MFC CDT Probability and Statistics Coursework

Leo Collins

December 31, 2024

1 Model

The model will be a two-dimensional linear Gaussian state space model

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{w}_t \tag{1}$$

where $\mathbf{x}_t = (x_{0,t}, x_{1,t})^T$ is the state vector at time t,

$$\mathbf{A} = \begin{pmatrix} 1.0 & 0.1 \\ 0.0 & 1.0 \end{pmatrix}$$

is the state transition matrix, and $\mathbf{w}_t \sim \mathcal{N}(0, \mathbf{Q})$ is the process noise at time t with covariance matrix $\mathbf{Q}$. This model could represent an object moving randomly with position $x_{0,t}$ and velocity $x_{1,t}$. The position is then updated with timestep 0.1, with noise. At each time t the state is observed according to

$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t + \mathbf{v}_t \tag{2}$$

where $\mathbf{y}_t \in \mathbb{R}^2$ is the observation vector at time t, $\mathbf{C} = \mathbf{I}$ is the observation matrix, and $\mathbf{v}_t \sim \mathcal{N}(0, \mathbf{R})$ is the observation noise at time t with covariance matrix $\mathbf{R}$. The initial state $\mathbf{x}_0$ is drawn from a Gaussian distribution with mean μ_0. Figure 1 shows some simulated data from the model. The blue dashed line shows the true state of the model $\mathbf{x}_t$, and the yellow dots show the observations $\mathbf{y}_t$.

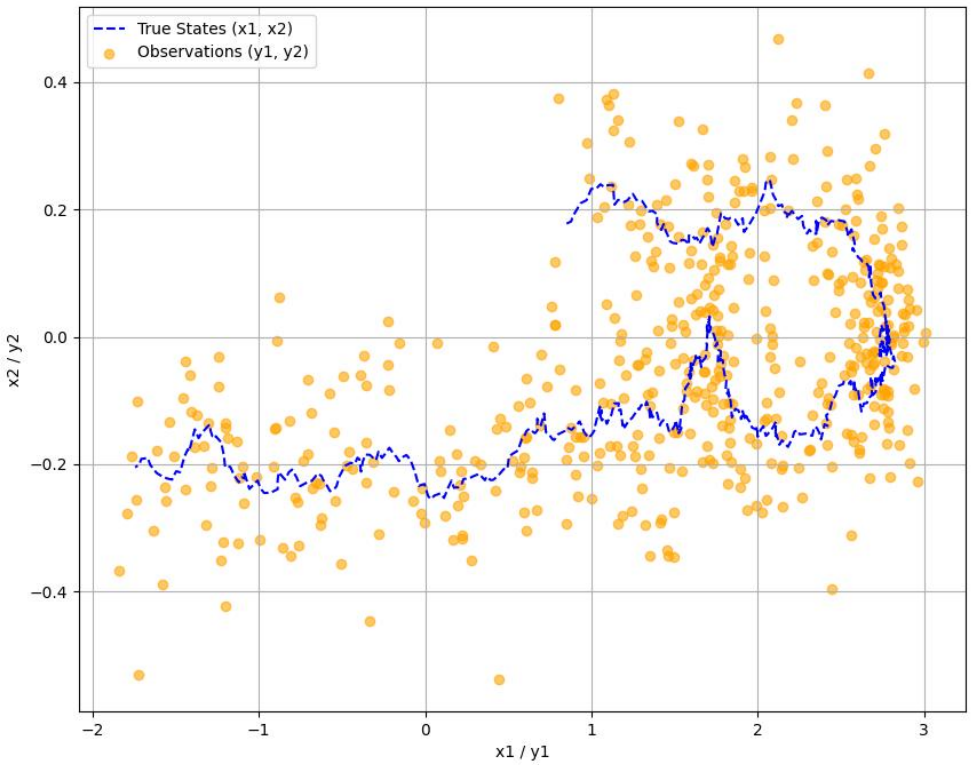


Figure 1: Simulated data from the model.

2 Kalman Filter

The Kalman filter is a recursive algorithm that estimates the state $\hat{\mathbf{x}}_k$ and covariance $\mathbf{P}_k$ of a linear Gaussian state space model. It assumes that the state and observation noise are Gaussian, and is optimal in this case in the sense that it minimises the mean squared error of the state estimate $\hat{\mathbf{x}}_k$.

The implementation of the Kalman filter is shown in Listing 1. The log marginal likelihood $l = \log p(\mathbf{y})$ is calculated using the update rule [1]

$$l_k = l_{k-1} - \frac{1}{2}\left(\log|\mathbf{S}_k| + \tilde{\mathbf{y}}_k^T \mathbf{S}_k^{-1} \tilde{\mathbf{y}} + 2\log(2\pi)\right), \tag{3}$$

where $\tilde{\mathbf{y}}_k = \mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_{k|k-1}$ is the innovation, and $\mathbf{S}_k = \mathbf{C}\mathbf{P}_{k|k-1}\mathbf{C}^T + \mathbf{R}$ is the innovation covariance. With the parameters found in the notebook, the log marginal likelihood is approximately 832.224.

Figure 2 shows the Kalman filter applied to the simulated data. The Kalman filter manages to estimate the true state of the model $\mathbf{x}_t$ well, despite the large observational noise.

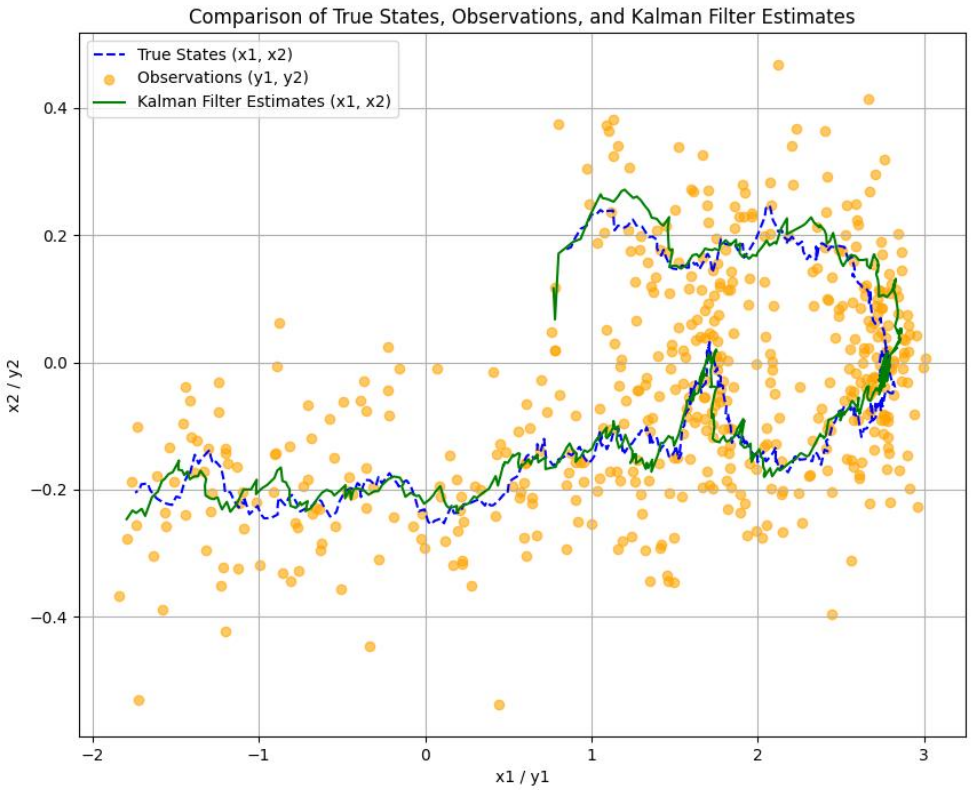


Figure 2: Kalman filter (green) applied to the simulated data, compared to the true state (dashed blue) and observations (yellow).

3 Particle Filter

The particle filter is a Monte Carlo algorithm that approximates the state of a state space model. In contrast to the Kalman filter, the particle filter can handle non-linear and non-Gaussian models. Figure 3 shows the particle filter with 100 particles applied to the simulated data. The particle filter manages to estimate the true state of the model fairly accurately, despite the limited number of particles used.

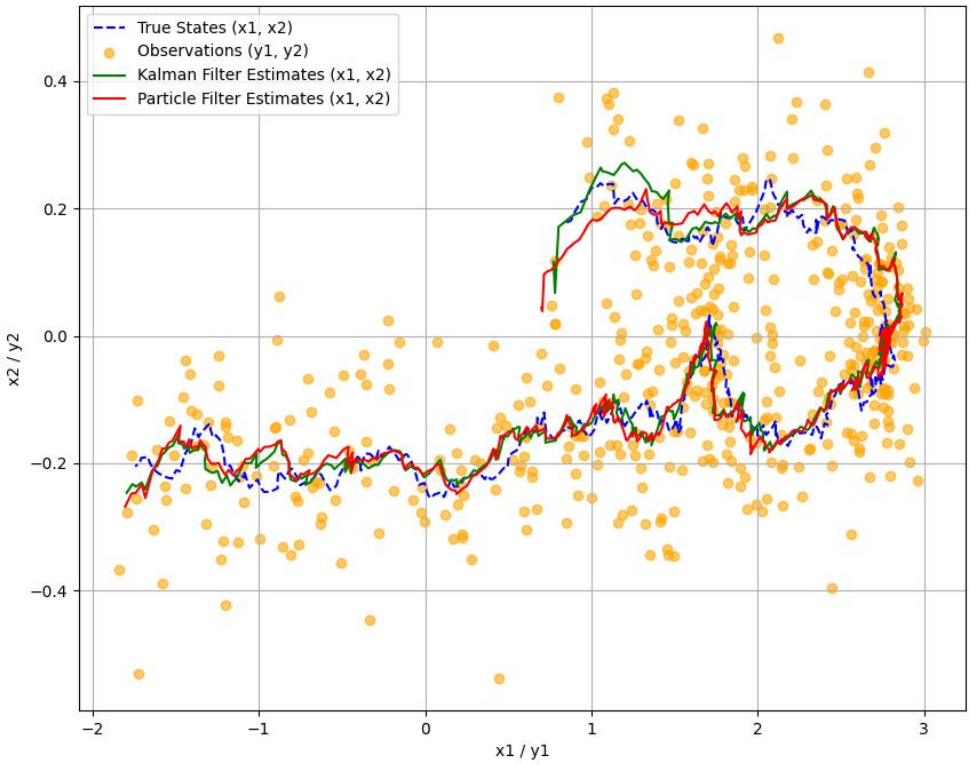


Figure 3: Particle filter (red) with 100 particles applied to the simulated data, compared to the true state (dashed blue), the Kalman filter (green), and observations (yellow).

If we increase the number of particles used, we can improve the accuracy of the particle filter. Figure 4 shows the particle filter with 1000 particles applied to the

simulated data. The particle filter estimate is now much closer to the Kalman filter estimate.

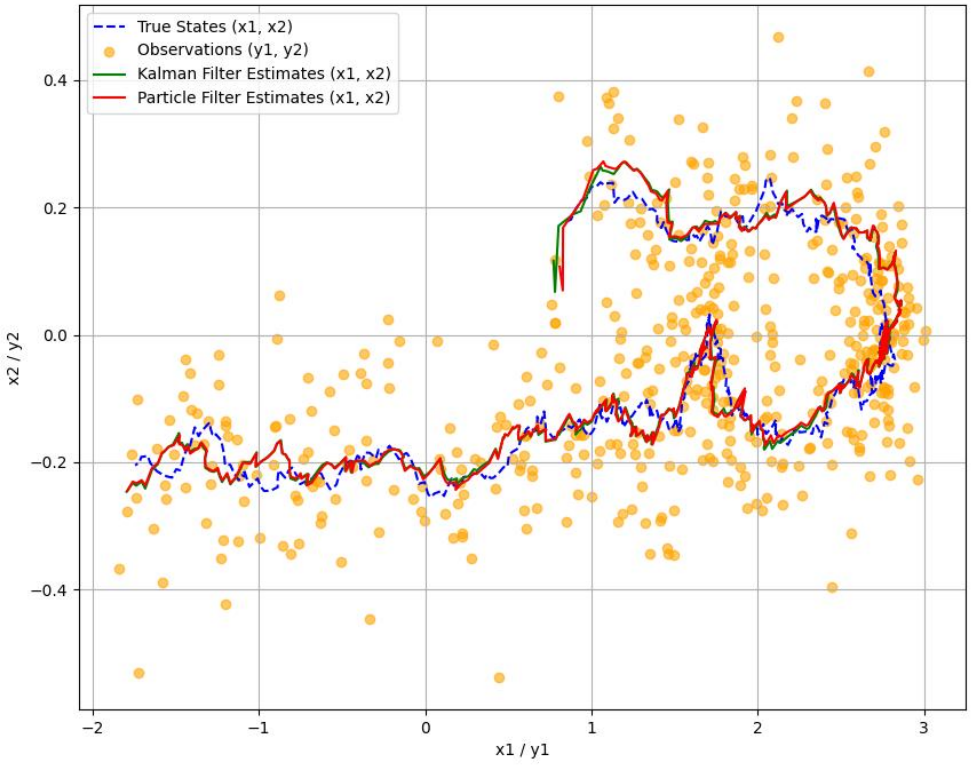


Figure 4: Particle filter (red) with 1000 particles applied to the simulated data, compared to the true state (dashed blue), the Kalman filter (green), and observations (yellow).

Figure 5 shows box plots of the log marginal likelihood for differing numbers of particles. We can see that the mean converges to the log marginal likelihood given by the Kalman filter, and the variance decreases as the number of particles increases.

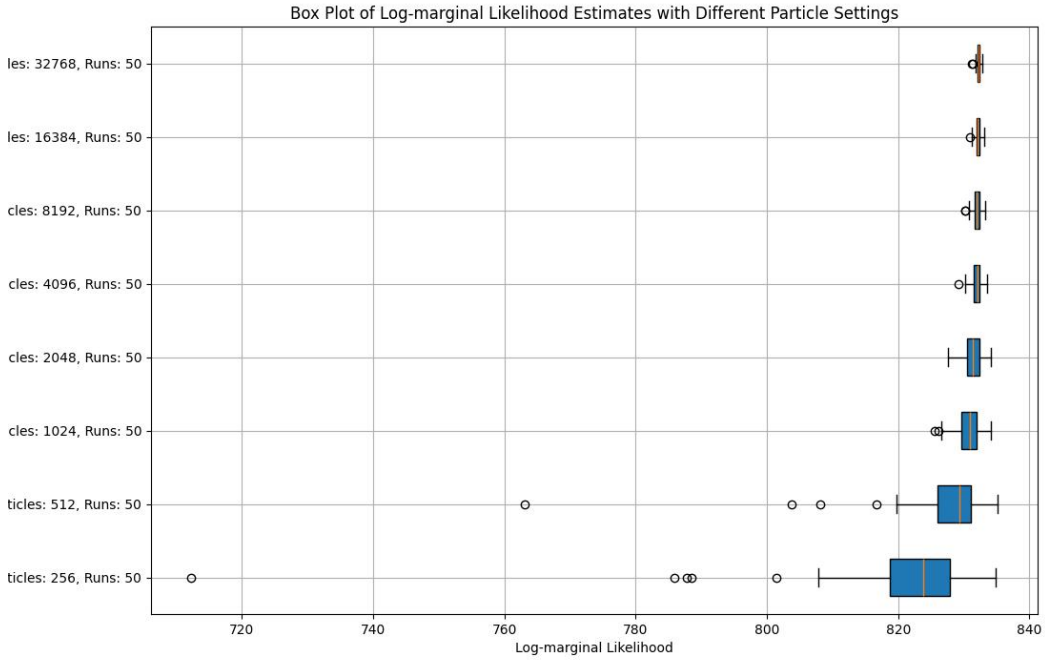


Figure 5: Box plots of log marginal likelihood for differing numbers of particles.

4 Convergence

Figure 6 shows how the mean log marginal likelihood of 50 runs of the particle filter converges to the log marginal likelihood of the Kalman filter. This is expected as the Kalman filter gives the optimal estimate for a linear Gaussian model, and the particle filter is an unbiased estimator of the true state.

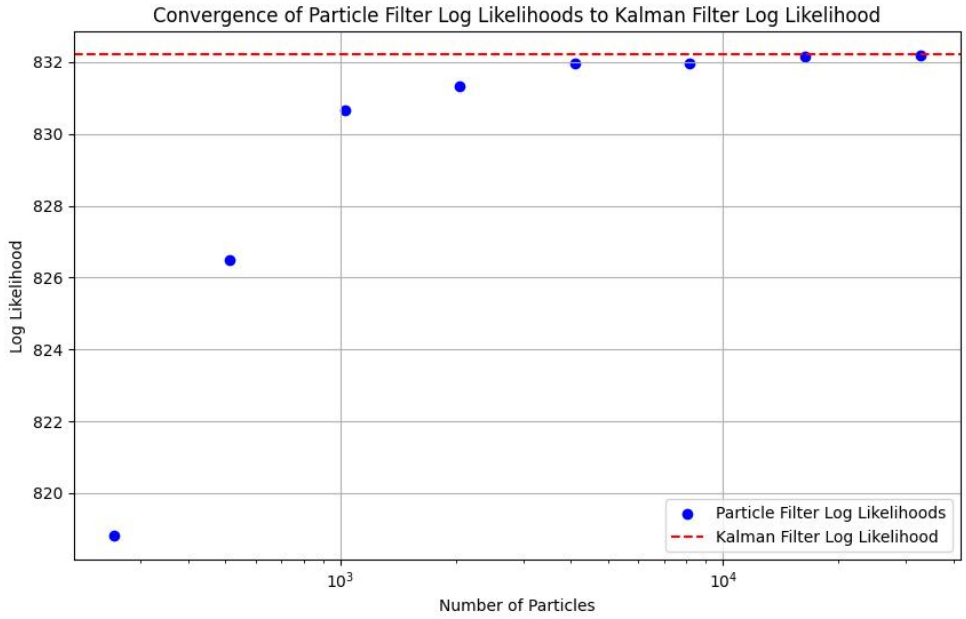
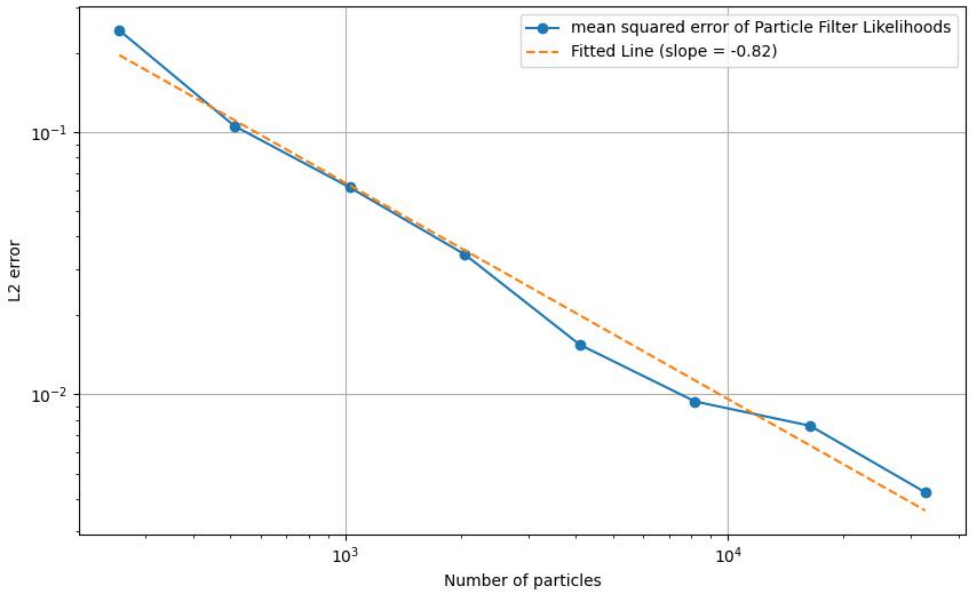


Figure 6: Mean log marginal likelihood of the particle filter estimates converges to the Kalman filter log marginal likelihood as the number of particles increases. 50 runs were used for each number of particles.

Figure 4 shows a log-log plot of the L^2 error of the mean particle filter estimate comapred to the Kalman filter esimate. The order of convergence is approximately 0.82.


mean squared error of Particle Filter Likelihoods
Fitted Line (slope = -0.82)
L2 error
10^{-1}
10^{-2}
10^{3}
10^{4}
Number of particles

References

[1] Helmut Lütkepohl. *Introduction to multiple time series analysis*. Springer Science & Business Media, 2013.

A Code

```python
import numpy as np

def kalman_filter(y, A, Q, C, R, mu_0, Sigma_0):
    T = y.shape[0]
    dim_x = A.shape[0]
    mu = np.zeros((T, dim_x))
    Sigma = np.zeros((T, dim_x, dim_x))
    log_likelihood = 0

    # Initial values
    mu_t = mu_0
    Sigma_t = Sigma_0

    for t in range(T):
        # Prediction step
        mu_pred = A @ mu_t
        Sigma_pred = A @ Sigma_t @ A.T + Q

        # Update step
        y_diff = y[t] - C @ mu_pred  # Innovation
        S = C @ Sigma_pred @ C.T + R  # Innovation covariance
        K = Sigma_pred @ C.T @ np.linalg.inv(S)  # Kalman gain
        mu_t = mu_pred + K @ y_diff  # State estimate
        Sigma_t = Sigma_pred - K @ C @ Sigma_pred  # State covariance

        # Compute log-likelihood
        log_likelihood += -0.5 * (
            np.log(np.linalg.det(S)) + y_diff.T @ np.linalg.inv(S) @ y_diff + 2 * np.log(2 * np.pi)
        )

        mu[t] = mu_t
        Sigma[t] = Sigma_t

    return mu, Sigma, log_likelihood
```

Listing 1: Kalman filter implementation.

```
def particle_filter(y, A, Q, C, R, mu_0, Sigma_0, N):
T = y.shape[0]
particles = np.random.multivariate_normal(mu_0, Sigma_0, size=N)
weights = np.ones(N) / N

x_est = np.zeros((T, 2))
log_likelihood = 0

for t in range(T):
    # Evolve particles according to state transition model
    particles_pred = (A @ particles.T).T + np.random.multivariate_normal(np.zeros(2), Q, size=N)

    # Compute weights based on observation likelihood
    y_diff = y[t] - np.dot(particles_pred, C.T)
    likelihoods = np.exp(-0.5 * np.sum(y_diff @ np.linalg.inv(R) * y_diff, axis=1))
    likelihoods /= np.sqrt((2 * np.pi)**2 * np.linalg.det(R))

    weights *= likelihoods
    weights /= np.sum(weights)  # normalise weights

    log_likelihood += np.log(np.sum(likelihoods) / N)

    # Estimate state
    x_est[t] = np.average(particles_pred, axis=0, weights=weights)

    # Resample particles
    particles = particles_pred[np.random.choice(N, N, p=weights)]
    weights = np.ones(N) / N

return x_est, log_likelihood
```

Listing 2: Particle filter implementation.