Normalized innovation error squared (NIS) based Kalman Filter Auto-tuning

Hao Zhu

June 22, 2022

1 Preliminaries

1.1 Kalman Filter

A Kalman filter mainly consists of two steps:

Predict

Predicted (*a priori*) state estimate

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k$$

Predicted (*a priori*) estimate covariance

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k$$

Update

Innovation pre-fit residual

$$\tilde{y}_k = z_k - H_k \hat{x}_{k|k-1}$$

Innovation covariance

$$S_k = H_k P_{k|k-1} H_k^T + R_k$$

Optimal Kalman gain

$$K_k = P_{k|k-1} H_k^T S_k^{-1}$$

Updated (*a posteriori*) state estimate

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k$$

Updated (*a posteriori*) estimate covariance

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}$$

The performance of Kalman filter highly depends on the two parameters:

- State noise covariance matrix: Q
- Observation noise covariance matrix: R

Our task is to establish an auto-tuning process to find the optimal value of these two parameters under a given situation.

More details about Kalman filter see also [Faragher, 2012].

1.2 Objective Function

Here we consider the Normalized Innovation Error Squared (NIS), which is computed from

$$\epsilon_{z,k} = \tilde{y}_k^T S_{k|k-1}^{-1} \tilde{y}_k$$

It is often assumed that the prediction and observation errors are Gaussian. So when the dynamical consistency conditions are met under the chosen parameters (Q and R), $\epsilon_{z,k}$ will be χ^2-distributed random variables with n_z degrees of freedom, i.e.:

$$E[\epsilon_{z,k}] \approx n_z$$

where n_z refers to the measurement(observation)-vector dimension. Then an objective function for parameters tuning based on NIS can be build:

$$J_{NIS}(Q,R) = \left| \log \left(\frac{\sum_{k=1}^{T} \epsilon_{z,k}/T}{n_z} \right) \right|$$

A close to zero value of this objective function will reveal a dynamical consistent estimation of the observations by the Kalman filter using the chosen parameters.

More information see also [Chen et al., 2018], [Chen et al., 2019].

1.3 Optimization

Here we use the Tree-structured Parzen Estimators (TPE) algorithm for optimization. TPE is a kind of Sequential Model-Based Optimization (SMBO) algorithm (Algorithm 1) which iterates between fitting models and using the calculated objective function value to make choices about which parameters to investigate next. When the searching iteration ends, SMBO will return the parameters which generate the optimal objective function value. The main idea of TPE is similar to Bayesian optimization but the algorithm is different. This approach can find the optimal objective function value with out stuck in the local minimum (or maximum) as well but behaves better than Bayesian optimization with Gaussian process regression.

Algorithm 1 Sequential Model-Based Optimization (SMBO)
$\boldsymbol{R}$ keeps track of all target algorithm runs performed so far and their performances, $\mathcal{M}$ is SMBO's model, $\vec{\boldsymbol{\Theta}}_{new}$ is a list of promising configurations.
(Modified from [Hutter et al., 2011])

Input: Target algorithm A with parameter configuration space $\boldsymbol{\Theta}$; instance set Π; objective function J
Output: Optimized (incumbent) parameter configuration, $\boldsymbol{\theta}_{inc}$

1: $[\boldsymbol{R}, \boldsymbol{\theta}_{inc}] \leftarrow Initialize(\boldsymbol{\Theta}, \Pi)$
2: **repeat**
3: $\quad \mathcal{M} \leftarrow FitModel(R)$
4: $\quad \vec{\boldsymbol{\Theta}}_{new} \leftarrow SelectConfigurations(\mathcal{M}, \boldsymbol{\theta}_{inc}, \boldsymbol{\Theta})$
5: $\quad [\boldsymbol{R}, \boldsymbol{\theta}_{inc}] \leftarrow Intensify(\vec{\boldsymbol{\Theta}}_{new}, \boldsymbol{\theta}_{inc}, \mathcal{M}, \boldsymbol{R}, \Pi, J)$
6: **until** *termination criteria met*
7: ***return*** $\boldsymbol{\theta}_{inc}$

More information about SMBO: [Hutter et al., 2011], and TPE: [Bergstra et al., 2011].

2 Example: Finding the optimal Kalman filter parameters for fruit fly *Drosophila melanogaster* trajectory smoothing

Suppose that we need to find the optimal Kalman filter parameters for smoothing the recorded *Drosophila* trajectory in an arena. We already have several recorded trajectories (observations) which are consists of the x and y position (in pixel unit) of the fly in the arena. The basic idea is that we can first find the optimal filter parameters based on some recorded trajectories through the auto-tuning procedure described above, and then test the filter performance on a test trajectory. Then the optimal parameters found can be used in the smoothing of other *Drosophila* trajectories.

2.1 Designing Kalman Filter

Our observation of the fly position consists of a 2-dimensional vector (x, y) which is recorded with a time interval of τ. Taking the velocity of fly along the two axis into account, the state vector can be represented as a 4-dimensional vector[1]:

$$\mathbf{x} = [x, \dot{x}, y, \dot{y}]^T$$

Under the state transition function $\mathbf{x} = F\mathbf{x}$, the state transition matrix (F) is

$$F = \begin{bmatrix} 1 & \tau & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \tau \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The covariance matrix of the motion noise $Q(\tau)$ is defined as a function of τ:

$$Q(\tau) = \begin{bmatrix} \frac{1}{3}\tau^3 & \frac{1}{2}\tau^2 & 0 & 0 \\ \frac{1}{2}\tau^2 & \tau & 0 & 0 \\ 0 & 0 & \frac{1}{3}\tau^3 & \frac{1}{2}\tau^2 \\ 0 & 0 & \frac{1}{2}\tau^2 & \tau \end{bmatrix} \sigma_w^2$$

where σ_w^2 represents the variance in the motion noise.

Since the observation vector (z) is a 2-dimensional vector consists of the observed x and y coordinate of the fly, the observation matrix H should be a 2×4 matrix:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The observation noise covariance matrix is then defined as

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \sigma_v^2$$

[1] From here we start to use $\mathbf{x}$ for state vector and x for coordinate.

where σ_v^2 represents the variance in the observation noise. Based on the design of motion and observation noise covariance matrix above we only need to optimize the scaler σ_w^2 and σ_v^2 later.

Besides, we also need to specify the initial state ($\mathbf{x}_0$) and state covariance matrix (P_0):

$$\mathbf{x}_0 = [x_0, 0, y_0, 0]^T$$

$$P_0 = p \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where x_0 and y_0 represent the initial observation of the fly position and we set $p = 1000$ which is a scaler for P_0.

2.2 Performing Optimization

Since here we use more than one trajectories to find the optimal Kalman filter parameters, in each iteration, we calculate the respective objective function value for each trajectory and then take the average as the final "loss" for the batch of trajectories, i.e.

$$J_{NIS,batch}(Q,R) = \frac{1}{M}\sum_{m=1}^{M} J_{NIS,m}(Q,R) = \frac{1}{M}\sum_{m=1}^{M}\left|\log\left(\frac{\sum_{k=1}^{T}\epsilon_{z,k,m}/T}{n_z}\right)\right| \tag{1}$$

where M is the number of trajectories in the training set.

Based on this process σ_w^2 and σ_v^2 is optimized to minimize the objective function value using TPE algorithm. The entire flow of the optimization process is listed in Algorithm 2.

Algorithm 2 Optimizing the Kalman filter parameters.
$\boldsymbol{R}$ keeps track of all "loss" under each parameter configuration, $\vec{\boldsymbol{\Theta}}_{init} \subset \boldsymbol{\Theta}_{\sigma_w^2,\sigma_v^2}$ is used for initialize the TPE model.

Input: Kalman filter model Π_{KF}; parameter configuration space $\boldsymbol{\Theta}_{\sigma_w^2,\sigma_v^2}$; objective function $J_{NIS,batch}$; batch of observations (trajectories) X_{batch}

Output: Optimal parameter configuration $\boldsymbol{\theta}_{\sigma_w^2,\sigma_v^2,optim}$

Initialize TPE with $[\Pi_{KF}, \vec{\boldsymbol{\Theta}}_{init}, X_{batch}]$
while maximum iterations not reached **do**
  $J_{NIS,batch} \leftarrow \Pi_{KF}(\boldsymbol{\theta}_{\sigma_w^2,\sigma_v^2}, X_{batch})$
  $\boldsymbol{R} \Leftarrow J_{NIS,batch}$
  $\boldsymbol{\theta}_{\sigma_w^2,\sigma_v^2} \leftarrow$ Update TPE model with $\boldsymbol{R}$
end while

Python implementation and results of this example can be found in `main.pdf` and `output` directory of https://github.com/HaoZhu10015/Kalman-Filter-Auto-Tuning.

References

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

Zhaozhong Chen, Christoffer Heckman, Simon Julier, and Nisar Ahmed. Weak in the nees?: Auto-tuning kalman filters with bayesian optimization. In *2018 21st International Conference on Information Fusion (FUSION)*, pages 1072–1079. IEEE, 2018.

Zhaozhong Chen, Nisar Ahmed, Simon Julier, and Christoffer Heckman. Kalman filter tuning with bayesian optimization. *arXiv preprint arXiv:1912.08601*, 2019.

Ramsey Faragher. Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]. *IEEE Signal processing magazine*, 29(5):128–132, 2012.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.