Exploring the Univariate Linear gaussian state space model

Max Richards

2025-07-17

Setting up the model

Consider the univariate linear model of the form;

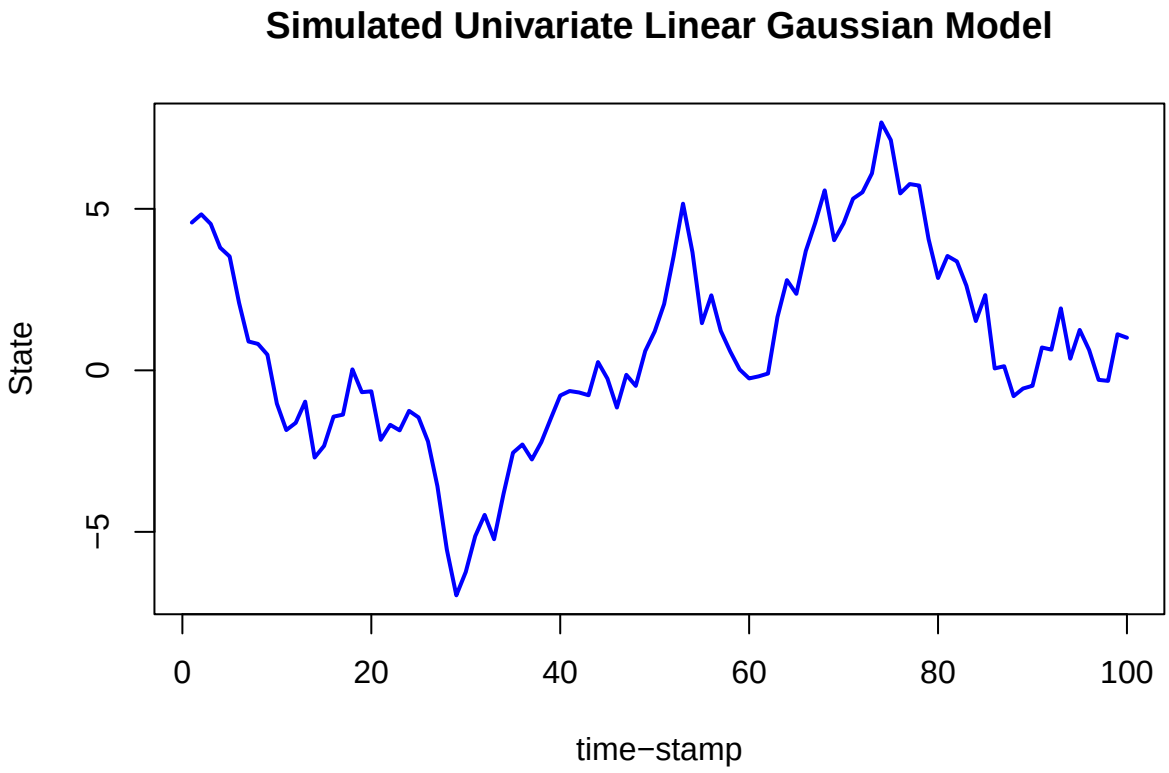
$$y_t = bx_t + \epsilon_t, \epsilon_t \sim N(0, r)$$

Let us take the state equation;

$$x_t = ax_{t-1} + \eta_t, \eta_t \sim N(0, q)$$

We simulate this model below, using $r = 1$, $q = 1$, $a = 0.9$, $b = 1$ and $T = 100$ where T denotes the number of time stamps in our particles.

We can then plot this so we can visualise our model.



Implementing the Particle Filter strategy

We now want to apply the boot-strap particle filter method to our simulated linear gaussian state space model. This is one of the most fundamental particle filtering algorithms for approximating the posterior distribution ($p(x_t|y_{1:t})$) over hidden states in state-space models.

We will first divert to an overview of how this strategy works.

We already have stablished our state space model (the state equation seen above), and we have that $x_t \sim p(x_t|x_{t-1})$, as well as $y_t \sim p(y_t|x_t)$.

We simulate a set of N different particles, $\{x_t^{(i)}\}_{i=1}^N$, and assign a weight, $w_t^{(i)}$, to each of these particles.

We have seen in literature and outside sources that we can now formulate a Monte Carlo approximation to the posterior. This expression will be formulated as follows;

$$\hat{p}(x_t|y_{1:t}) = \sum_{i=1}^{N} w_t^{(i)} \delta(x_t - x_t^{(i)})$$

Where δ denotes the **Dirac Delta Function** [1] .

Let us assume we have N particles of length $t-1$ and their respective weights. We can then begin the Step-by-Step algorithm that follows for each new time stamp t.

1) Resample:

We draw N new particles by sampling with replacement from our original N particles, with probabilities proportional to their weights. This results in a new equally-weighted population, and is done to remove degeneracy.

2) Propagate:

For each resampled particle we have $x_t^{(i)} \sim p(x_t|x_{t-1}^{(i)})$, and this evolves the state forwards, using the system dynamics.

3) Weights:

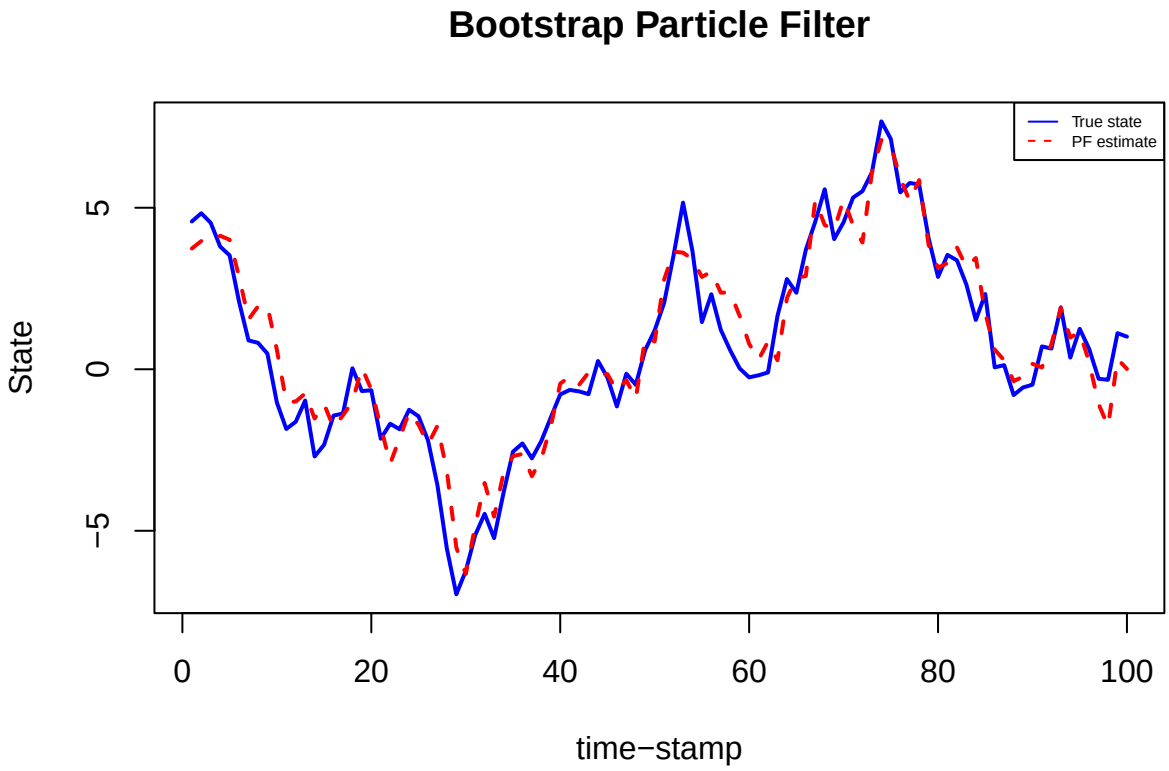
We can compute the updated weights at the new time stamp, by using the following relationship, $w_t^{(i)} \propto p(y_t|x_t^{(i)})$.

We can normalise, and now observe,

$$w_t^{(i)} = \frac{w_t^{(i)}}{\sum_{j=1}^{N} w_t^{(j)}}$$

And we result with an approximate posterior $\{x_t^{(i)}, w_t^{(i)}\}$.

We can now compare the results of the filtering method with the true state we obtained from how we defined our linear gaussian state space model earlier on.


Bootstrap Particle Filter
True state
PF estimate
State
5
0
−5
0
20
40
60
80
100
time−stamp

Applying the Kalman Filter

From the meeting with Adam on 15/07/2025, I was advised to compare how the results of implementing such a filtering method on a model defined as above would compare against the Kalman filter.

In the ideal setting, the results of the two estimations should match up perfectly, given the nature of the state space we are working on.

We now, once again divert our attention, this time to an overview of how this strategy works.

The required state model and observed equation are staisfied by what we defined at the beginning of this text.

Once again, the goal here is to compute the posterior distribution $p(x_t|y_{1:t})$, i.e. the best estimate of the current hidden state given all past data. This is essentially done in two key steps, the prediction step (from $t-1$ to t), and the updating step (condition on y_t.

Goal: At each time stamp t, given all observations up to $y_{1:t}$, we want to estimate the mean $\hat{x}_t = \mathbb{E}[x_t|y_{1:t}]$ and the variance $P_t = Var(x_t|y_{1:t})$.

Prediction step:

Predict the next state, $\hat{x}_{t|t-1} = a\hat{x}_t - 1$.

Predict the variance/uncertainty, $P_{t|t-1} = a^2 P_{t-1} + q$.

Note: Variance increases due to the additional process noise after applying the filtering process at each new time stamp.

Updating step:

We now consider the new observation $y_t = bx_t + \epsilon_t$, which was characterised by our setup earlier.

We compute what is called the *Kalman Gain*, K_t, which essentially tells us how much we trust the observation, relative to our prior prediction:

$$K_t = \frac{P_{t|t-1} \times b}{b^2 P_{t|t-1} + r}$$

We then update our estimate following;

$$\hat{x}_t = \hat{x}_{t|t-1} + K_t(y_t - b\hat{x}_{t|t-1})$$

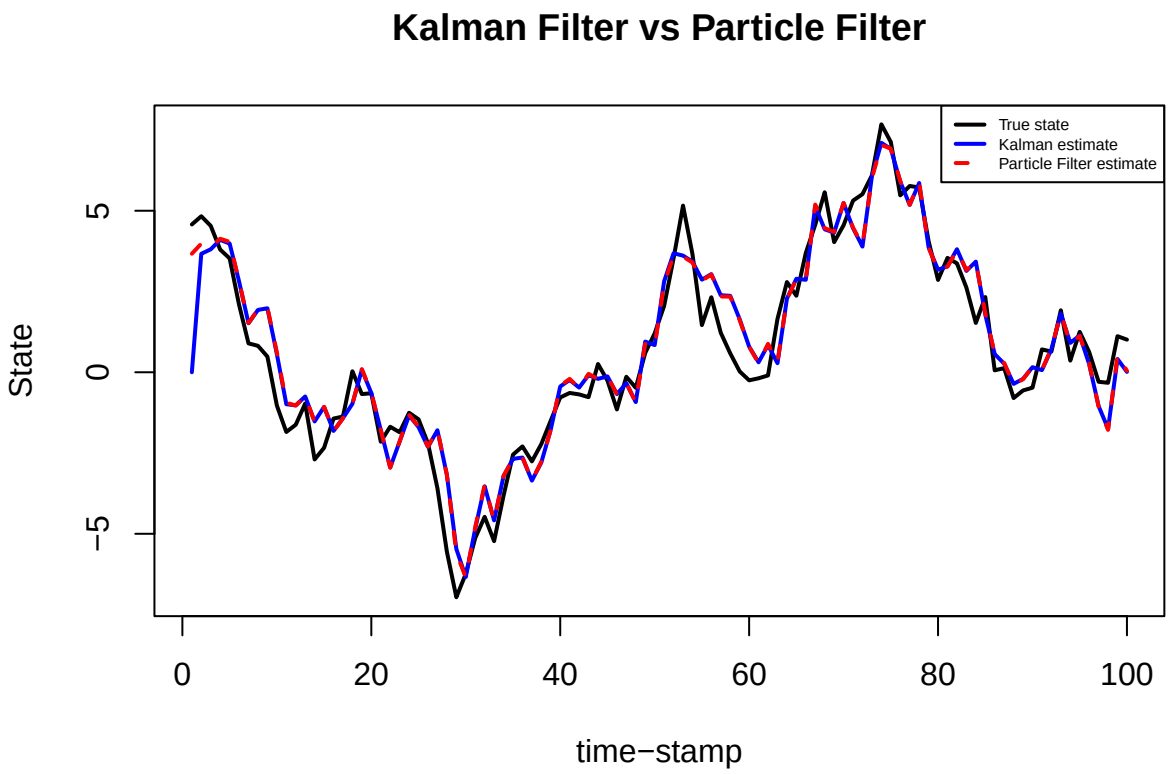
Aside for mental note: Think of this step as New estimate = prediction + adjustment × (measurement error)

And update our uncertainty/variance following;

$$P_t = (1 - K_t b)P_{t|t-1}$$

Repeating this over some fixed number of time stamps performs the Kalman Filter procedure.

We now plot the linear Gaussian state space model we constructed, as well as the estimated trajectories from the bootstrap particle filter, and the Kalman filter to compare how all 3 appear.



RMSE's were calculated for the two methods for comparison between methods.

```
## Kalman Filter RMSE: 0.9398
```

```
## Particle Filter RMSE: 0.8194
```

Surprisingly the Kalman filter apparently performs ever-so-slightly worse than the bootstrap filter, which was a surprise as we know the Kalman filter to be the optimal method which gives us the exact posterior mean.

This is likely due to the slight variation between the two methods at the early time-stamps. This is because the particle filters are stochastic, and so if we run it once and compare RMSE, it could in theory randomly perform better than KF by chance, which has most likely occurred in this case here.

[1] - https://en.wikipedia.org/wiki/Dirac_delta_function

Reflections and Improvements

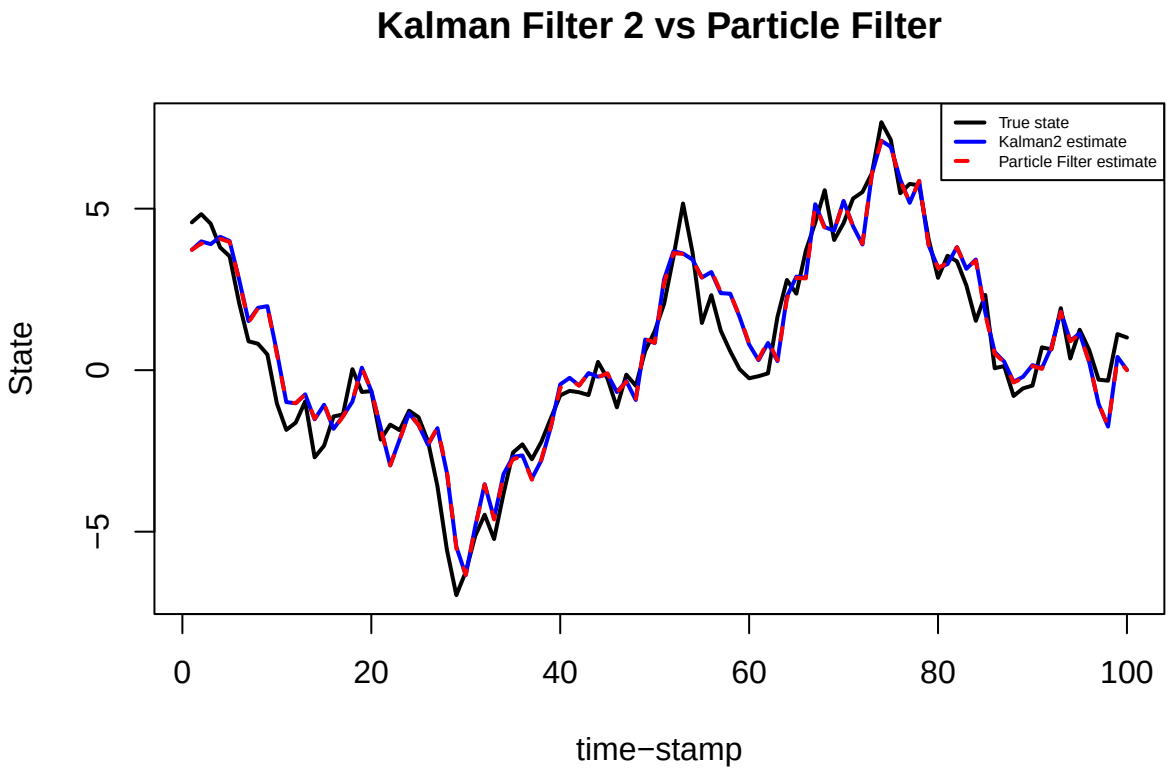
After initial feedback from Adam, it became clear that the reason for the difference between the two methods was due to the fact that the particle filter incorporates the first observation y_1 immediately by weighting the initial particles, while the Kalman filter was only initialising the prior at $t = 1$ but does not update using our observation y_1.

In other words, the bootsrap particle filter's estimate at time 1 is already a posterior mean, $\mathbb{E}[x_1|y_1]$ because it weights particles by $p(y_1|x_1)$, while the Kalman filter estimate at time $t = 1$ is just the prior mean, $\mathbb{E}[x_1] = 0$, disregarding the information we actually observed from our defined uni variate linear model, y_1.

If the actual initial latent state was far from zero (due to the simulation draw), this causes the initial estimate discrepancy I observed.

Hence I adjusted code to now account for this by adding an initial measurement step after setting the prior for the Kalman filter function.

And so our new plot looks as follows;



RMSE's were calculated for the two methods for comparison between methods.

```
## Kalman Filter 2 RMSE: 0.8208
```

```
## Particle Filter RMSE: 0.8205
```

Still some variation due to the sources of uncertainty. The 2 main sources of randomness to account for include:

1) The observation sequence itself
2) The fundamental stochasticity of the Algorithm

Implementing a Numerical Integration Approach

The next step from this point was to implement the numerical integration approach (i.e. grid-based filtering), and then compare this to our Kalman and Bootstrap particle filter.

We note that the Kalman filter gives the exact filtering distribution for our linear Gaussian model, whilst the numerical integration approach approximates the filtering distribution via discretised integration over a grid, and the bootstrap particle filter uses a stochastic approximation, so we essentially make a comparison of deterministic approximation (grid method) vs Stochastic approximation (bootstrap particle filter) vs Exact solution (Kalman Filter).

How the numerical Integration approach works:

1) Predicts the prior distribution at time t, by integrating over the transition model
2) Updates the prior using the likelihood of the observation y_t to get the posterior
3) Normalise and compute expectations numerically

Steps of Implementation:

1) Create a grid over possible state values x, in our case we did the space from -10 to 10 with $1000 = K$ points.
2) Initialise prior over the latent state using the stationary distribution of $x_t \sim \mathbb{N}(0, \frac{q}{1-a^2})$. So, $p(x_1) \approx [p(x_1 = x_1), ..., p(x_1 = x_K)]$.
3) Loop over time; Predict - Compute $p(x_t|y_{1:t-1})$, Update - Multiply by the likelihood $p(y_t|x_t)$ then normalise, Estimate - Estimate the filtered mean, $\sum x_i \times p(x_i|y_{1:t})$

More in-depth breakdown of step 3):

At time $t = 1$ we use Bayes' rule: $p(x_1|y_1) \propto p(y_1|x_1) \times p(x_1)$. Each value of x_1 on the grid is re-weighted by its likelihood under the observation model: $y_1 = bx_1 + \epsilon, \epsilon \sim \mathbb{N}(0, r)$. This gives us the posterior distribution over x_1, and we now compute $\hat{x}_1 = \sum_{i=1}^{K} x_i \times p(x_1 = x_i|y_1)$.

Now for times t= 2, ... , T, we apply the transition dynamics $p(x_t|y_{1:t-1}) = integral(p(x_t|x_{t-1}) \times p(x_{t-1}|y_{1:t-1}))dx_{t-1}$

which is ultimately a convolution from the previous step $p(x_{t-1}|y_{1:t-1})$, and the transition kernel $p(x_t|x_{t-1}) = \mathbb{N}(x_t; ax_{t-1}, q)$. This is done now by building a matrix of transition probabilities, and applying matrix multiplication to get the predicted prior. Repeating of this process gives us the predicted prior over x_t. Once we have this prior, like when we dealt with the $t = 1$ case, we apply bayes' rule and so here we have $p(x_t|y_{1:t}) \propto p(y_t|x_t) \times p(x_t|y_{1:t-1})$, and we perform $\hat{x}_t = \sum_{i=1}^{K} x_i \times p(x_t = x_i|y_{1:t})$ at each step to give us the point estimate (the posterior mean).

Now plot all 3 together for a comparison under this model with the defined state space.