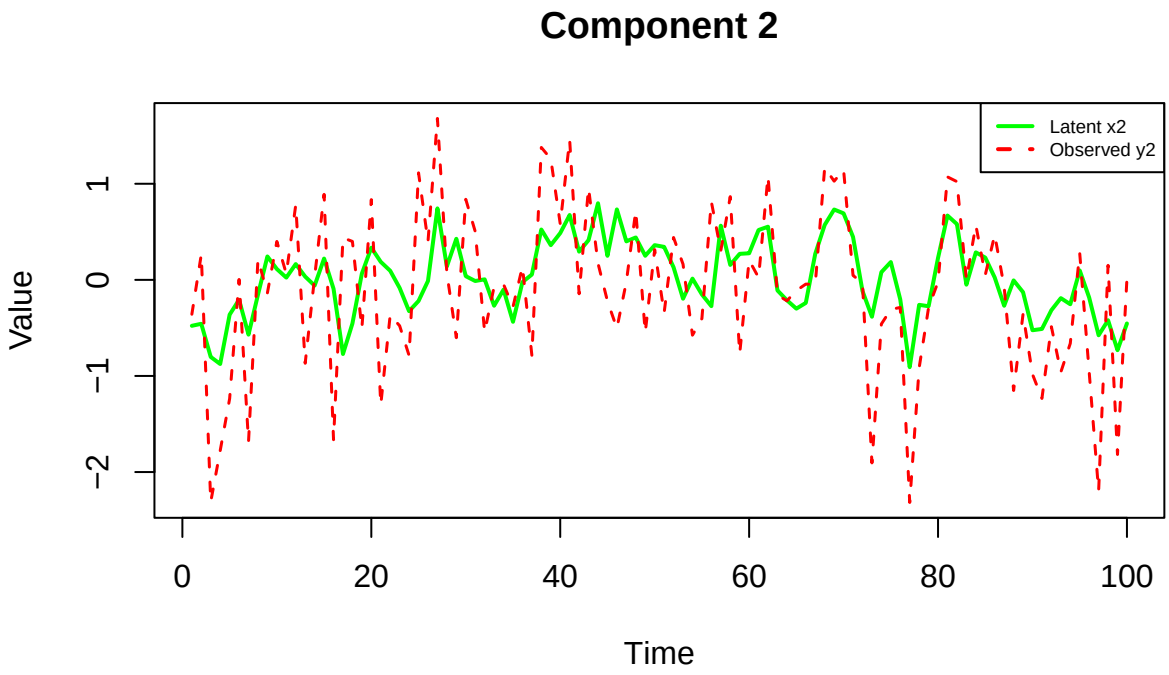
$$x_t = Ax_{t-1} + \eta_t, \eta_t \sim \mathbb{N}(0, Q)$$
$$y_t = Bx_t + \epsilon_t, \epsilon_t \sim \mathbb{N}(0, R)$$

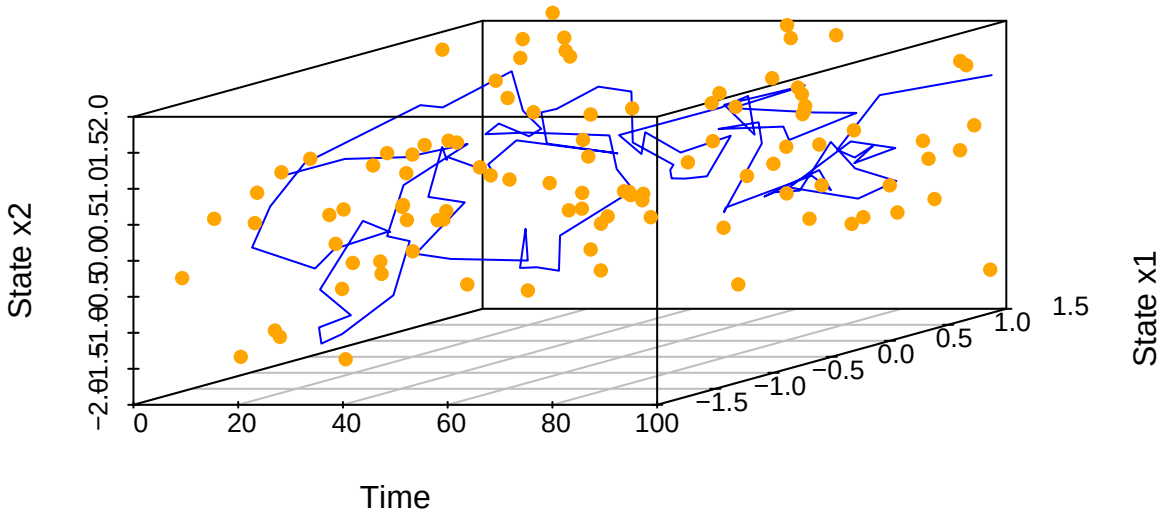
where x_t is a 2D latent state vector at time t, y_t is a 2D observation vector at time t, A is the 2×2 state transition matrix, and B is the 2×2 observation matrix, whilst Q is the 2×2 Process noise covariance matrix, and R is the 2×2 observation noise covariance matrix.

So we begin by simulating the 2D Linear Gaussian State space model and plotting the output.

I produced individual 2D plots looking at each component, as well as 3D plots combining both of the individuals states, as well as the time in which they were observed.


Component 1
Latent x1
Observed y1
Value
Time
2
1
0
−1
−2
0
20
40
60
80
100


Component 2
Latent x2
Observed y2
Value
Time
1
0
−1
−2
0
20
40
60
80
100


2D Latent State & Observations
State x2
State x1
Time
0
20
40
60
80
100
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Adding the Bootstrap Particle Filter

I now incorporate and add the bootstrap particle filter to this simulation, and this time introduce a systematic re-sampling function.

Once again, the purpose of such a particle filter, is to approximate the posterior distribution $p(x_t|y_{1:t})$, recursively using sets of particles and weights, which we constantly update. In the 2D case, the latent state $x_t \in \mathbb{R}^2$ and each particle is a 2D vector, which will make our code more complex of course, however the process generalises over from what I covered last week and builds on very naturally from 1 dimension.

We already defined our observation equation, and latent state equation earlier, but I am also going to define the initial state as $x_0 \sim \mathbb{N}(\mu_0, \Sigma_o)$.

The Process

Remember, the aim is to approximate $p(x_t|y_{1:t})$ using N particles, $\{x_t^{(i)}\}_{i=1}^N$ with equal weights, after the resampling.

Step 1: Initialisation, t = 0

Sample $x_0^{(i)} \sim \mathbb{N}(\mu_0, \Sigma_0)$ for $i = 1, ..., N$. Set weights $w_0^{(i)} = \frac{1}{N}$.

Step 2: For each time t = 1,. . . ,T

For each Particle $i = 1, .., N$, we have,

$x_t^{(i)} \sim p(x_t|x_{t-1}^{(i)}) = \mathbb{N}(Ax_{t-1}^{(i)}, Q)$

We calculate the likelihood:

$w_t^{(i)} \propto p(y_t|x_t^{(i)}) = \mathbb{N}(y_t; Bx_t^{(i),R})$

And normalise the weights,

$\tilde{w}_t^{(i)} = \frac{w_t^{(i)}}{\sum_{j=1}^N w_t^{(j)}}$.

We then resample N particles with replacement from $\{x_t^{(i)}\}$ according to weights $\tilde{w}_t^{(i)}$, of course this is done to help avoid degeneracy (i.e. one particle carrying all the weight). We let the resampled particles become the new $x_t^{(i)}$ and we reset the weights to be $\frac{1}{N}$.

We can then estimate the latent state as;

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

I then plot this in order to see how this particle filter does in this state space.

I will use the RMSE here as a measurement of error from the simulated model, which can be calculated as follows;

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

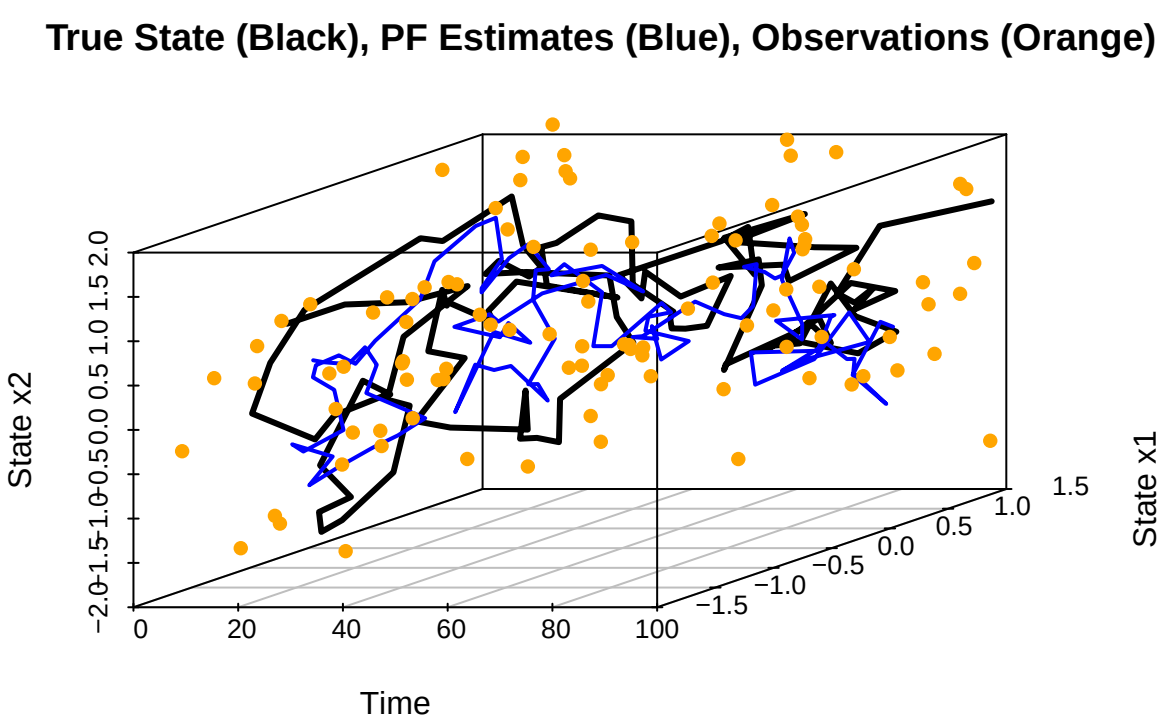


Table 1: RMSE Comparison of Filters

Filter	RMSE
Particle Filter	0.6312

Adding the Kalman Filter

I now will add the Kalman filter, and calculate the RMSE for each of the two filters in this case.

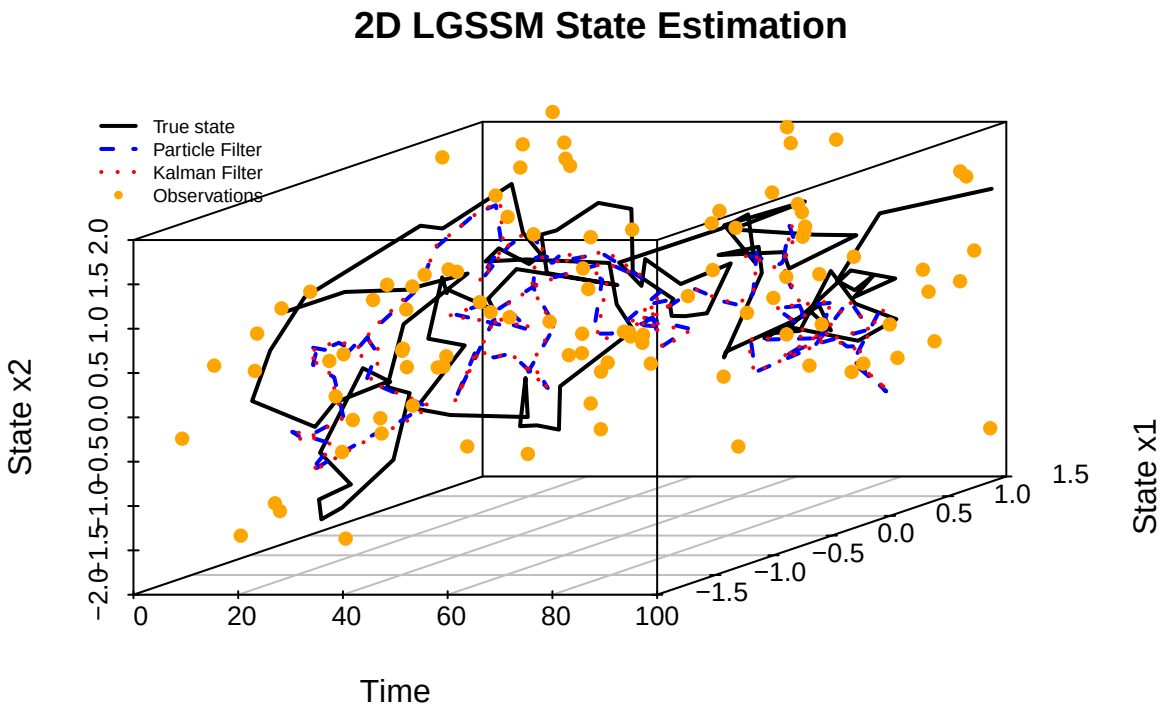


Table 2: RMSE Comparison of Filters

Filter	RMSE
Particle Filter	0.6312
Kalman Filter	0.6349

Alternatives to RMSE

RMSE generally appears to be the common means of comparison between the true latent states and the filtering methods, however as discussed with Adam in our meeting of week 1, their are alternative measures that can be used.

RMSE only captures one aspect (the average squared error) and may miss nuances in performance.

Angle Error

In the 2D cases, this measure can be quite useful, as it allows us to estimate the angular error between estimated and true vectors;

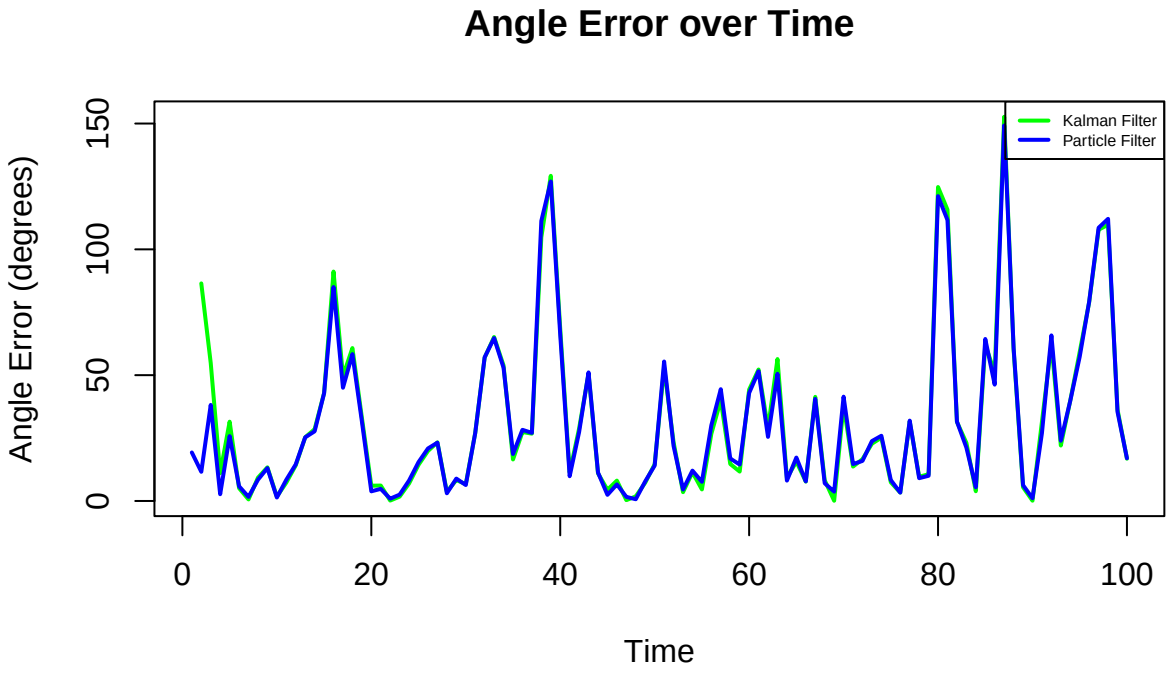
$$\theta = \arccos(\frac{\hat{x}_t^{\perp} x_t}{||\hat{x}_t||||x_t||})$$

Most filters (like the Kalman filter or particle filter) are evaluated using position-based errors like RMSE or MAE. These measure how far the estimated state is from the true state in terms of distance. However, this ignores directional accuracy, which is often just as important especially when your state includes things such as 2D velocity vectors, Displacement vectors, Force or momentum vectors.

Angle error captures whether your estimate is pointing in the right direction, regardless of how far it is in magnitude.

```
## Mean Angle Error (KF): 32.37452 degrees
```

```
## Mean Angle Error (PF): 31.09258 degrees
```



Confidence Ellipse Coverage

From what I can find, this appears to be a powerful yet often underused method for evaluating how well a filter estimates uncertainty, not just the mean state.

When the filter estimates both a mean $\hat{x}_t$ of the state at time t and a covariance matrix P_t of the uncertainty in that estimate, it gives you a full distribution, of which describes a cloud of likely positions for the true state and its confidence ellipse, which is a contour line enclosing some percentage of the probability mass.

For 2D Gaussian state spaces, the 95% confidence ellipse is defined as all the points x satisfying;

$$(x - \hat{x}_t)^{\perp} P_t^{-1} (x - \hat{x}_t) \leq \chi^2_{2,0.95}$$

We essentially ask at each time stamp, whether or not the true state falls inside the predicted 95% ellipse.

This can often be useful, as if your filter says it's 95% confident, then $\approx 95\%$ of true states should fall within those ellipses. If too few do, your filter is overconfident (underestimating uncertainty). If too many do, your filter is under confident (overestimating uncertainty).

RMSE only tells you how far off your predictions are. Coverage tells you whether you trust the confidence the filter provides. That's especially important for risk-aware decision-making, tracking, or sensor fusion.

The process:

1) Compute the **Mahalanobis distance**, $d_t^2 = (x - \hat{x}_t)^{\perp} P_t^{-1} (x - \hat{x}_t)$.
2) Check whether $d_t^2 \leq \chi^2_{2,0.95} \approx 5.991$
3) Count how many times this is true over all time stamps t and compute the percentage.

```
## 95% Confidence Ellipse Coverage: 93 %
```

Our output of 93 here tells us that our filter is slightly over confident.

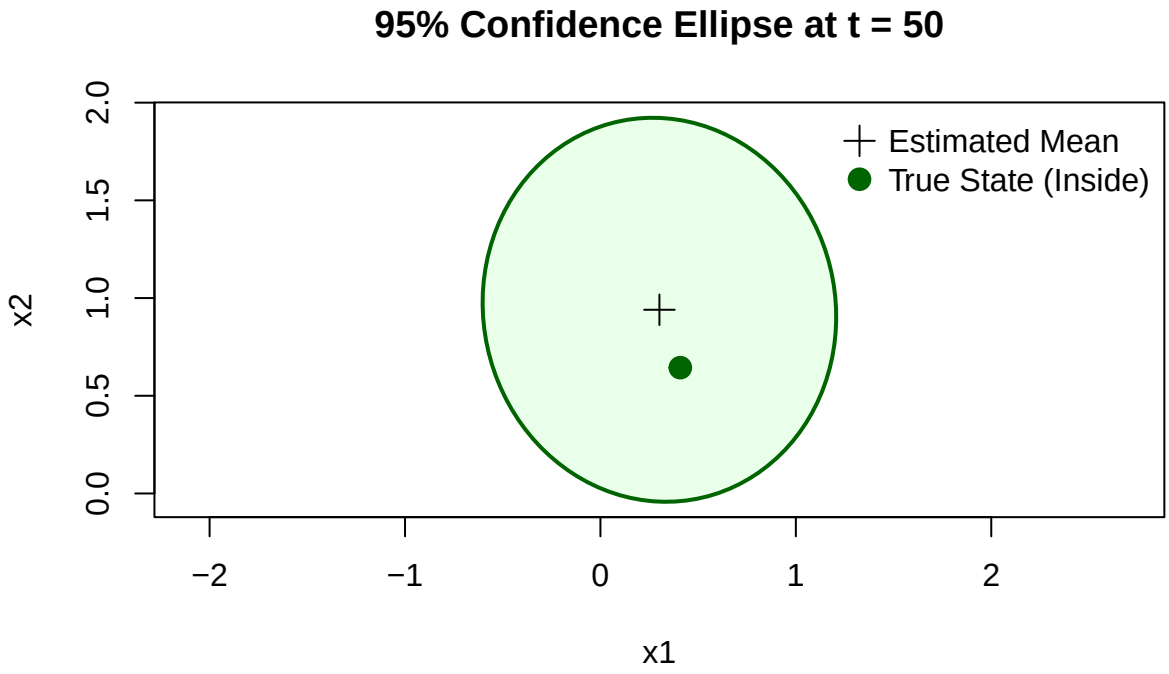
The Kalman filter predicted that its 95% confidence ellipses would contain the true state 95% of the time. But in reality, based on the simulation, they only did so 93% of the time.

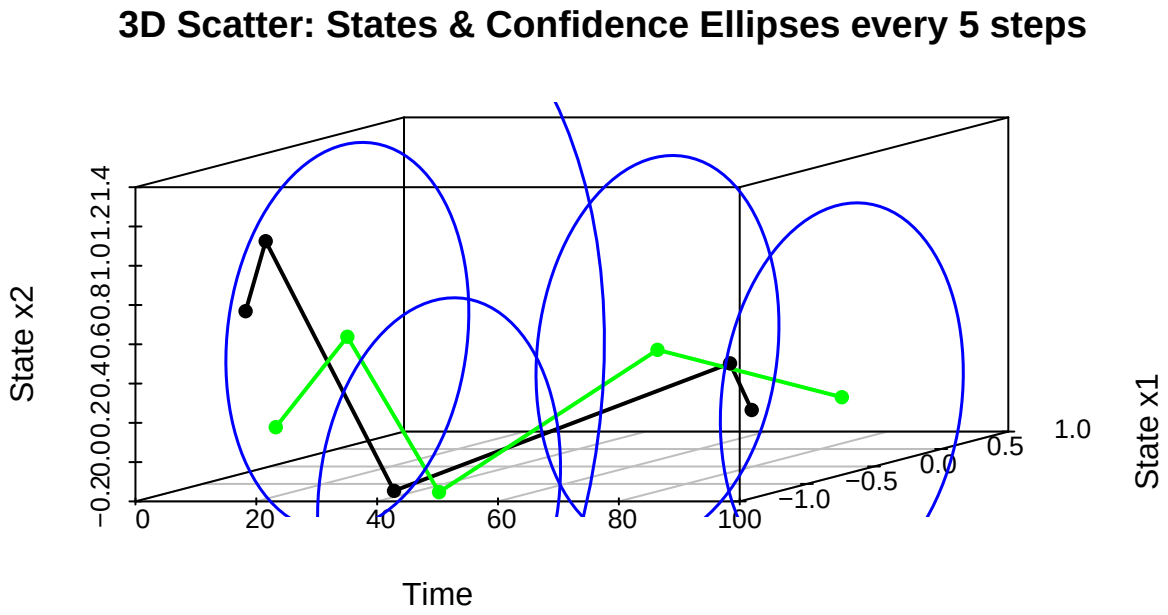
The 95% threshold is theoretical and generally over many simulations, the average coverage should be near 95%. But in any one run sampling variability might cause the observed coverage to fluctuate (e.g. 91% one time, 96% another), 93% is within a reasonable tolerance band of 95%.

For sake of understanding, I add some plots below to observe how this ellipse works at time stamp $t = 50$ and note that the observed value was actually located within it here, as well as a 3D plot displaying a few of these time stamps and how they work.

```
##
## Attaching package: 'ellipse'
```

```
## The following object is masked from 'package:graphics':
##
##     pairs
```


95% Confidence Ellipse at t = 50
Estimated Mean
True State (Inside)
x2
x1
0.0
0.5
1.0
1.5
2.0
−2
−1
0
1
2


3D Scatter: States & Confidence Ellipses every 5 steps
State x2
State x1
Time
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
20
40
60
80
100
−1.0
−0.5
0.0
0.5
1.0

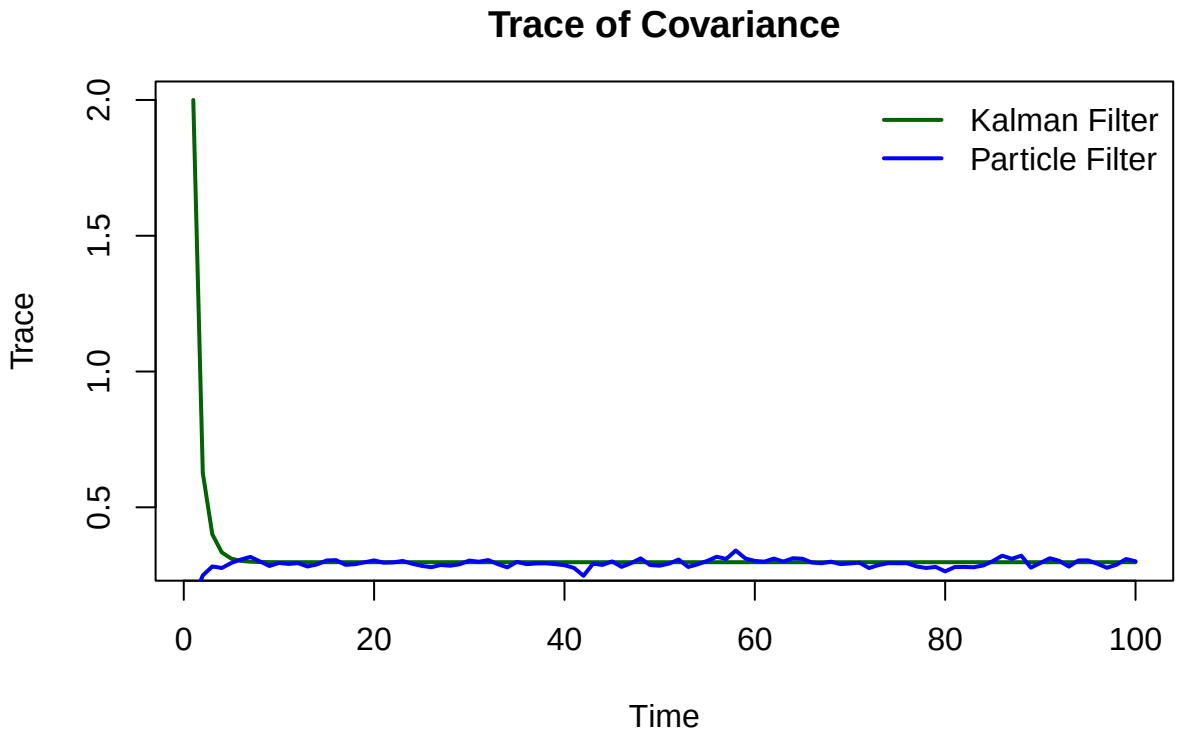
Covariance Determinant or Trace

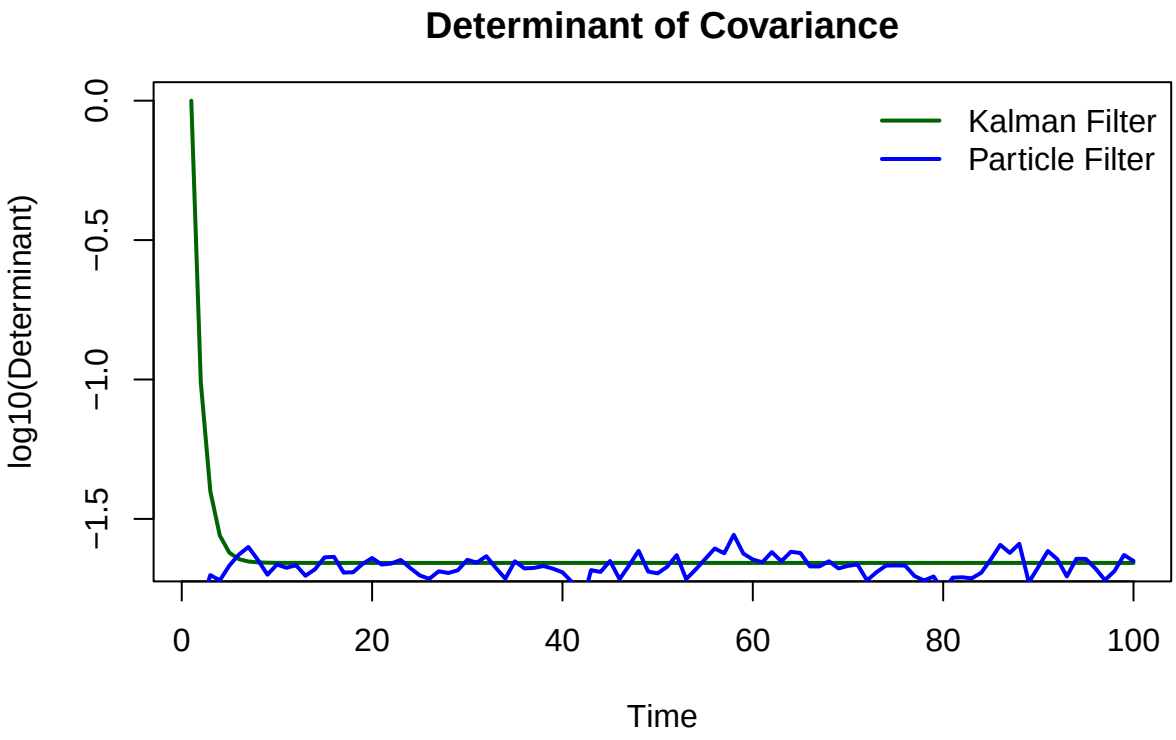
In state estimation, the covariance matrix P represents uncertainty in our estimate. It evolves over time as the filter updates its beliefs using process models and measurements.

Key Measures:

The trace $tr(P)$ represents the total variance across all state dimensions, the higher the trace, the more uncertainty there is. The determinant $|P|$ represents the volume of the uncertainty ellipsoid in state space. It accounts for correlations between variables. A very small determinant implies the filter is overconfident.

A filter with low trace/determinant typically indicates more confidence in its state estimate, but confidence $\neq$ correctness, as a too small covariance can be a red flag for overconfidence and divergence risk.




Determinant of Covariance
log10(Determinant)
0.0
−0.5
−1.0
−1.5
0
20
40
60
80
100
Time
Kalman Filter
Particle Filter