Kalman filter tutorial for beam current tracking

Victor Watson[1]

[1]Nuclear Science Division
Lawrence Berkeley National Laboratory

June 2025

Introduction

Kalman Filter

- Widely used for **tracking** purposes
- Can be seen as an online optimal **low pass filter**
- Has an **adaptive gain** based on **Bayesian statistics**

Characteristics

- As a filter it is a **linear** and **invariant system**
- Still behaves **very well with non-linear** signals
- Gives a **prediction** of the current state based on a **linearisation** of past data
- **Updates** the prediction with the **measure** and gives the **estimate** thanks to the **optimal gain**

Theory and Algorithm

Theory

- The prediction of the state is given by:

$$X_p(k) = \hat{X}(k-1) + \frac{dX}{dt} * \delta t + \mathcal{N}(0, \mathbf{P}_p(k)) \tag{1}$$

with $\hat{X}(k-1)$ the previous estimated state and $\mathbf{P}_p$ the prediction covariance matrix

- The measure follows:

$$Z_m(k) = Z(k) + \mathcal{N}(0, \mathbf{R}) \tag{2}$$

with $Z(k)$ the real value and $\mathbf{R}$ the covariance of the noise

- The estimate is:

$$\hat{X}(k) = X_p(k) + (\mathbf{K} * Z_m(k))^T \tag{3}$$

Theory and Algorithm

Theory

- With the optimal Kalman gain:

$$\mathbf{K} = (\mathbf{P}(k-1)\mathbf{H}^T)^T\mathbf{S}(k)^{-1} \tag{4}$$

H is the transition matrix from state space to measure space

- The prediction covariance matrix is calculated:

$$\mathbf{P}_p(k) = \mathbf{F} * \mathbf{P}(k-1)\mathbf{F}^T + \mathbf{Q} \tag{5}$$

where F is the state transition matrix **P** is the estimate covariance matrix and **Q** is a matrix set by the user inversely proportional to the confidence they have that the system is linearisable.

Theory and Algorithm

Theory

- **S** is the covariance of the innovation:

$$\mathbf{S}(k) = \mathbf{H} * \mathbf{P}_p(k) + \mathbf{R} \tag{6}$$

where **R** is set by the user and is proportional to the standard deviation of the measure noise

Algorithm

- Initialization
  - Set **F**, **H**, **R**, **Q**, $\hat{X}(0)$ (we use the same 4 matrices for $\hat{\sigma}_{beam}$)
  - **F** is the transition matrix from a state to the next if $X = [position, \frac{dx}{dt}]^T$ then $F = [[1, \delta t], [1, 0]]$ that way $X(k) = [X(k-1)[0] + \frac{dx}{dt}\delta t, \frac{dx}{dt}]$ we suppose the system locally linear so **F** propagate dx.
  - **H** usually the first part of X (in our case X(k)[0]) is homogeneous to the measure so $\mathbf{H} = [1, 0]^T$ (state is 2d measure is 1d)

Theory and Algorithm

Algorithm

- Initialization
  - $\hat{X}(0)$ **best guess** about the state of the space in the beginning usually the first measure for $X(0)[0]$ and start with a null dynamic (unless better idea) $X(0)[1] = 0$
  - **R** Covariance of the measure dimension of the measure (in our case a scalar) pick a **big number** because we assume the measure is imprecise
  - **Q** Covariance of the prediction dimension of the state (in our case 2x2) pick a **small number** because we assume the system is mostly linearisable (eye(2)*something small) the **ratio R/Q** will give the reactivity or the cutting frequency of the filter if this ratio is **too small**, the filter will follow the noise and be completely **useless**, if it is **too big** the filter will take a **longer time** to catch-up when the **trajectory** of the system changes

Theory and Algorithm

Prediction

- Prediction
  - Predicting X:
$$X_p(k) = \mathbf{F}\hat{X}(k) \tag{7}$$
  - Covariance of predicted X:
$$\mathbf{P}_p(k) = \mathbf{F}\mathbf{P}(k-1)\mathbf{F}^T + \mathbf{Q} \tag{8}$$
  - Predicting σ:
$$\sigma_p(k) = \mathbf{F}\hat{\sigma}(k) \tag{9}$$
  - Covariance of predicted X:
$$\mathbf{Ps}_p(k) = \mathbf{F}\mathbf{Ps}(k-1)\mathbf{F}^T + \mathbf{Q} \tag{10}$$

Estimation

- Update
  - Innovation of X (z is the measure):

$$y = z(k) - \mathbf{H}X_p(k) \tag{11}$$

  - Covariance of innovation:

$$\mathbf{S}(k) = \mathbf{H}\mathbf{P}_p(k)\mathbf{H}^T + \mathbf{R} \tag{12}$$

  - Optimal Kalman Gain:

$$\mathbf{K}(k) = (\mathbf{P}(k-1)\mathbf{H}^T)^T\mathbf{S}(k)^{-1} \tag{13}$$

  - Estimating X

$$\hat{X}(k) = X_p(k) + (\mathbf{K}(k)y)^T \tag{14}$$

  - updating X covariance:

$$\mathbf{P}(k) = (\mathbb{1}_{2\times 2} - \mathbf{K}\mathbf{H})\mathbf{P}_p(k) \tag{15}$$

Theory and Algorithm

Estimation

- Update
  - Innovation of σ using $\hat{X}$:

$$y = \sqrt{(z(k) - \hat{X}(k)[0])^2} - \mathbf{H}\sigma_p(k) \tag{16}$$

  - Covariance of innovation:

$$\mathbf{Ss}(k) = \mathbf{HPs}_p(k)\mathbf{H}^T + \mathbf{R} \tag{17}$$

  - Optimal Kalman Gain:

$$\mathbf{Ks}(k) = (\mathbf{Ps}(k-1)\mathbf{H}^T)^T\mathbf{Ss}(k)^{-1} \tag{18}$$

  - Estimating σ

$$\hat{\sigma}(k) = \sigma_p(k) + (\mathbf{Ks}(k)y)^T \tag{19}$$

  - updating σ covariance:

$$\mathbf{Ps}(k) = (\mathbb{1}_{2\times 2} - \mathbf{Ks}(k)\mathbf{H})\mathbf{Ps}_p(k) \tag{20}$$

Code

object KFpar

- Contains all the **variables** necessary to characterize the state and **parameters** of the filter
- Is initialized with **cLin** for **Q** and **cMeas** for **R**[a]
- Receive the first measure in **KFpar.X[0]** and the elapsed time before every update in **KFpar.F[0,1]**
- Store Current KFpar.X[0], slope KFpar.X[1] and stdev KFpar.Sig[0]

[a] cLin and cMeas are scalar used to set **Q** and **R** considering $\mathbf{Q} = \mathbb{1}_{StateDim} * cLin$ and $\mathbf{R} = \mathbb{1}_{MeasureDim} * cMeas$

function EstimareState

- Updates the estimate of the Kalman Filter and its parameters with the new measure (KFpar.F needs to be updated before)
- KFpar = EstimateState(KFpar, measure)

Examples

Data

- Series of examples generated with the **KFtutorial.py** and **KFtutorialAtan.py**
- These are **illustrating** the impact of the **filter settings** in several cases
- These codes are adapted to track a 1D signals and extract from it estimates of the mean the slope and the standard deviation

Cases studies

- KFtutorial.py applies the Kalman filter to one of the times series in the files 'd1, 'd2, or 'd3'
- KFtutorialAtan.py allows the user to change the settings of an arctangent and add perturbation to it before applying the Kalman Filter

Example 'd1'

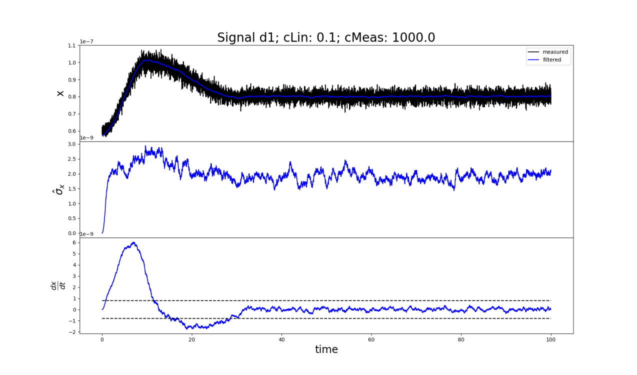

Figure: R = 1000; Q = 0.1

Q and R are adapted to most of the noise is filtered but x is tracked successfully (Note: the estimate of σ is impacted by the slope in the beginning)

Example 'd1'

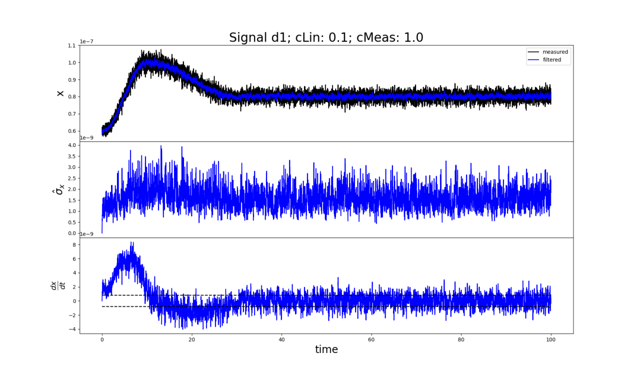

Figure: R = 1; Q = 0.1

Q and R are not adapted and **too much noise** is still impacting the estimate

Example 'd1'

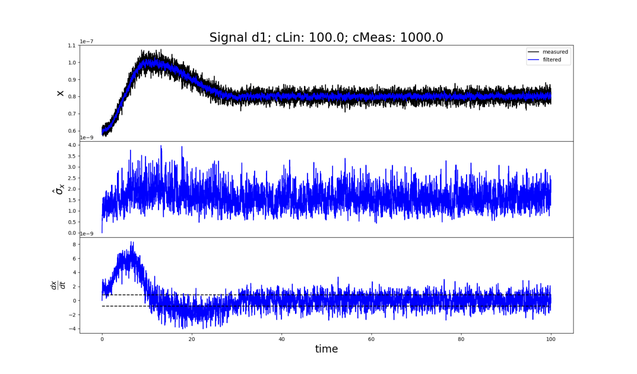

Figure: R = 100; Q = 1000

Q and R are not adapted and **too much noise** is still impacting the estimate note result is similar to the previous figure because ratio Q/R is the same

Example 'd1'

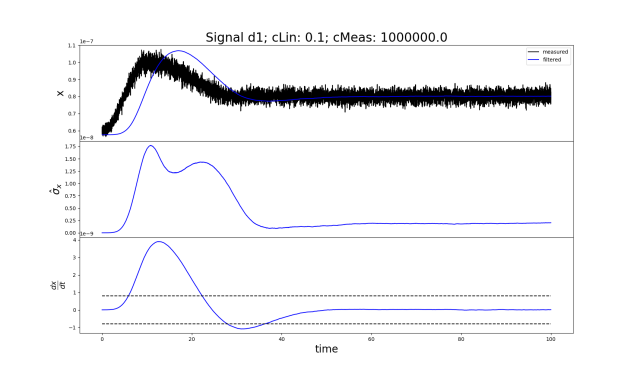

Figure: R = 1000000; Q = .1

Q and R are not adapted and the filter **cannot track** correctly the changes in x the delay impacts the slope estimate even more

Example 'd2'

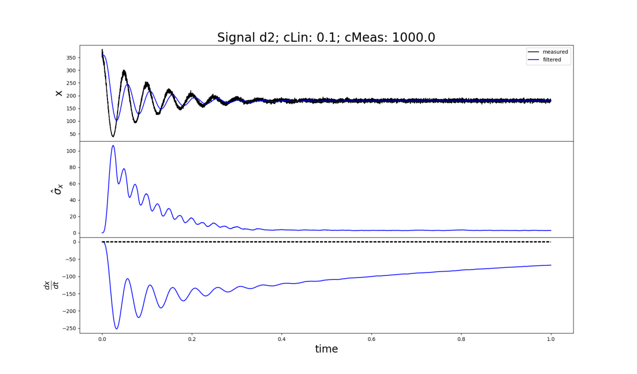

Figure: R = 1000; Q = .1

Q and R are not adapted and the filter **cannot track** correctly the changes in x

Example 'd2'

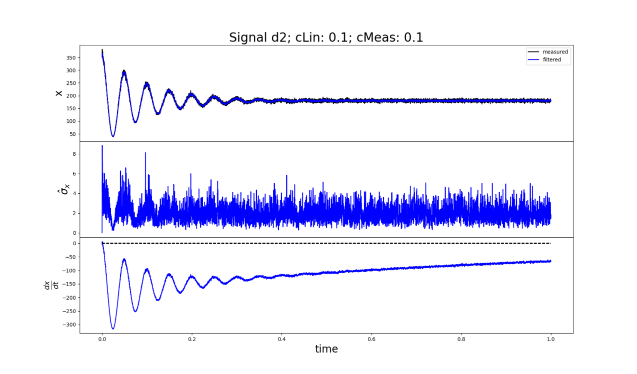

Figure: R = .1; Q = .1

Q and R are adapted to track x but the filter is **not very robust to noise** and the **slope convergence** is way **too slow**. I this case we see the limits of the assumption of the linearisable aspect of the signal.

Example 'd3'

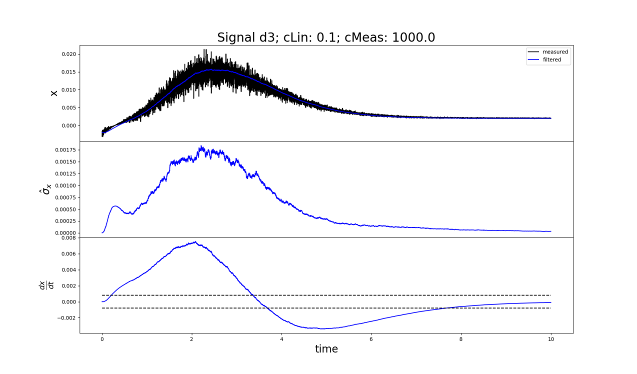

Figure: R = 1000; Q = .1

Q and R are adapted to track x and filter most of the noise but takes longer to estimate the slope due to the change in direction.

Example Atan

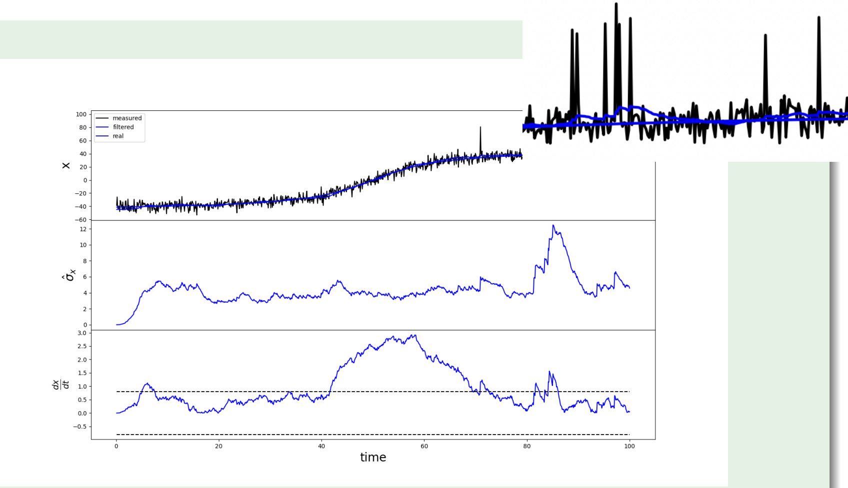

Figure: R = 1000; Q = .1

Q and R are adapted to track x and filter most of the noise.

Example Atan

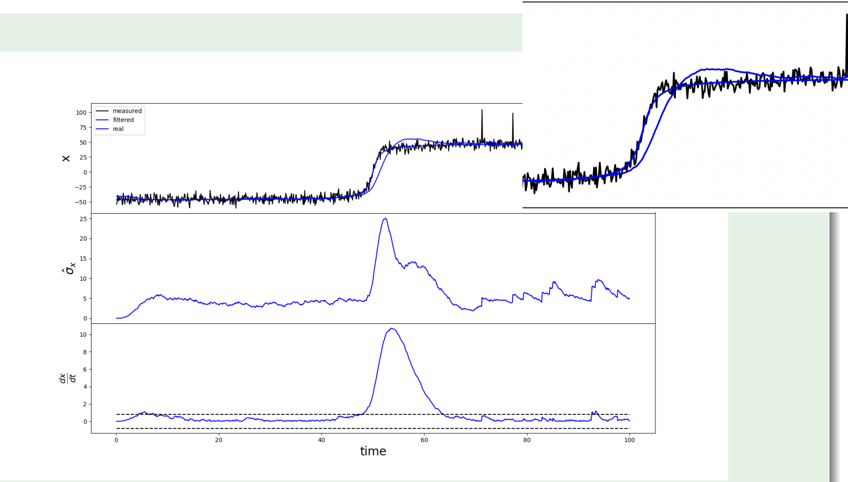

Figure: R = 1000; Q = .1

Q and R are not adapted to track x. Here the settings did not change but the change of **trajectory is sharper** and the settings are no longer adapted

Conclusion

Essential

- This Kalman filter is **adapted to a 1D signal** and estimate the measured value as well as the slope and standard deviation
- The use of this tool is valid as long as the system dynamic is **sufficiently linearisable vis a vis noise**
- If **brutal changes** in direction add detection and **reset filter**

How to

Set **cLin** (small) and **cMeas** (big) to determine the reactivity vs cutting frequency adapted to the dynamic and the noise

initialize KFpar.X[0] with the first measure

At every measure

- Enter time to since the last measure **F[0,1]** $= \delta t$
- **Update the Object**: KFpar = EstimateState(KFpar, measure)

Current: KFpar.X[0] ; Slope: KFpar.X[1] ; Stdev: KFpar.Sig[0]

Appendix

Compact alternative

- AlternateKFtutorial.py contains a **compact alternative** that does the same thing
- It encapsulate the function EstimateState as a method of the object **KFparAlt**

How to

Set **cLin** (small) and **cMeas** (big) to determine the reactivity vs cutting frequency adapted to the dynamic and the noise (ex: myKFparAlt = KFparAlt(.1,1000))

initialize myKFparAlt.X[0] with the first measure

At every measure

- **Update the Object**: myKFparAlt.EstimateState(measure, deltaT)

Current: myKFparAlt.X[0] ; Slope: myKFparAlt.X[1] ; Stdev: myKFparAlt.Sig[0]