Kalman Filters and Extensions

Authors: Matvei Kreinin, Maria Nikitina, Anastasia Voznyuk, Petr Babkin
Course: Bayesian Multimodelling 2024 - 2025

1 Bringing Kalman Filters to Deep Learning

Despite their foundational role in probabilistic state-space modelling, Kalman filters [3] remain surprisingly underrepresented in the deep learning community. Our new framework "**Kalman filter and his friends**" aims to change that.

1.1 Why Should DL Researchers Care about Kalman Filters?

Kalman filters serve as the backbone for many advanced models, including:

- Gaussian Processes, used for regression and time-series analysis.
- State-Space Models (SSMs) like S4/S6, crucial for sequence modelling in modern deep learning.
- Hidden Markov Models (HMMs), widely applied in speech recognition and bioinformatics.
- Recurrent Neural Networks (RNNs), which share conceptual similarities in sequential data processing.

Kalman filters provide a principled way to estimate hidden states from noisy observations, something neural networks often do implicitly but with less interpretability. Kalman filter is extremely

powereful when the noise in the data is roughly Gaussian, and a lot of tasks in DL fit into this description. By integrating these filters into DL pipelines, we gain:

- Stronger theoretical grounding – Kalman-based approaches have well-defined probabilistic properties.
- Better uncertainty quantification – Unlike many DL models, Kalman filters explicitly model uncertainty.
- Efficiency interpretability – Compared to deep neural networks, Kalman filters can be computationally efficient and easier to analyze.

2 Theoretical Background

We will be working with a discrete-time linear dynamic system. It consists of two parts - a state equation and an observation equation. We have state variable x_t and our observed variable y_t. We try to predict y_t using x_t. We assume there is a linear relationship between these two variables and that there is Gaussian noise in "measurements" of both of these variables:

$$x_{t+1} = A_t x_t + w_t \tag{1}$$

$$y_t = H_t x_t + v_t \tag{2}$$

where

- x_t is the hidden state at time t,
- A_t is the transition matrix,
- y_t is the observation,
- H_t is the observation matrix

The stochastic nature of the process is expressed with v_t is Gaussian measurement noise and w_t is Gaussian process noise:

$$w_t \sim \mathcal{N}(0, Q) \quad v_t \sim \mathcal{N}(0, R)$$

The Kalman filter is a online algorithm that estimates the hidden state of a dynamic system based on noisy observations. The model updates its estimation of the weights sequentially as new data comes in and provides an optimal solution (in a least-squares sense)

2.1 Algorithm

At its core, the Kalman filter operates on a predict-update cycle. This process is Bayesian, meaning the filter maintains a probabilistic belief over the system's state, updating it as new information comes in. The ultimate goal of the Kalman filter is to predict the next observation of the observed variable y by taking the best estimation of the hidden state variable x.

1. **Predict (estimation) step**: Given the previous state estimate $\hat{x}_{t-1|t-1}$, we predict the next state using a known transition model and calculate the current uncertainty P. Together they parametrize a probability density function of the estimate of the observed variable. The predicted observation for the next time step will be the maximum likelihood estimate (the mean).

$$\hat{x}_{t|t-1} = A\hat{x}_{t-1|t-1} \tag{3}$$

$$P_{t|t-1} = AP_{t-1|t-1}A^T + Q \tag{4}$$

2. **Update(correction) Step**: When a new observation arrives, we refine the predicted state using the measurement and its associated uncertainty.

   (a) Compute Kalman Gain:

$$K_t = P_{t|t-1}H^T \left(HP_{t|t-1}H^T + R\right)^{-1}, \tag{5}$$

   (b) Update state estimate:

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t \left(y_t - H\hat{x}_{t|t-1}\right) \tag{6}$$

   (c) Update uncertainty:

$$P_{t|t} = (I - K_t H)\, P_{t|t-1} \tag{7}$$

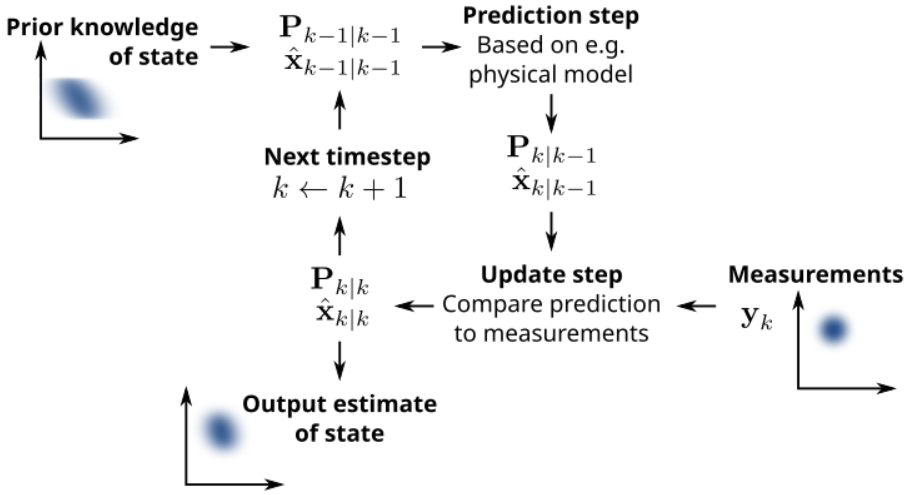

Figure 1: Scheme of calculating Kalman filters.

2.2 Kalman Gain

The Kalman Gain, denoted as K_t serves as an adaptive weighting mechanism in the estimation step. It assigns more weight to the more accurate estimate — typically, the one with a smaller Mean Squared Error (MSE). We can think of the Kalman Gain as a balancing act between what the model predicts and what the new data suggests.

Let's take variances of our current estimations and measurements. They define the error and the uncertainty.

$$K = \frac{\sigma_{\text{EST}}^2}{\sigma_{\text{MEA}}^2 + \sigma_{\text{EST}}^2} \tag{8}$$

This process demonstrates that K is influenced by the linear relationship between x and σ_{EST}^2 and y and σ_{MEA}^2.

As a result, a larger K suggests greater trust in the measurements (new data), as the model is not very sure about its current state. Vice versa, a smaller K implies greater confidence in the prediction, when the model is quite confident about its current state and does not need to adjust much based on the new data. Finally, the uncertainty of the estimation decreases as we have more samples.

3 Non-Linear Kalman Filters

You might have noticed that everything above far is just a fancy *linear* model. As shown above, we work only with linear transformation, thus limiting the set of problems that can be solved with Kalman Filter. To deal with it, there exists several modifications of Kalman filters.

3.1 Extended Kalman Filter (EKF)

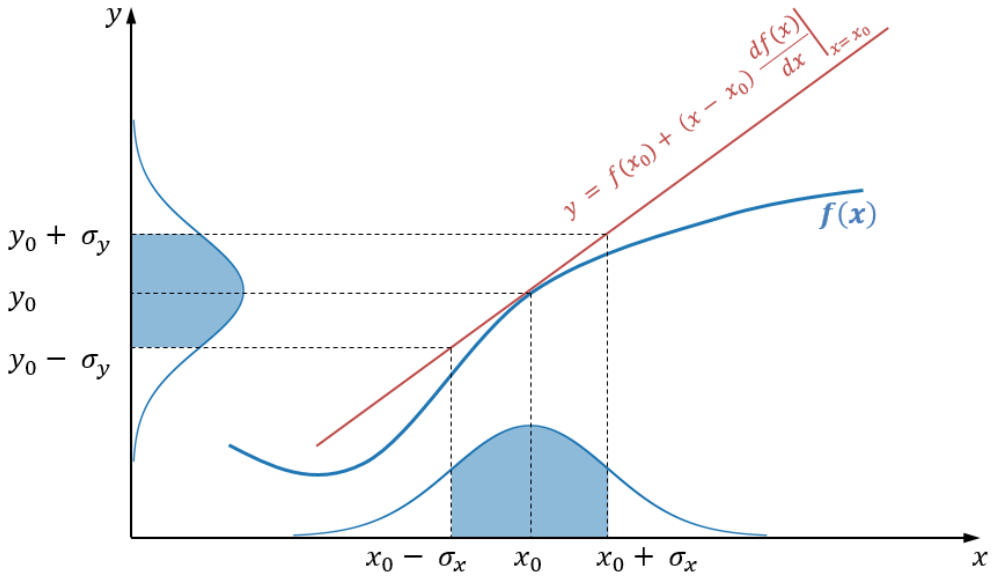

Figure 2: Analytical Linearization for EKF. From [1]

The first trick is to use the Taylor approximation of the first order because it is also linear. What we want to do is find a tangent line for a point $x = x_0$. Using the tangent line, we can project the uncertainty to the y axis and keep its' shape Gaussian.

The only change in how Kalman filter is calculated is that instead of matrices A and H, we take derivatives of corresponding functions. For example, the equation 4 turns into:

$$P_{t|t-1} = \frac{\partial a}{\partial x} P_{t-1|t-1} \left(\frac{\partial a}{\partial x} \right)^T + Q \tag{9}$$

3.2 Unscented Kalman Filter

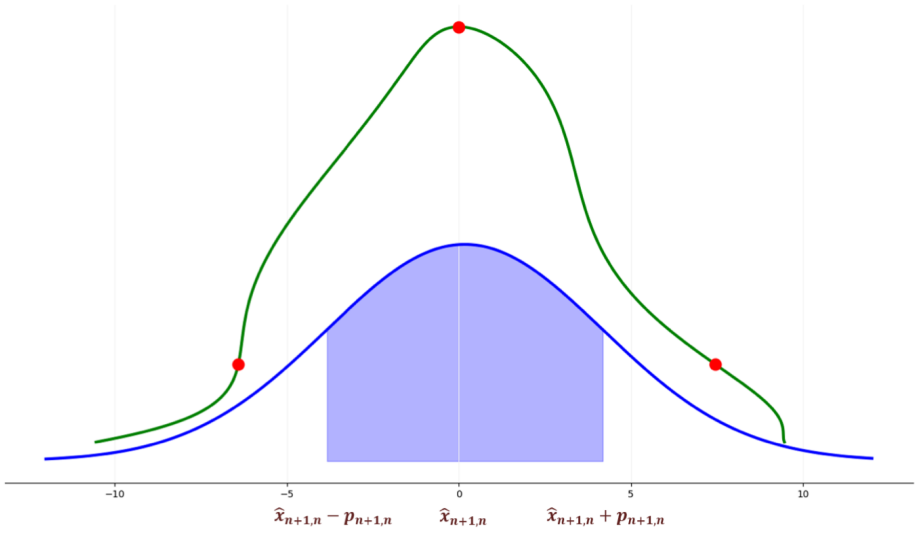

Figure 3: Example of 1D Unscented Transform. From [1]

When the state transition and observation models are nearly linear, EKF works well. But with strong non-linearities, EKF's linearization can lead to inaccurate estimates and unreliable uncertainty measures.

The Unscented Kalman Filter [2] offers a better alternative. Instead of approximating non-linear functions with derivatives, it uses a statistical technique called the **unscented transform**. It is a method for calculating the statistics of a random variable that undergoes a non-linear transformation. It includes three steps:

1. Select a set of $2N$ sigma points from the input distribution. The points are selected according to a specific, deterministic algorithm.
2. Propagate each selected point through the non-linear function, producing a new set of points belonging to the output distribution.
$$\mathcal{X}_{t+1,t} = \begin{bmatrix} \mathcal{X}_{t+1,t}^{(0)} & \mathcal{X}_{t+1,t}^{(1)} & \cdots & \mathcal{X}_{t+1,t}^{(2N)} \end{bmatrix}$$
3. Compute sigma points weights $\boldsymbol{w}$.
4. Approximate the sample mean and covariance of the output distribution using the propagated set of points and carefully chosen weights.
$$\hat{\boldsymbol{x}}_{t+1,t} = \sum_{i=0}^{2N} w_i \mathcal{X}_{t+1,t}^{(i)}$$
$$\boldsymbol{P}_{t+1,t} = \sum_{i=0}^{2N} w_i \left(\mathcal{X}_{t+1,t}^{(i)} - \hat{\boldsymbol{x}}_{t+1,t} \right) \left(\mathcal{X}_{t+1,t}^{(i)} - \hat{\boldsymbol{x}}_{t+1,t} \right)^T$$

4 Our Framework

We provide a clean, minimalistic, and extensible implementation of different Kalman filter variants:

- Standard Kalman Filter – The foundational model for linear state estimation.
- Extended Kalman Filter (EKF) – Handling nonlinear dynamics through local linearization.
- Unscented Kalman Filter (UKF) – A more accurate approach using sigma-point sampling.
- Variational Kalman Filters – Leveraging modern probabilistic techniques for scalable inference.

After installing the package, you can quickly set up and run a Kalman filter with just a few lines of code. For example, to use the standard Kalman filter, simply import the class, define your system parameters, and iterate over your measurements:

```python
import numpy as np
from kalman.filters import KalmanFilter

# Example: 1D constant position model
A = np.array([[1]])      # State transition matrix (position stays the same)
H = np.array([[1]])      # Observation matrix (we observe the position directly)
Q = np.array([[0.01]])   # Process noise covariance
R = np.array([[1]])      # Observation noise covariance
x0 = np.array([[0]])     # Initial state estimate
P0 = np.array([[1]])     # Initial covariance estimate

kf = KalmanFilter(A, H, Q, R, x0, P0)

# Simulated noisy measurements
measurements = [1.2, 0.9, 1.0, 1.1, 0.95]

for z in measurements:
    kf.predict()
    kf.update(np.array([[z]]))
    print("Current state estimate:", kf.x.flatten()[0])
```

References

[1] Becker, A. (2023). *Kalman filter: From the ground up.*

[2] Julier, S. and Uhlmann, J. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422.

[3] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45.