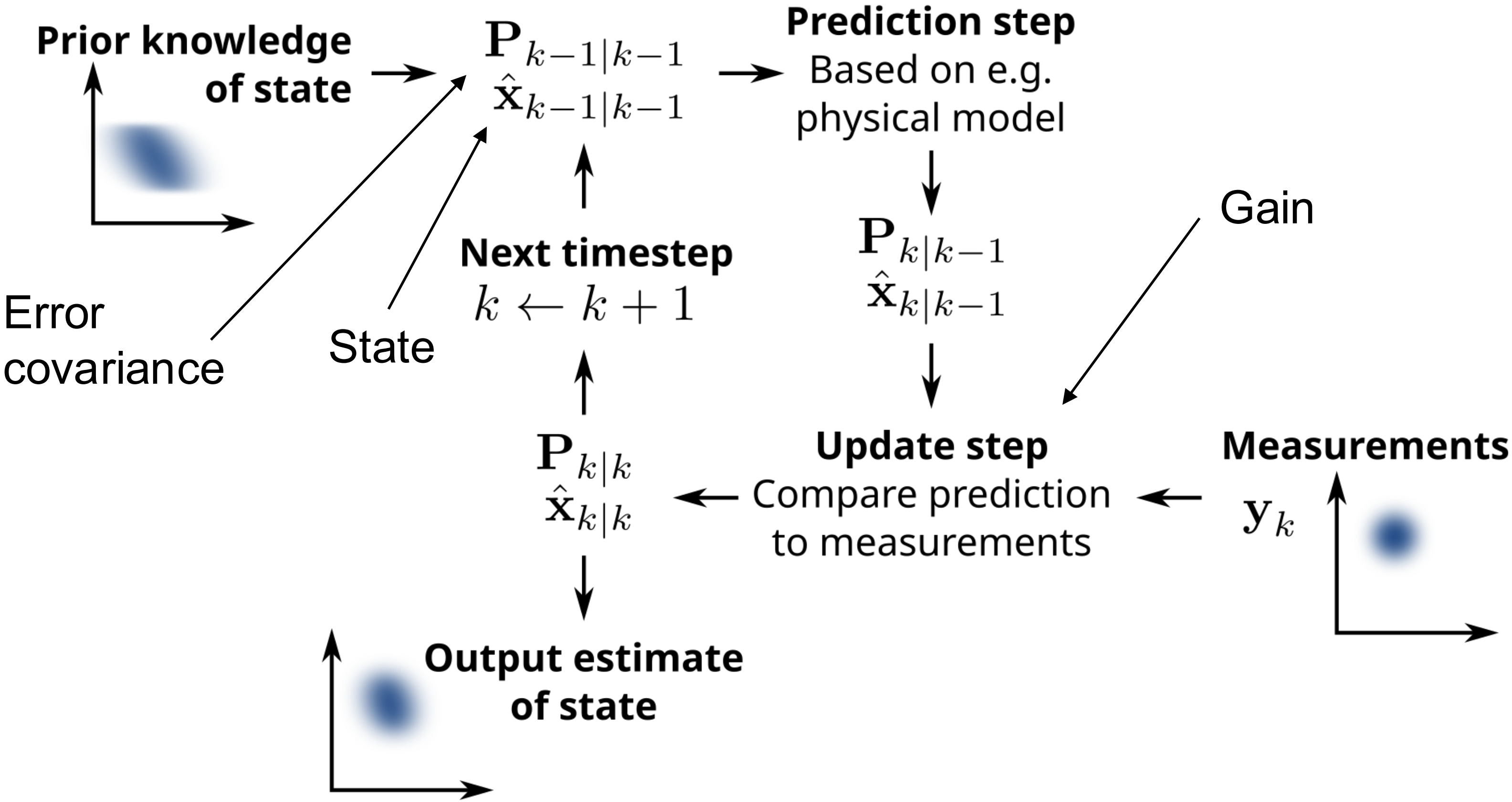
NTNU

Norwegian University of Science and Technology

System Identification with Kalman Filters

Adrian Langmo, Giacomo Gardella, Terje Haughland, Daniel Centurion Barrionuevo, Erik Odinsen, Muhammad Hilmi

Kalman Filters



Examples

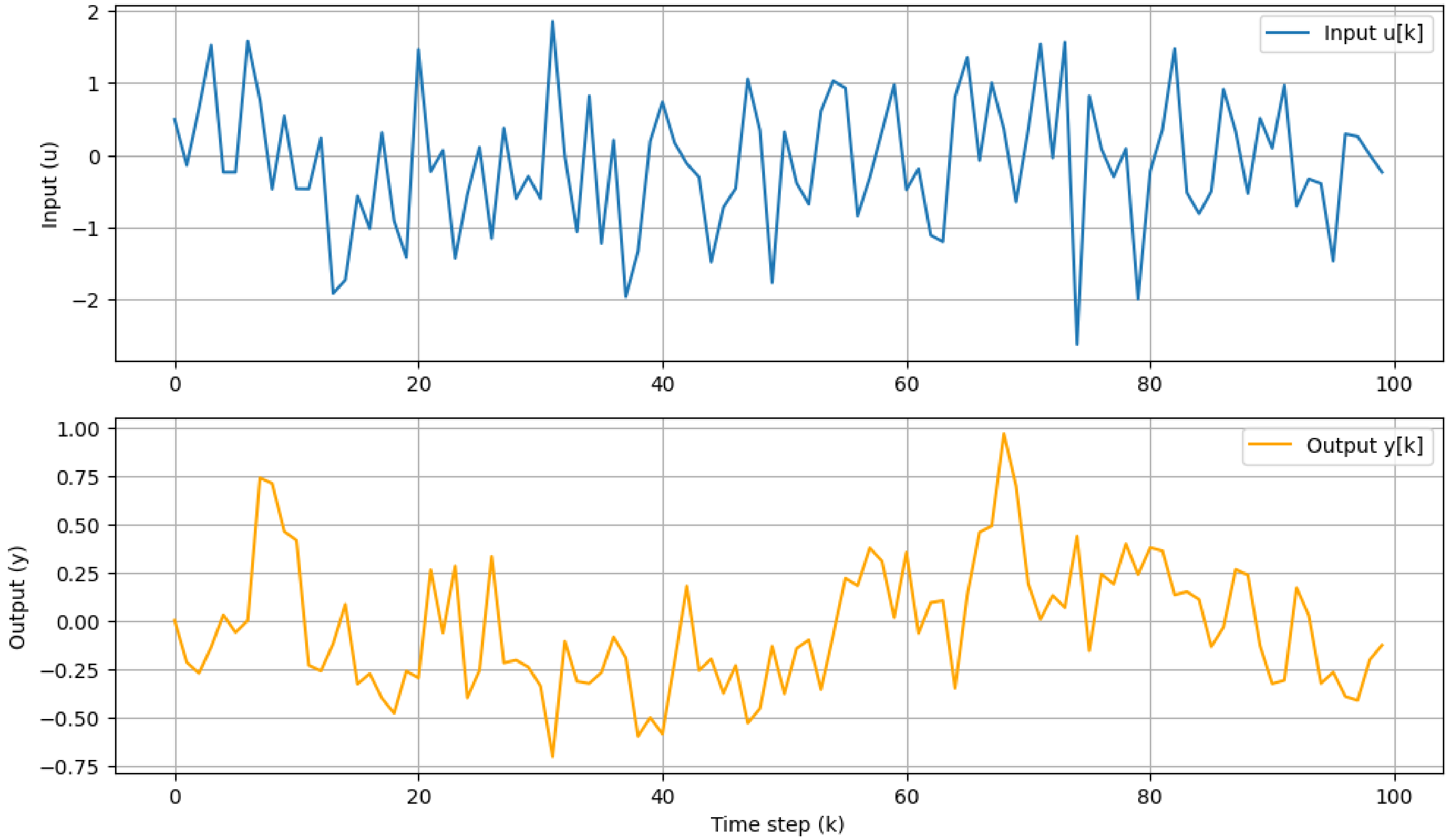
- Take an ARX model:

$$y_k = ay_{k-1} + bu_{k-1} + v_k$$

y output, *u* input, *v* noise, and *k* time step.

→ Estimate parameters *a* and *b*

$$y_k = 0.7y_{k-1} + 0.2u_{k-1} + v_k$$



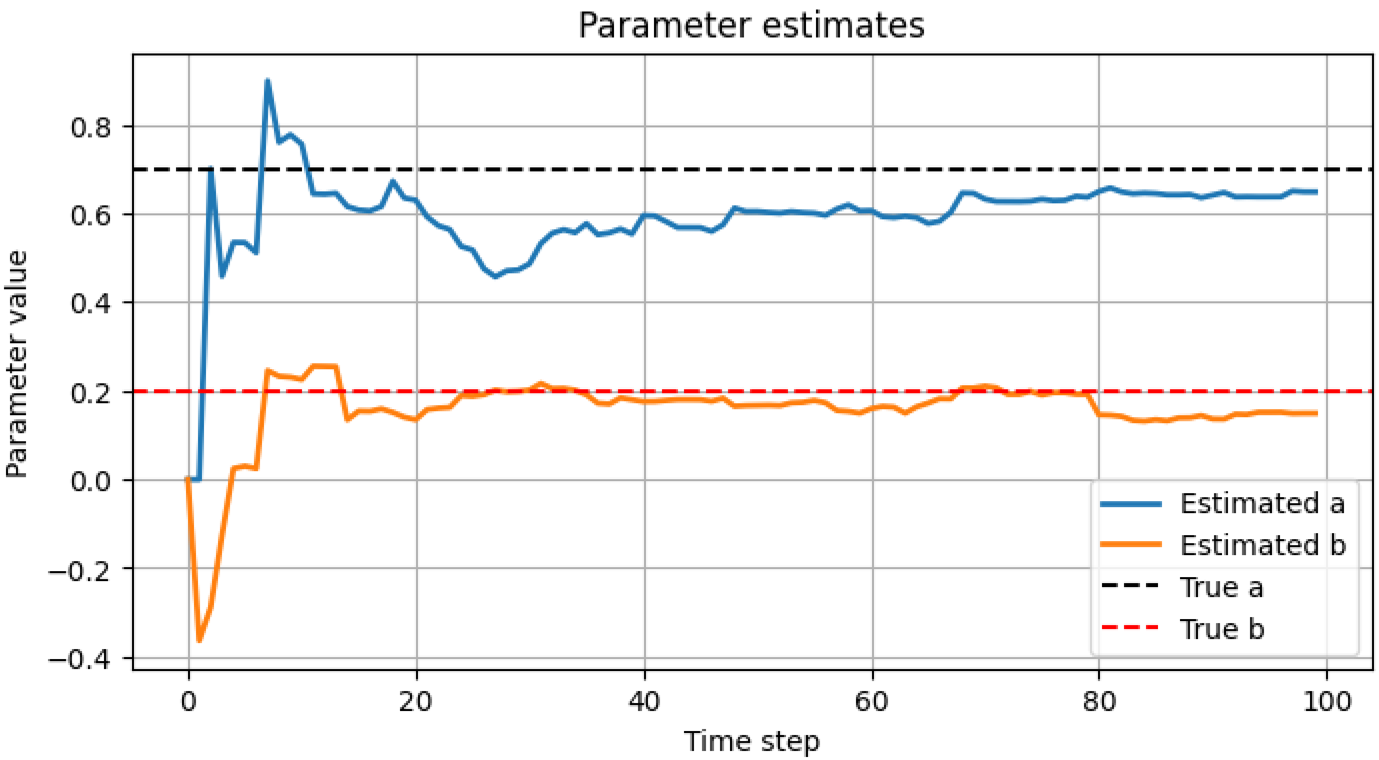
from filterpy.kalman import KalmanFilter

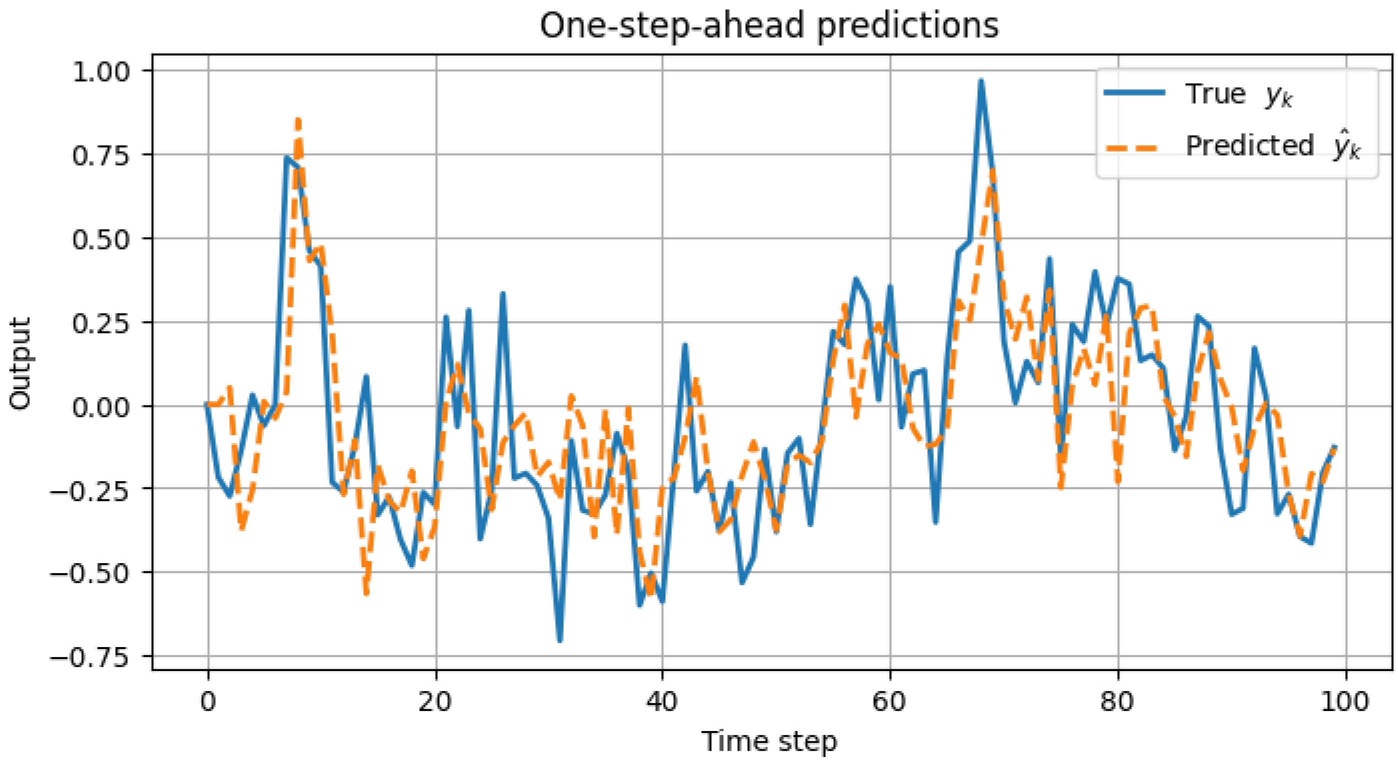
OR

Equations

```python
# FilterPy Kalman filter + one-step-ahead prediction
kf = KalmanFilter(dim_x=2, dim_z=1)
kf.F = np.eye(2)
kf.Q = 1e-4 * np.eye(2)
kf.R = np.array([[v_var]])
kf.x = np.array([[0.], [0.]])
kf.P = np.eye(2)

theta_hat = np.zeros((2, N))
y_pred = np.zeros(N)
for k in range(1, N):
    phi = np.array([[y[k-1], u[k-1]]])
    kf.H = phi
    kf.predict()
    y_pred[k] = float(phi @ kf.x)
    kf.update(np.array([[y[k]]]))
    theta_hat[:, k] = kf.x.ravel()
```

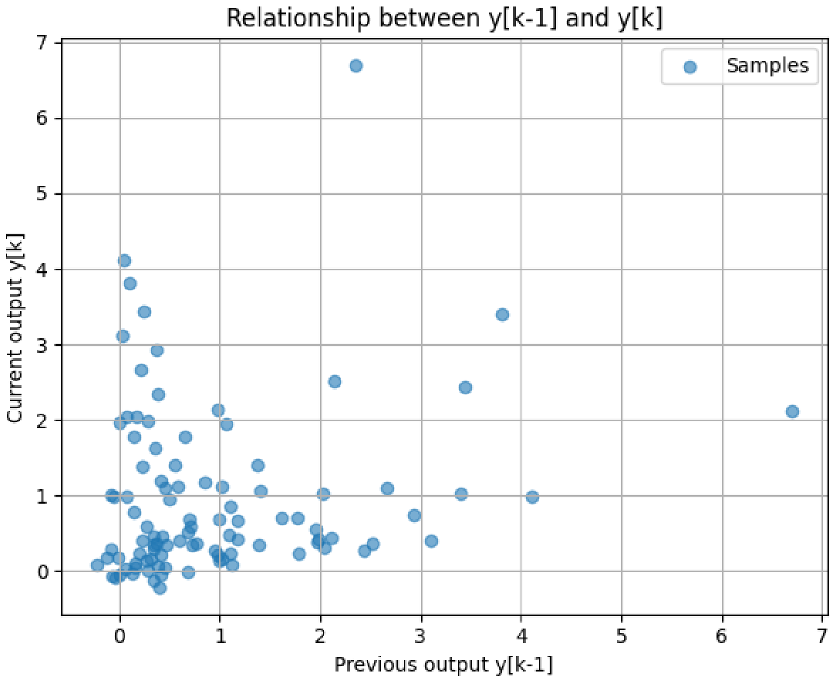
```python
# Kalman Filter loop
for k in range(1, N):
    phi = np.array([[y[k-1]], [u[k-1]]])  # regressor

    # Prediction
    theta_pred = theta_hat[:, k-1].reshape(-1,1)
    P_pred = P + Q

    # Kalman gain
    K = P_pred @ phi / (phi.T @ P_pred @ phi + R)

    # Measurement update
    y_pred = (phi.T @ theta_pred).item()
    theta_new = theta_pred + K * (y[k] - y_pred)
    P = (np.eye(2) - K @ phi.T) @ P_pred

    theta_hat[:, k] = theta_new.ravel()
```

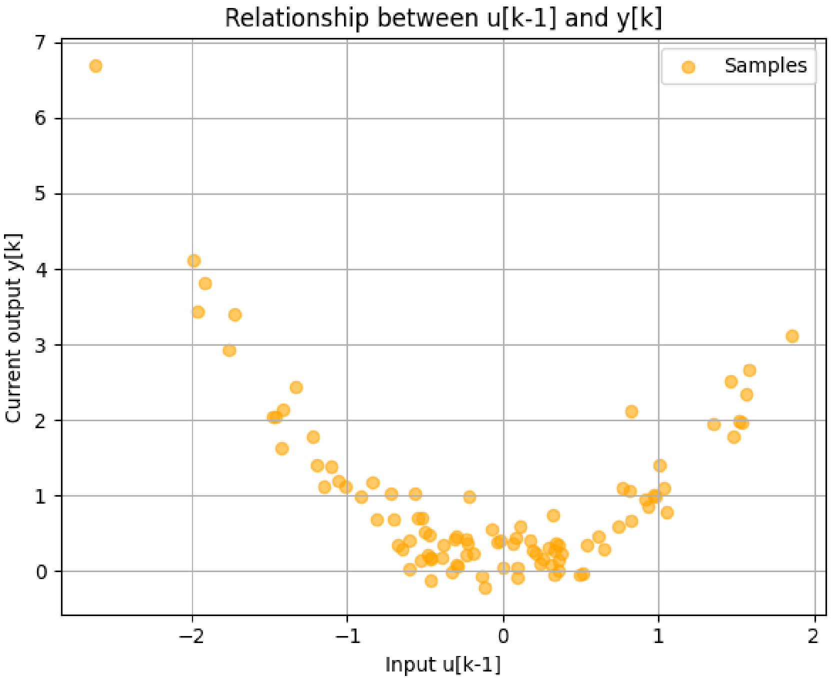

Parameter estimates
Parameter value
Time step
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
20
40
60
80
100
Estimated a
Estimated b
True a
True b


One-step-ahead predictions
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Output
0
20
40
60
80
100
Time step
True y_k
Predicted $\hat{y}_k$

Nonlinear

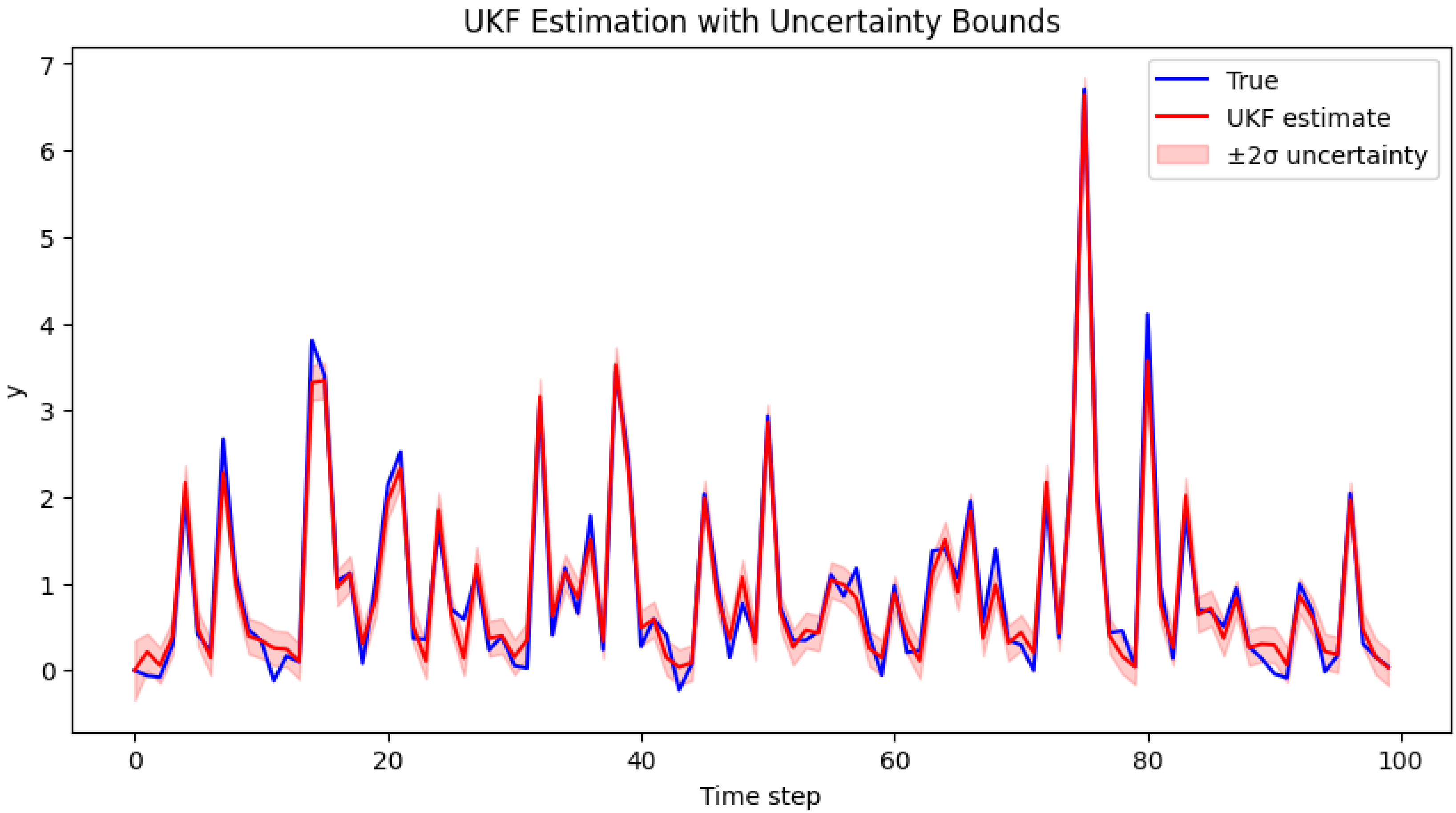
$$y_k = a\, y_{k-1} + b\,(u_{k-1})^2 + v_k$$





```python
from filterpy.kalman import UnscentedKalmanFilter as UKF
from filterpy.kalman import MerweScaledSigmaPoints
```

```python
# UKF setup
points = MerweScaledSigmaPoints(n=1, alpha=0.1, beta=2., kappa=0)
ukf = UKF(dim_x=1, dim_z=1, fx=fx, hx=hx, dt=1.0, points=points)
ukf.x = np.array([0.0])
ukf.P *= 1.0
ukf.R = v_var
ukf.Q = 0.01

# Run UKF
y_est = np.zeros(N)
y_std = np.zeros(N)  # store uncertainty (std dev)
for k in range(N):
    ukf.predict(k=k)
    ukf.update(y[k])
    y_est[k] = ukf.x[0]
    y_std[k] = np.sqrt(ukf.P[0, 0])  # 1D state variance
```


UKF Estimation with Uncertainty Bounds
True
UKF estimate
±2σ uncertainty
y
Time step
0
1
2
3
4
5
6
7
0
20
40
60
80
100

Joint Estimation

Augmented system dynamics

$$\begin{bmatrix} x(k) \\ \theta(k) \end{bmatrix} = \begin{bmatrix} A & \Psi(y(k), u(k)) \\ 0 & I \end{bmatrix} \begin{bmatrix} x(k-1) \\ \theta(k-1) \end{bmatrix} + \begin{bmatrix} \eta(k) \\ \xi(k) \end{bmatrix}$$

$$y(k) = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ \theta(k) \end{bmatrix} + \epsilon(k)$$

Priori estimation (prediction)

$$\begin{bmatrix} \hat{x}(k) \\ \hat{\theta}(k) \end{bmatrix} = \begin{bmatrix} A & \Psi(y(k), u(k)) \\ 0 & I \end{bmatrix} \begin{bmatrix} \hat{x}(k-1) \\ \hat{\theta}(k-1) \end{bmatrix}$$

Kalman filter

$$\begin{aligned} P(k|k-1) &= AP(k-1|k-1)A^T + Q(k) \\ \Sigma(k) &= CP(k|k-1)C^T + R(k) \\ K(k) &= P(k|k-1)C^T\Sigma^{-1}(k) \\ P(k|k) &= (I_n - K(k)C)P(k|k-1) \end{aligned}$$

RLS-like

$$\begin{aligned} &\Omega(k) = CA(k)\Upsilon(k-1) + C\Psi(y(k), u(k)) \\ &\Upsilon(k) = (I_n - K(k)C)A\Upsilon(k-1) + (I_n - K(k)C)\Psi(y(k), u(k)) \\ &\Lambda(k) = (\lambda\Sigma(k) + \Omega(k)S(k-1)\Omega^T(k))^{-1} \\ &\Gamma(k) = S(k-1)\Omega^T(k)\Lambda(k) \\ &S(k) = \lambda^{-1}(S(k-1) - S(k-1)\Omega^T(k)\Lambda(k)\Omega(k)S(k-1)) \end{aligned}$$

Posteriori estimation (update)

$$\begin{aligned} \hat{\theta}(k|k) &= \hat{\theta}(k|k-1) + \underline{\Gamma(k)}(y(k) - C\hat{x}(k|k-1)) \\ \hat{x}(k|k) &= \hat{x}(k|k-1) + \underline{K(k)}(y(k) - C\hat{x}(k|k-1)) + \underline{\Upsilon(k)}(\hat{\theta}(k|k) - \hat{\theta}(k|k-1)) \end{aligned}$$

Zhang, Q. (2018). Adaptive Kalman filter for actuator fault diagnosis. *Automatica*, *93*, 333-342

Why?

Augmented estimation:

Observability

$$\mathrm{Rank}[\mathcal{O}(A_{\mathrm{aug}}(k), C_{\mathrm{aug}})] = n + p \mid \mathrm{Rank}[\mathcal{O}(A, C)] = n$$

Controllability

$$\mathrm{Rank}[\mathcal{C}(A_{\mathrm{aug}}(k), Q^{\frac{1}{2}}_{\mathrm{aug}}(k))] = n + p \mid \mathrm{Rank}[\mathcal{C}(A, Q^{\frac{1}{2}}(k))] = n$$

Persistent excitation

$$\sum_{j=k}^{k+N-1} \Psi(y(j), u(j))^T \Psi(y(j), u(j)) \geq \sigma I_p \geq 0$$

Joint estimation:

Observability

$$\mathrm{Rank}[\mathcal{O}(A, C)] = n$$

Controllability

$$\mathrm{Rank}[\mathcal{C}(A, Q^{\frac{1}{2}}(k))] = n$$

Persistent excitation

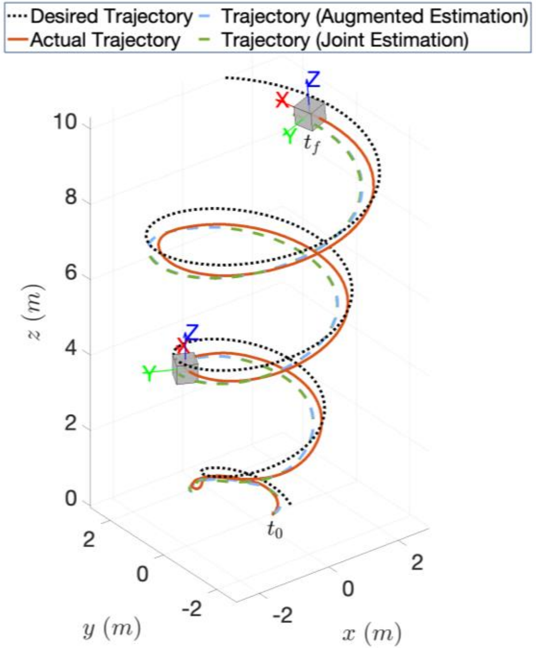
$$\sum_{j=k}^{k+N-1} \Omega^T(j) \Sigma^{-1}(j) \Omega(j) \geq \sigma I_p \geq 0$$

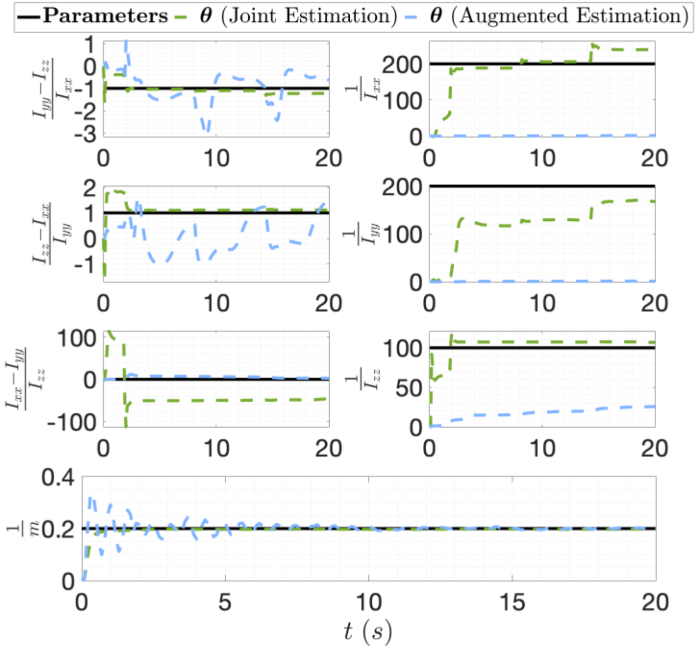
M. Hilmi, A. Hasan and M. A. Lundteigen, "Self-Tuning Kalman Filter for Fault Estimation of Nonlinear Systems," *in the 64th IEEE Conference on Decision and Control (CDC)*, Rio de Janeiro, Brazil, 2025.

WyNDA Example

Library functions

$$\Phi_i(*) = [q(k)r(k) \quad \tau_\phi(k) \quad p(k)r(k) \quad \tau_\alpha(k) \quad p(k)q(k) \quad \tau_\psi(k) \quad T_z(k)]$$





https://github.com/hilmi-ica/WyNDA-UAV.git

M. Hilmi, S. F. Ramadhan, A. Widyotriatmo an A. Hasan, "Real-Time Parameter Estimation for Digital Twins of Unmanned Aerial Vehicles," in *IEEE Transactions on Intelligent Vehicles*, 2025 [Submitted].