Introduction

This toolbox is intended for the efficient computation of linear dynamic problems of high dimension. These problems are solved by using a user selected Kalman filter (KF) algorithm. 1D, 2D and 3D data types are supported. This toolbox requires either the use of MATLAB or the open-source alternative GNU Octave (later referred simply as Octave). The only differences in using either of the two software with this toolbox are that some preliminary computations are faster to perform in MATLAB (such as Matèrn prior) and thus the overall speed is better in MATLAB. However, all the features are available for the Octave version and most of the selections should compute nearly identically in terms of speed. All operating systems are supported though testing has not been performed on Mac. For installation instructions, see the readme in GitHub.

This toolbox is primarily powered by ArrayFire functions and as such allows the use of either OpenCL, CUDA or CPU as the program "backend". This backend can be selected at runtime without the need to separately recompile the code. Separate functions are included to query the supported devices on each backend. Only one device can be used at a time, i.e. there is no multi-GPU/heterogeneous support. There are no backend limitations on either MATLAB or Octave. On Mac-based computers, however, CUDA is no longer supported.

All computations are performed in SINGLE (32-bit float) precision. However, all the input variables can be in double precision.

An overview of many of the methods presented can be found from [1] (`http://urn.fi/URN:ISBN:978-952-61-3815-2`).

As to what can be considered as "high dimensional" in KF depends on the computational resources. However, in general cases where the number of estimates (states) is over ¿ 10000 can be considered as high dimensional. Regular KF, however, can still function fine even in such cases. The primary thing that needs to be taken into account is the memory required by the error covariance(s) and the Kalman gain. For example, estimating a 128×128 image takes one gigabyte of memory on the device with the error covariance matrix alone ($16384 * 16384 * 4/1024^3 = 1$) on single precision.

Input data

The compulsory inputs from the user are the measurement data and the observation/system matrix. Related to these, the number of measurements for each time step is required as well as the number of unique cycles in the system matrix.

If the dynamic problem is defined with the following linear equation

$$y_k = H_k x_k, \tag{1}$$

then y_k are the measurements at time k and H_k is the observation/system matrix at time k (x_k is the estimate). The measurement data can be either real- or complex-valued, but has to be full (i.e. not sparse), and can be dynamic 1D, 2D or 3D data (with the corresponding estimates). The observation/system matrix can also be either real- or complex-valued, but it can also be sparse (also sparse complex). Note that the sparse matrices need to formatted as MATLAB/Octave sparse matrices. All sparse matrices will be converted to single precision format using compressed sparse row (CSR) by the toolbox. This process requires some additional memory on the host (global) side (i.e. device memory will only contain the single precision data).

Measurement data should be input into the variable `options.m0`. Observation/system matrix into the variable `options.H`. The estimates are saved in the variable `options.xt`. Number of measurements is input into `options.Nm` and the number of unique cycles to `options.matCycles`, see general settings for more info on these two.

Algorithms

Total of 12 different Kalman filter (KF) algorithms are available. However, not every algorithm supports every other feature present. In this section the equations and references for all the different algorithms are presented.

Regular (discrete) KF

This is the original KF algorithm [2, 3]

$$P_k^- = F_{k-1} P_{k-1}^+ F_{k-1}^T + Q_{k-1} \tag{2}$$
$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \tag{3}$$
$$x_k^- = F_{k-1} x_{k-1}^+ + G_{k-1} u_{k-1} = \text{a priori estimate} \tag{4}$$
$$x_k^+ = x_k^- + K_k(y_k - H_k x_k^-) = \text{a posteriori estimate} \tag{5}$$
$$P_k^+ = (I - K_k H_k) P_k^-, \tag{6}$$

where P_k^- is the a priori error covariance, F_{k-1} the state transition matrix, Q_{k-1} the process noise covariance, K_k is the Kalman gain, R_k the observation error covariance, x_k^- a prior estimate, G_{k-1} (optional) state transition matrix for (optional) u_{k-1} and P_k^+ the a posteriori error covariance.

The regular KF is the only algorithm that supports all features present in the toolbox.

Information filter

The information filter [4] is a modification of the regular KF. It is beneficial when the number of measurements per time step is much larger than the number of variables to be estimated. Though it is recommended that the number of variables to be estimated is not high (e.g. > 16384) as otherwise the computations will be quite slow. The equations for the information filter are the following

$$(P_k^-)^{-1} = Q_{k-1}^{-1} - Q_{k-1}^{-1} F_{k-1}((P_{k-1}^+)^{-1} + F_{k-1}^T Q_{k-1}^{-1} F_{k-1})^{-1} F_{k-1}^T Q_{k-1}^{-1} \tag{7}$$
$$(P_k^+)^{-1} = (P_k^-)^{-1} + H_k^T R_k^{-1} H_k \tag{8}$$
$$K_k = P_k^+ H_k^T R_k^{-1} \tag{9}$$
$$x_k^- = F_{k-1} x_{k-1}^+ + G_{k-1} u_{k-1} \tag{10}$$
$$x_k^+ = x_k^- + K_k(y_k - H_k x_k^-). \tag{11}$$

Information filter does not support consistency tests.

One step KF

One step KF [5] is another modification of the regular KF. No a priori covariances or estimates are computed, so it might save memory on some occasions. The equations are as follows

$$K_k = F_{k-1} P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \tag{12}$$
$$P_{k+1}^- = F_k P_k^- (F_k - K_k H_k)^T + Q_k \tag{13}$$
$$x_{k+1}^- = F_{k-1} x_k^- + K_k(y_k - H_k x_k^-). \tag{14}$$

One step KF does not support consistency tests, Kalman smoother or (split) complex data (referred as complex type 3).

Regular Ensemble Kalman filter

Ensemble Kalman filters (EnKF) are Monte Carlo approximations of the regular KF for high dimensional cases. Instead of using the full error covariance, the ensemble covariance is used instead. Total of five different EnKF algorithms are included with the first one being the original EnKF [6, 7]

$$i = 1, \ldots, N (\text{ensemble size}) \tag{15}$$
$$\bar{X}_1 = x_0 \tag{16}$$
$$\hat{x}_k^i = F_{k-1}\bar{X}_k^i + G_{k-1}u_{k-1} + w_k^i, \quad w_k^i \sim \mathcal{N}(0, Q_k) \tag{17}$$
$$X_k = \begin{bmatrix} \hat{x}_k^1 & \hat{x}_k^2 & \cdots & \hat{x}_k^N \end{bmatrix} \tag{18}$$
$$\hat{y}_k^i = y_k + v_k^i, \quad v_k^i \sim \mathcal{N}(0, R_k) \tag{19}$$
$$Y_k = \begin{bmatrix} \hat{y}_k^1 & \hat{y}_k^2 & \cdots & \hat{y}_k^N \end{bmatrix} \tag{20}$$
$$A_k = X_k - E(X_k)e_{1\times N} \tag{21}$$
$$P_k = \frac{A_k A_k^T}{N-1} \tag{22}$$
$$K_k = P_k H_k^T \left(H_k P_k H_k^T + R_k\right)^{-1} \tag{23}$$
$$X_k^a = X_k + K_k \left(Y_k - H_k X_k\right) \tag{24}$$
$$E(X_k) = \frac{1}{N}\sum_{i=1}^{N} \hat{x}_k^i \tag{25}$$
$$\bar{X}_{k+1} = E(X_k^a) \tag{26}$$

Note that the last step of $\bar{X}_{k+1}$ can also use the original ensembles (without computing the mean) for the next step (i.e. $\bar{X}_{k+1} = X_k^a$). This can be controlled with `options.useEnsembleMean`. This also applies to the other ensemble methods below.

Regular EnKF does not (currently) support consistency tests, steady state KF or Kalman smoother.

Alternate EnKF

This is an alternate formulation of the regular EnKF [8]. The Kalman gain is not explicitly computed here and thus some memory may be saved. Furthermore, $H_k A_k$ is precomputed.

$$P_k = \frac{H_k A_k \left(H_k A_k\right)^T}{N-1} + R_k \tag{27}$$
$$X_k^a = X_k + \frac{1}{N-1} A_k \left(H_k A_k\right)^T P_k^{-1} \left(Y_k - H_k X_k\right) \tag{28}$$

As with the regular EnKF, consistency tests, steady state KF and Kalman smoother are not supported. This filter should give exactly the same results as regular EnKF (not counting floating point errors).

Cholesky EnKF

The Cholesky EnKF [8] is more suitable when the number of measurements (and the number of estimates) are large. However, the observation error covariance R should be easily invertible or static. This method also takes advantage of the Cholesky decomposition. The equations are the following

$$\hat{x}_k^i = F_{k-1}\bar{X}_k^i + w_k^i, \quad w_k^i \sim \mathcal{N}(0, Q_k) \tag{29}$$
$$X_k = \begin{bmatrix}\hat{x}_k^1 & \hat{x}_k^2 & \cdots & \hat{x}_k^N\end{bmatrix} \tag{30}$$
$$\hat{y}_k^i = y_k + v_k^i, \quad v_k^i \sim \mathcal{N}(0, R_k) \tag{31}$$
$$Y_k = \begin{bmatrix}\hat{y}_k^1 & \hat{y}_k^2 & \cdots & \hat{y}_k^N\end{bmatrix} \tag{32}$$
$$D_k = Y_k - H_k X_k \tag{33}$$
$$A_k = X_k - E(X_k)e_{1\times N} \tag{34}$$
$$C_k = I + (H_k A_k)^T R_k^{-1} \frac{1}{N-1}(H_k A_k) \tag{35}$$
$$L_k L_k^T = \text{chol}(C_k) \tag{36}$$
$$Z_k = (H_k A_k)^T R_k^{-1} D_k \tag{37}$$
$$W_k = C_k^{-1} Z_k \tag{38}$$
$$M_k = R_k^{-1}\left(D_k - \frac{1}{N-1}(H_k A_k) W_k\right) \tag{39}$$
$$Z_k = (H_k A_k)^T M_k \tag{40}$$
$$X_k^a = X_k + \frac{1}{N-1} A_k Z_k \tag{41}$$
$$\bar{X}_{k+1} = E(X_k^a) \tag{42}$$

As with the regular EnKF, consistency tests, steady state KF and Kalman smoother are not supported. This filter should give exactly the same results as regular EnKF.

Square-root EnKF

This is a type of square-root EnKF [8] and is also a modification of the above Cholesky EnKF. This method is numerically more stable than the above. However, the observation noise covariance needs to be decomposed.

$$\hat{x}_k^i = F_{k-1}\bar{X}_k^i + w_k^i, \quad w_k^i \sim \mathcal{N}(0, Q_k) \tag{43}$$
$$X_k = \begin{bmatrix}\hat{x}_k^1 & \hat{x}_k^2 & \cdots & \hat{x}_k^N\end{bmatrix} \tag{44}$$
$$\hat{y}_k^i = y_k + v_k^i, \quad v_k^i \sim \mathcal{N}(0, R_k) \tag{45}$$
$$Y_k = \begin{bmatrix}\hat{y}_k^1 & \hat{y}_k^2 & \cdots & \hat{y}_k^N\end{bmatrix} \tag{46}$$
$$A_k = X_k - E(X_k)e_{1\times N} \tag{47}$$
$$S_k S_k^T = \text{chol}(R_k) \tag{48}$$
$$B_k = S_k^{-1} H_k A_k \tag{49}$$
$$P_k^{-1} = S_k^{-1}\left[I - \frac{1}{N-1} B_k \left(I + \frac{1}{N-1} B_k^T B_k\right)^{-1} B_k^T\right] S_k^{-T} \tag{50}$$
$$D_k = Y_k - H_k X_k \tag{51}$$
$$X_k^a = X_k + \frac{1}{N-1} A_k (H_k A_k)^T P_k^{-1} D_k \tag{52}$$
$$\tag{53}$$

As with the regular EnKF, consistency tests, steady state KF and Kalman smoother are not supported. This filter should give exactly the same results as regular EnKF.

Ensemble transform Kalman filter

Ensemble transform KF (ETKF) [9] is a type of square root filter [10] where the ensemble mean is updated instead of the complete ensemble. This is computationally efficient as long as the inverse of the observation error covariance is easy to compute. ETKF supports both a high number of estimates as well

as high number of measurements. The orthogonal matrix is assumed to be identity and is thus omitted from the equations below

$$[U_k, \Sigma_k, V_k] = \text{SVD}\left(\rho\,(N-1)\,I + (H_k X_k)^T\, R_k^{-1}\,(H_k X_k)\right) \tag{54}$$
$$M_k = \sqrt{N-1} U_k \Sigma_k^{-1/2} U_k^T \tag{55}$$
$$\bar{w}_k = U_k \Sigma_k^{-1} U_k^T\,(H_k X_k)^T\, R_k^{-1}\left(y_k - H_k \bar{X}_k\right) \tag{56}$$
$$\bar{W}_k = (\bar{w}_k, \ldots, \bar{w}_k) \tag{57}$$
$$X_k^a = \bar{X}_k + A_k\left(\bar{W}_k + M_k\right). \tag{58}$$

Consistency tests and steady state KF are not supported with ETKF. Note that the above equations follow [10] rather than the original article [9].

Error-subspace transform Kalman filter

Error-subspace transform KF (ESTKF) [10, 11] is a modification of the ETKF. ESTKF should be computationally slightly more efficient than the ETKF and also supports high dimensional estimates and measurements. As with ETKF, the orthogonal matrix is assumed to be the identity.

$$A_{ij} = \begin{cases} 1 - \frac{1}{N}\frac{1}{\frac{1}{\sqrt{N}}+1} & \text{for } i = j, i < N \\ -\frac{1}{N}\frac{1}{\frac{1}{\sqrt{N}}+1} & \text{for } i \neq j, i < N \\ -\frac{1}{\sqrt{N}} & \text{for } i = N \end{cases} \tag{59}$$
$$L_k = X_k A \tag{60}$$
$$[U_k, \Sigma_k, V_k] = \text{SVD}\left(\rho\,(N-1)\,I + (H_k L_k)^T\, R_k^{-1}\,(H_k L_k)\right) \tag{61}$$
$$M_k = \sqrt{N-1} U_k \Sigma_k^{-1/2} U_k^T A^T \tag{62}$$
$$\bar{w}_k = U_k \Sigma_k^{-1} U_k^T\,(H_k L_k)^T\, R_k^{-1}\left(y_k - H_k \bar{X}_k\right) \tag{63}$$
$$\bar{W}_k = (\bar{w}_k, \ldots, \bar{w}_k) \tag{64}$$
$$X_k^a = \bar{X}_k + L_k\left(\bar{W}_k + M_k\right) \tag{65}$$

Consistency tests and steady state KF are not supported with ESTKF.

Dimension reduction Kalman filter

Dimension reduction KF (DRKF) [12] works similarly to the regular KF, except that the estimation error covariance is computed in a reduced dimension, Ψ_k^+ (as well as an initial a posteriori estimate, α_k). Custom prior covariances P_r can be used, but the default is the Gaussian prior used in the original article [12]. This algorithm works in cases with large number of estimates, however, large number of measurements might cause memory issues.

$$x_k^- = F_{k-1} x_{k-1}^+ + G_{k-1} u_{k-1} \tag{66}$$
$$A_k A_k^T = \text{chol}(\Psi_{k-1}^+) \tag{67}$$
$$B_k = F_{k-1} P_r A_k \tag{68}$$
$$(P_k^-)^{-1} P_r = Q_{k-1}^{-1} P_r - Q_{k-1}^{-1} B_k (B_k^T Q_{k-1}^{-1} B_k + I)^{-1} B_k^T Q_{k-1}^{-1} P_r \tag{69}$$
$$\Psi_k^+ = \left((H_k P_r)^T R_k^{-1} (H_k P_r) + P_r^T (P_k^-)^{-1} P_r\right)^{-1} \tag{70}$$
$$\alpha_k = \Psi_k^+ (H_k P_r)^T R_k^{-1} (y_k - H_k x_k^-) \tag{71}$$
$$x_k^+ = x_k^- + P_r \alpha_k \tag{72}$$
$$P_k^+ = P_r \Psi_k^+ P_r^T. \tag{73}$$

DRKF does not support consistency tests.

Conjugate gradient Kalman filter

Conjugate gradient KF (CGKF) [13] is a modification of the regular KF that uses the conjugate gradient to approximate the high-dimensional components, such as the estimate error covariances and Kalman gain. As such, it is especially suited for cases with both high number of estimates as well as observations. However, it is an approximation of the KF and may not yield as optimal results. In this toolbox, the CGKF is implemented in such a way as to reduce the storage of any necessary variables. This is achieved by computing mainly matrix-vector multiplications instead of matrix-matrix multiplications. Furthermore, the eigenvalue decomposition is replaced, due to technical constraints, with singular value decomposition. CGKF is defined with the following equations (note that the below equations also contain the CG and Lanczos iterations)

$$x_k^- = F_{k-1}x_{k-1}^+ + G_{k-1}u_{k-1} \tag{74}$$
$$P_k^- = F_{k-1}B_{k-1}B_{k-1}^T F_{k-1}^T + Q_{k-1} \tag{75}$$
$$A = H_k P_k^- H_k^T + R_k \tag{76}$$
$$b = y_k - H_k x_k^- \tag{77}$$
$$r_0 = b - Af^0 \tag{78}$$
$$p_0 = r_0 \tag{79}$$
$$n = 1, \ldots, N_{\text{iter}} \tag{80}$$
$$\gamma_{n-1} = \frac{r_{n-1}^T r_{n-1}}{p_{n-1}^T A p_{n-1}} \tag{81}$$
$$f_n = f_{n-1} + \gamma_{n-1}p_{n-1} \tag{82}$$
$$r_n = r_{n-1} - \gamma_{n-1}Ap_{n-1} \tag{83}$$
$$\beta_n = -\frac{r_n^T r_n}{r_{n-1}^T r_{n-1}} \tag{84}$$
$$p_n = r_n - \beta_n p_{n-1} \tag{85}$$
$$P_k = \left\{p^i\right\}_{i=0}^{n-1} \tag{86}$$
$$D_k = P_k^T A P_k \tag{87}$$
$$C_k = P_k D_k^{-\frac{1}{2}} \tag{88}$$
$$x_k^+ = x_k^- + P_k^- H_k^T f_{N_{\text{iter}}} \tag{89}$$
$$A = P_k^- - P_k^- H_k^T C_k C_k^T H_k P_k^- \tag{90}$$
$$w_0 = \frac{\hat{w}_0}{||\hat{w}_0||} \tag{91}$$
$$\alpha_0 = w_0^T A w_0 \tag{92}$$
$$\hat{w}_1 = Aw_0 - \alpha_0 w_0 \tag{93}$$
$$n = 1, \ldots, N_{\text{iter}} \tag{94}$$
$$\nu_n = ||\hat{w}_n|| \tag{95}$$
$$w_n = \frac{\hat{w}_n}{\nu_n} \tag{96}$$
$$u_n = Aw_n - \nu_n w_{n-1} \tag{97}$$
$$\alpha_n = \nu_n^T u_n \tag{98}$$
$$\hat{w}_{n+1} = u_n - \alpha_n w_n \tag{99}$$
$$W_k = \left\{w^i\right\}_{i=0}^{n-1} \tag{100}$$
$$T_k = W_k^T A W_k \tag{101}$$
$$[U_k, \Sigma_k, V_k] = \text{SVD}\left(T_k\right) \tag{102}$$
$$B_k = W_k U_k \Sigma_k^{\frac{1}{2}}. \tag{103}$$

It is possible to (optionally) force orthogonalization of p_n and $\hat{w}$ by setting `options.forceOrthogonalization` to true. This increases the computation time, but increases accuracy. If set to false, the estimates might

even fail to compute.

Steady state KF is not currently supported with CGKF. Note that while KS is supported, it will most likely not give much of any benefit to the estimates.

Conjugate gradient variational Kalman filter

The conjugate gradient variational KF (CGVKF) is very similar to the CGKF. However, the estimates are obtained by minimizing the log posterior distribution rather than computing certain parts of the KF with CG.

$$x_k^- = F_{k-1}x_{k-1}^+ + G_{k-1}u_{k-1} \tag{104}$$
$$(P_k^-)^{-1} = Q_{k-1}^{-1} - Q_{k-1}^{-1}F_{k-1}B_k\left(I + B_k^T F_{k-1}^T Q_{k-1}^{-1}F_{k-1}B_k\right)^{-1} B_k^T F_{k-1}^T Q_{k-1}^{-1} \tag{105}$$
$$A = H_k R_k^{-1} H_k^T + (P_k^-)^{-1} \tag{106}$$
$$b = H_k R_k^{-1} y_k + (P_k^-)^{-1} x_k^- \tag{107}$$
$$r_0 = b - Af^0 \tag{108}$$
$$p_0 = r_0 \tag{109}$$
$$n = 1, \ldots, N_{\text{iter}} \tag{110}$$
$$\gamma_{n-1} = \frac{r_{n-1}^T r_{n-1}}{p_{n-1}^T A p_{n-1}} \tag{111}$$
$$f_n = f_{n-1} + \gamma_{n-1}p_{n-1} \tag{112}$$
$$r_n = r_{n-1} - \gamma_{n-1}Ap_{n-1} \tag{113}$$
$$\beta_n = -\frac{r_n^T r_n}{r_{n-1}^T r_{n-1}} \tag{114}$$
$$p_n = r_n - \beta_n p_{n-1} \tag{115}$$
$$P_k = \left\{p^i\right\}_{i=0}^{n-1} \tag{116}$$
$$D_k = P_k^T A P_k \tag{117}$$
$$B_k = P_k D_k^{-\frac{1}{2}} \tag{118}$$
$$x_k^+ = f_{N_{\text{iter}}}. \tag{119}$$

It is possible to (optionally) force orthogonalization of p_n by setting `options.forceOrthogonalization` to true. This increases the computation time, but increases accuracy. If set to false, the estimates might even fail to compute though the CGVKF should be numerically more stable than CGKF.

Steady state KF is not currently supported with CGVKF. Note that while KS is supported, it will most likely not give much of any benefit to the estimates.

Spectral Kalman filter

The spectral KF (SpecKF) [14] is a modification of the original KF, where the estimation error covariances are not explicitly computed, but rather versions that are multiplied with the observation/system matrix, i.e. the cross covariances. This algorithm is beneficial in cases where the number of observations/measurements is much lower than the number of estimates. It is also beneficial for random walk situations, otherwise an error term is required. The formulas are

$$T_k = F_{k-1}P_{k-1}^F + Q_{k-1}H_k^T \tag{120}$$
$$K_k = T_k\left(H_k T_k + R_{k-1}\right)^{-1} \tag{121}$$
$$x_k^- = F_{k-1}x_{k-1}^+ + G_{k-1}u_{k-1} \tag{122}$$
$$x_k^+ = x_k^- + K_k\left(y_k - H_k x_k^-\right) \tag{123}$$
$$P_k^F = \left(I - K_k H_k^T\right)\left(2F_{k-1}P_{k-1}^F - F_{k-1}P_{k-1}^H + Q_{k-1}F_{k-1}^T H_{k-1}^T\right) + \epsilon_k \tag{124}$$
$$K_k = \left(I - K_k H_k\right)T_k, \tag{125}$$

where ϵ_k is the error term from $(I - F_{k-1})^2$, i.e. $\epsilon_k \approx (I - F_{k-1})^2$ and should be near zero.

Note that the spectral KF is still work-in-progress, i.e. the state transition matrix F is not supported. The features of specKF are also more limited as no complex type 3 is supported, or KS, or consistency tests. Due to the lack of support for F, the current SpecKF is more accurately the HiKF [15].

Spatial regularization

There are a total of four different (optional) spatial regularization methods. These can be selected with `options.regularization`, where 0 means no regularization, 1 means augmented KF (AKF), 2 means spatial prior KF (SPKF), 3 means denoised KF (e.g. TVKF) and 4 is another type of denoising that has an inherent positivity constraint.

Augmented KF

In AKF [16, 17], the observation/measurement dimension is augmented with the spatial regularization, i.e. the regularization functions as another set of observations/measurements. This means that both the measurement vector y and observation/system matrix H are augmented:

$$\bar{y}_t = \begin{pmatrix} y_t \\ \beta\kappa L x_{\text{ref}} \end{pmatrix}$$
$$\bar{H}_t = \begin{pmatrix} H_t \\ \beta\kappa L \end{pmatrix}.$$

L is the regularization matrix, κ an optional weighting matrix that can be e.g. an anatomical weight, x_{ref} is the reference image, e.g. an anatomical image and β is the regularization parameter. The observation noise covariance is modified as

$$\bar{R}_t = \begin{pmatrix} R_t & 0 \\ 0 & \sigma_R I \end{pmatrix}, \tag{126}$$

where σ_R is the variance of the augmented observations (default value is 1). AKF can be combined with any algorithm, but due to augmenting the observation dimension with the estimation (state) dimension it can be very memory intensive.

Spatial prior KF

SPKF [18, 19] is somewhat similar to the AKF, but instead of augmenting any of the matrices/vectors, the state transition matrix F is used instead. This also means that when using SPKF no other state transition matrix can be used. In SPKF, the state transition is defined as

$$F_k = \left(I + \beta Q_k (\kappa L)^T \kappa L\right)^{-1}. \tag{127}$$

The matrices L and κ are defined exactly the same for SPKF and AKF. The process noise covariance is also modified as

$$\bar{Q}_k = Q_k F. \tag{128}$$

SPKF can be beneficial when compared to the AKF when the process noise covariance is constant as the dimensions are smaller than in AKF. However, the computation of the matrix F requires the matrix to be initially full after which small values are removed and the matrix turned into sparse matrix. This means that system memory requirements can be initially much higher than the final device memory requirements. The inversion of matrix F is precomputed, but can be slow if the matrix is large. As SPKF modified the state transition matrix, it can be used with any algorithm, but cannot be combined with other state transition matrices.

Denoising type 1 (TVKF)

Denoising type 1 [16, 1] includes two different denoising methods. Unlike the previous two regularization methods, these denoisers only denoise the a posteriori estimates before the next time step.

The first denoising method of denoising type 1 is based on the interior point method [20]

$$u_k^{s+1} = u_k^s - \beta P_k^+ \nabla \Psi(u_k^s), \tag{129}$$

where u_k^s is the a posteriori estimate x_k^+ at the denoising iteration s. The a posteriori error covariance P_k^+ can be optionally used to weight the denoising (this can be toggled with `options.includeCovariance`). This weighting, however, is not available with ensemble based filters. Finally, $\nabla\Psi(u_k^s)$ is the gradient of the chosen prior or Lu_k^s. The prior can be chosen with `options.prior`.

A total of three different priors are available when using a priori data (e.g. anatomical reference) or two when no a priori information is available. These include two different reference weighting methods for total variation (TV) [16, 21] (prior 1 and 2), asymmetric parallel level sets (APLS) [21] (pior 3) and smoothed anisotropic TV (SATV) [22] (prior 4). The TV methods and APLS are identical when no reference data is used and SATV does not support any reference data. Regularization matrix L is constructed exactly the same as with AKF (see the relevant section for details) (prior 7). The positive smoothing parameter for TV, α, is controlled with `options.TVsmoothing`. SATV ϕ value is controlled with `options.SATVPhi` and η value for APLS with `options.eta`, see the relevant articles for details.

For TV type 1, the prior is computed as follows when optional weighting is used

$$\Psi(u_k^s) = \int_\Omega \sqrt{(\nabla u_k^s)^T D(\nabla u_k^s) + \alpha}\, dx \tag{130}$$

where $u_k := \hat{x}_k^+$ is the a posteriori estimate at time k, α is a TV smoothing parameter (to guarantee differentiability), Ω is the whole image domain, and

$$D = I - \lambda\eta\eta^T \tag{131}$$

is a weighting matrix, implementing a structured TV using the parallel level sets approach [23, 21]. Without the weighting matrix (i.e. $D = I$), the functional would be isotropic TV. λ in (131) is defined as

$$\lambda = 1 - \exp\left(-\frac{\|\nabla\tilde{x}_{\text{ref}}\|^2}{C^2}\right) \tag{132}$$

and η as

$$\eta = \begin{cases} \frac{\nabla\tilde{x}_{\text{ref}}}{\|\nabla\tilde{x}_{\text{ref}}\|} & \text{when } \|\nabla\tilde{x}_{\text{ref}}\| \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{133}$$

Forward difference is used to compute the gradients. Reference image based weighting is toggled with `options.useAnatomical`. Separate reference data can be, optionally, used for the real and imaginary parts, toggled with `options.complexRef`.

The second denoising method for denoising type 1 uses total generalized variation [24] and is based on [25]. This is used when prior 8 is selected. This method also does not support the use of the a posteriori covariance as a weighting matrix. Four values can be controlled, the regularization parameters `options.lambda1` and `options.lambda2`, relaxation parameter `options.relaxationParameter` and the proximal value `options.proximalValue`.

The number of denoising iterations is controlled with `options.nIterDenoise`.

Denoising type 2

This method does not support any priors and uses the regularization matrix L. It is based on [26, 27]. The regularization matrix can be constructed just as in AKF. For details on the method, see the previous references. The main difference of this method, when compared to the previous one, is that this one enforces non-negativity on the estimate and can thus be more optimal for cases that require non-negativity. For cases where negative estimates appear, this method is not recommended. The step size can be controlled with `options.gamma`.

Matèrn covariance

It is possible to construct the process noise covariance Q by using the covariance functions from the Matèrn family [28, 29]. The covariance function can be constructed for $\nu = \frac{5}{2}, \nu = \frac{3}{2}, \nu = \frac{1}{2}$ and $\nu = \infty$. The process noise covariance is formulated as

$$Q_k = C_s - \exp(-2\lambda\Delta t)C_s, \tag{134}$$

where λ is a positive constant and Δt the difference between adjacent time steps. C_s is the Matèrn covariance where $s = \nu$. The supported C_s matrices are thus

$$C_{\nu=\infty} = \sigma^2 \exp\left(-\frac{r^2}{2l^2}\right) \tag{135}$$

$$C_{\nu=\frac{1}{2}} = \sigma^2 \exp\left(-\frac{r}{l}\right) \tag{136}$$

$$C_{\nu=\frac{3}{2}} = \left(1 + \frac{\sqrt{3}r}{l}\right) \exp\left(-\frac{\sqrt{3}r}{l}\right) \tag{137}$$

$$C_{\nu=\frac{5}{2}} = \left(1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}r}{l}\right), \tag{138}$$

where l is the length scale of the process, σ^2 is the variance and r the Euclidean distance between two points. λ, l and σ can be input by the user while r is computed by the software.

Fading memory

Fading memory KF [30] allows the filter to "forget" past measurements and is defined as

$$P_k^- = \alpha F_{k-1} P_{k-1}^+ F_{k-1}^T + Q_{k-1}. \tag{139}$$

Fading memory KF is supported with the regular KF only. Note that the same parameter controls the covariance inflation, ρ, of ETKF and ESTKF. Both values can be set with `options.fadingAlpha`.

Covariance inflation

Covariance inflation is supported by ETKF and ESTKF. As with the fading memory, this can be with `options.fadingAlpha`.

Steady state KF and preiteration

Steady state KF is available for regular KF. In steady state KF, the Kalman gain matrix is precomputed such that the KF steps are repeated until the Euclidean norm of the previous diagonal elements minus the current diagonal elements of the a posteriori covariance is below a predetermined threshold (default value is 0.001, `options.steadyThreshold`). Once the steady state gain is obtained, the actual filter steps are computed. However, the error covariances and Kalman gain are no longer computed. Only the a priori and a posteriori estimates are then computed. Toggle the steady state KF with `options.steadyKF`.

It is possible to preiterate the initial value of the KF estimate at the same time. In this case, the KF estimates are computed normally while computing the gain and error covariances. The final KF estimate is then used as the initial value. This can be toggled with `options.computeInitialValue`. Note that in steady state KF, the steady state computations proceed normally in time as in regular KF, i.e. the computations use the same observation/system matrices and any other matrices that may vary with time and also increments the time index normally. As such, an approximate steady state can be reached, at best, when time-varying matrices are used. If all input matrices vary significantly over time, even approximate steady state will most likely never be reached. Due to the time-varying nature of the steady state computations, the at the same time computed KF initial value, if selected, will represent the KF value at the time step where the steady state was reached.

If the steady state has not been reached when the total number of time steps selected is reached, then no steady state filter will be used and the KF computations are computed normally. The previously computed values, however, will be used as initial values for the actual filter steps.

Preiterate the error covariances and Kalman gain

As an alternative to the steady state KF, it is possible to instead only preiterate the error covariances and Kalman gain with a predetermined number of iterations (set with `options.covIter`). As with the steady state KF, the initial values can be computed at the same time. This preiteration works exactly the same way as the steady state KF, but rather than continue the KF steps until convergence or the selected number of time steps have been computed, the user can select the number of preiterations to perform. However, the number of iterations cannot exceed the total number of time steps available.

Kalman smoother

Kalman smoother is available for select algorithms (selected with `options.useKS`). These include the regular KF, information filter, ETKF, ESTKF, DRKF and CGKF. Regular KF, information filter and CGKF use the Rauch-Tung-Striebel (RTS) fixed-interval smoother [31, 32] while DRKF uses a small modification of it. ETKF and ESTKF use the ensemble Kalman smoother (EnKS) [11]. The RTS smoother is defined as

Initialize:

$$x_T = x_T^+ \tag{140}$$
$$P_T = P_T^+ \tag{141}$$

For $k = T, \ldots, T_{\text{lag}}$

$$K_{s,k-1} = P_{k-1}^+ F_k^T (P_k^-)^{-1} \tag{142}$$
$$x_{k-1} = x_{k-1}^+ + K_{s,k-1}(x_k - x_k^-) \tag{143}$$
$$P_{k-1} = P_{k-1}^+ - K_{s,k-1}\left(P_k^+ - P_k\right) K_{s,k-1}^T. \tag{144}$$

Note that the smoother error covariance P_{k-1} is not computed by the toolbox and thus cannot be obtained at the moment. For DRKF, the above equations become

$$x_{k-1} = x_{k-1}^+ + P_r \Psi_k^+ P_r^T F_k^T (P_k^-)^{-1} \left(x_k - x_k^-\right). \tag{145}$$

EnKS for ETKF is defined as

$$G_k = \mathbf{1}_{(N)} + \rho\left(I - \mathbf{1}_{(N)}\right)\left(\bar{W}_k + M_k\right) \tag{146}$$
$$X_{k|k} = X_k \tag{147}$$
$$X_{i|k} = X_{i|i} \prod_{j=i+1}^{k} G_j \tag{148}$$

i = current state, j = latest time for which observations are taken into account.

and for ESTKF as [11]

$$G_k = \mathbf{1}_{(N)} + \rho A\left(\bar{W}_k + M_k\right) \tag{149}$$
$$X_{k|k} = X_k \tag{150}$$
$$X_{i|k} = X_{i|i} \prod_{j=i+1}^{k} G_j \tag{151}$$

i = current state, j = latest time for which observations are taken into account.

In both cases, $\mathbf{1}_{(N)}$ is a matrix containing N^{-1} in every element ($\bar{X}_k = \mathbf{1}_{(N)} X_k$). Note that $\prod$ refers to matrix multiplication in this case.

With all smoothers, it is possible define the number of time steps in each (fixed) interval (specified by `options.nLag`). This value determines the number of time steps from where the smoother is computed. The smoothing is started from the current time and then decreased until the number specified by the lag has been reached. Since multiple smoothing steps can be performed in series, it is possible to skip certain number of time steps (specified by `options.sSkip`) before the smoothing step is next performed. Note that if adjacent smoothing steps overlap, the previous smoothed steps are replaced by the current ones. For example, if a total 30 time steps are computed, with a lag/interval of 10 and skip of 5, the smoothing steps are performed at time steps (when starting from 1) 10, 15, 20, 25 and 30. At each of these steps, a total of 10 smoothed estimates are computed. For example at time 25, the smoothed estimates are computed for 15-24 while at time 30 these are computed for 20-29. The latter smoothed estimates overwrite the previous ones.

Furthermore, it is possible to conserve memory in the smoothing step and use steady state KS (not applicable to EnKS) (specified by `options.steadyKS`). In this case, only one smoothing gain is computed from the latest P_{k-1}^+, F_k and P_k^- matrices. The same gain is then used in every KS step.

To speed up the computations, it is also possible to approximate the a priori covariance P_k^- in the smoothing as a diagonal matrix (specified by `options.approximateKS`). This reduces accuracy, but speeds up the computations as the inverse is efficient to compute.

Smoothed estimates are stored in `options.xs`.

Consistency tests

When running KF, the selected process and/or observation noise covariances may not be optimal. In such a case, the *consistency tests* [33] can be used to determine whether the filter is performing optimally or not. In this toolbox, the normalized innovation squared (NIS) (sometimes called normalized error square) test, autocorrelation (whiteness) test and Bayesian p-test are available [34]. NIS and autocorrelation tests can be selected without the p-test, but p-test cannot be run without NIS and autocorrelation tests. The consistency tests can be enabled with `options.computeConsistency`, while p-test needs to be enabled with `options.computeBayesianP`. In all cases, the tests are based on the KF innovation

$$\nu_k = y_k - H_t \hat{x}_k^- . \tag{152}$$

The criteria for filter consistency and approximate optimality are that the innovation has to be approximately white and zero-mean, while also having magnitude commensurate to the innovation covariance

$$E_k = H_k^T P_k^- H_k^T + R_k. \tag{153}$$

Time-average NIS test, which can be used to examine the whiteness of the current model, is computed with

$$\hat{\epsilon}_k = \frac{1}{N_t} \sum_{k=1}^{N_t} \nu_k^T E_k^{-1} \nu_k, \tag{154}$$

where N_t is the number of time steps included. $L\hat{\epsilon}_k$ has a chi-square density distribution with $N_t M$ degrees of freedom, if ν_t is a zero-mean, white noise process [35] and M is the number of measurements/observations at time k. For chi-square distribution with D degrees of freedom, the 95% confidence intervals are obtained from

$$\chi_D^2(0.975) = \frac{1}{2}\left(1.96 + \sqrt{2D-1}\right)^2 \tag{155a}$$

$$\chi_D^2(0.025) = \frac{1}{2}\left(-1.96 + \sqrt{2D-1}\right)^2 . \tag{155b}$$

The time-average NIS should then have values between (155a) and (155b).

Like with NIS, the time-average autocorrelation is computed and should be within the predetermined confidence interval (with values closest to zero indicating more optimal parameters). The time-average autocorrelation for υ steps apart is defined as

$$\rho_\upsilon = \frac{1}{\sqrt{N_t}} \sum_{t=1}^{N_t} \nu_t^T \nu_{t+\upsilon} \left(\sum_{t=1}^{N_t} \nu_t^T \nu_t \sum_{t=1}^{N_t} \nu_{t+\upsilon}^T \nu_{t+\upsilon} \right)^{-\frac{1}{2}} . \tag{156}$$

For 95% acceptance interval, we have [35]

$$\rho_\upsilon \in \left[-\frac{1.96}{\sqrt{N_t}}, \frac{1.96}{\sqrt{N_t}} \right] . \tag{157}$$

The Bayesian p-test is similar to NIS, but is more demanding to compute. However, it can in some cases detect filter inconsistency better than the NIS test. Bayesian p-test is defined as

$$\hat{\epsilon}_{p,k} = \left(y - H_k x_k^+\right)^T \left(H_k^T P_k^+ H_k^T + R_k\right)^{-1} \left(y - H_k x_k^+\right) . \tag{158}$$

The user can determine the number of preiteration steps to perform before the tests are performed (`options.consistencyIter`), the number of time steps, N_t, in the tests (`options.consistencyLength`), and the step size in autocorrelation (`options.stepSize`).

Sliding window KF

The KF estimates can also be computed in a sliding window manner. In sliding window KF, measurements from N_w time steps are included all times totaling $N_w M$ measurements. After each time step, M elements are removed from the end of the measurement vector y and new M number of measurements are added at the beginning of y. In sliding window KF, measurements from multiple time steps are included with repeated measurements rather than use unique measurements at each time step. The observation/system matrix is also constructed similarly. The number of sliding time steps is controlled with `options.window`. For example, when window is 3, the first KF time step would contain measurements from time steps 1, 2 and 3, the next KF time step measurements from time steps 2, 3 and 4, and so on.

Kinematic model

While the user can input their own state transition matrix, the software does include one built-in state transition matrix. This is a first order kinematic model [36, 37] where the evolution models are constructed as

$$x_k = x_{k-1} + \dot{x}_{k-1}\Delta T + \frac{1}{2}\Delta T^2 w_k \tag{159}$$

$$\dot{x}_k = \dot{x}_{k-1} + \Delta T w_k, \tag{160}$$

where ΔT (adjusted with `options.deltaT`) is the difference between the time steps. The state transition matrix thus becomes

$$F = \text{blockdiag}\begin{pmatrix} I & \Delta T \\ 0 & I \end{pmatrix}, \tag{161}$$

with the following state transition matrix for the noise covariance

$$D = \begin{pmatrix} \frac{1}{2}\Delta T^2 & 0 & \cdots & 0 \\ 0 & \frac{1}{2}\Delta T^2 & 0 & \cdots \\ \vdots & 0 & \ddots & \ddots \\ \ddots & \ddots & \ddots & \frac{1}{2}\Delta T^2 \\ \Delta T & 0 & \cdots & 0 \\ 0 & \Delta T & 0 & \ddots \\ \vdots & \ddots & \ddots & \ddots \\ & \ddots & \ddots & \Delta T \end{pmatrix}. \tag{162}$$

The process noise covariance is thus $\hat{Q}_k = DQ_kD^T$.

When using the kinematic model, the "speed" parameters are output as well, i.e. the dimension of the estimate is doubled. Note that using the kinematic model with algorithms using the inverse process noise covariance might fail due to singular $\hat{Q}_k$.

Adjustable parameters

This section explains the various adjustable parameters that were not explained in the previous section, such as different regularization matrices or the different types of complex data estimates available.

Complex data estimates

For complex input data, there are three different ways to compute the estimates. This can be controlled with the variable `options.complexType`. If you are using real-valued data, set this to 0. With real-valued data, there is nothing special in computing the estimates. The first type of complex data estimates is achieved by setting `options.complexType = 1`. In this case, only the a posteriori estimate computations are different between real and imaginary data. For example, with regular KF, all the Kalman gain and error covariances are the same for the real and imaginary part. Also the input matrices H, F, Q and R should all be real-valued. This is computationally the fastest way to compute complex-valued estimates. There are some differences between different algorithms, but the key difference of complex type 1 is that as much as possible computations and matrices are shared between the real and imaginary parts. All input matrices need to be real-valued when using complex type 1.

When `options.complexType = 2`, the situation differs in that now the real and imaginary parts are computed completely separately. That is, no data, computations or error covariance or Kalman gain matrices are shared between the two. H, F, Q and R can be either real-valued or complex-valued. In the case that they are complex, the real parts will be used for the computation of the real-valued estimates while the imaginary parts will be used for the computation of the imaginary-valued estimates. If the matrices are real-valued, the same matrices will be used in the estimation of the real and imaginary parts.

The last method for computing the complex estimates is selected with `options.complexType = 3`. In this case, the real and imaginary parts are split into separate components such that all the input data is actually real-valued. For example, the following forms of the input matrices are used

$$\hat{y}_k = \begin{pmatrix} \mathrm{Re}(y_k) \\ \mathrm{Im}(y_k) \end{pmatrix}, \qquad \hat{H}_k = \begin{pmatrix} \mathrm{Re}(H_k) & -\mathrm{Im}(H_k) \\ \mathrm{Im}(H_k) & \mathrm{Re}(H_k) \end{pmatrix},$$
$$\hat{x}_k = \begin{pmatrix} \mathrm{Re}(x_k) \\ \mathrm{Im}(x_k) \end{pmatrix}, \qquad \hat{v}_k = \begin{pmatrix} \mathrm{Re}(v_k) \\ \mathrm{Im}(v_k) \end{pmatrix}.$$

The complex data is thus computed as real-valued, but behaves as complex. $\hat{H}_k$ is not explicitly created, but rather the real and imaginary components are stored independently and the matrix/vector multiplications are done as if $\hat{H}_k$ would have been formed. The input matrices need to be complex-valued, as the splitting is done by the software itself.

Regularization matrix

The regularization matrix L for AKF, SPKF and denoising types 1 and 4 can take five different forms. Alternatively, the user can input their own (full or sparse) regularization matrix. The form of regularization matrix can be selected with `options.Ltype`, where 0 means 1D (forward) difference matrix, 1 2D first order difference matrix, 2 2D second order difference matrix, 3 combination of first and second order 2D difference matrices, 4 3D first order difference matrix and 5 a custom user-input matrix. The strength of the second order 2D difference matrix with 3 can be controlled with `options.Lscale`. For example, 2D first order difference matrix is constructed as

$$D = \begin{pmatrix} -2 & 1 & 0 & \cdots & 1 & 0 & \cdots & \\ 0 & -2 & 1 & 0 & \cdots & 1 & 0 & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix}. \tag{164}$$

Weighting the regularization matrix

It is possible to weight the regularization matrix. This can be toggled with `options.augType`, where 0 means no weighting (L is used as-is), 1 means that κ is defined as

$$\kappa = \kappa_x + \kappa_y \tag{165}$$

where κ_x is

$$\kappa_x = \exp\left(-\frac{|\tilde{x}_{\mathrm{ref},j} - \tilde{x}_{\mathrm{ref},j-1}|}{C}\right),$$

where C is the edge threshold parameter (`options.C`). 2 defines the weighting as with denoising type 1 (i.e. the matrix D is used).

Reference image based weighting is toggled with `options.useAnatomical`. Separate reference data can be, optionally, used for the real and imaginary parts, toggled with `options.complexRef`.

Regularization parameter for all regularization methods

Separate regularization parameters can be used for the real and imaginary parts. They are controlled, respectively with `options.beta` and `options.betaI`.

Ensemble filter specific settings

The ensemble size can be controlled with `options.ensembleSize`. Covariance inflation for ETKF and ESTKF is controlled with `options.fadingAlpha`. The ensemble mean can be used for the next a priori estimate when selecting `options.useEnsembleMean` to true otherwise the full ensemble is used.

DRKF specific settings

In DRKF, the user can input their own prior covariance matrix. Whether the built-in Gaussian covariance is used can be toggled with `options.useCustomCov` and the custom covariance can be input to `options.Sigma`. It is recommended to use a full matrix here. When using the built-in covariance, the dimensions can be adjusted with `options.covDimX`, `options.covDimY` and `options.covDimY` (the matrix will thus be of size covDimX*covDimY*covDimZ×covDimX*covDimY*covDimZ). The Gaussian covariance is constructed as

$$\Sigma_{i,j} = \sigma^2 \exp -\frac{r^2}{2l^2}, \tag{166}$$

where σ^2 is the variance, l the correlation length and r the Euclidean distance between pixels. The first two values can be selected with `options.reducedCovSigma` and `options.reducedCorLength`, respectively.

Regardless of the covariance chosen, the prior covariance P_r is constructed as follows

$$[U, S, V] = \text{SVD}(\Sigma) \tag{167}$$

$$P_r = U_r S_r^{-\frac{1}{2}}, \tag{168}$$

where U_r and S_r use the r leading singular values. This number can be controlled with `options.reducedBasisN` and is the number of basis functions included. The final P_r matrix is formed by linearly interpolating each column of P_r to the target estimation dimension.

CGKF specific settings

For CGKF, the number of CG iterations can be controlled (`options.cgIter`) as well as the stopping threshold of the CG iterations (`options.cgThreshold`). The iterations are thus performed at most the number of times as specified in `options.cgIter`, but can be stopped earlier if the threshold goes below `options.cgThreshold`. Furthermore, it is possible to force orthogonalization of certain variables as specified above in the algorithm section. This increases accuracy, but increases computation costs.

Using 3D or pseudo-3D data

Both 3D and pseudo-3D data is supported. By default, 3D data is assumed if the output dimension `options.Nz` is bigger than one, but if `options.use3D` is toggled as false, the data is assumed to be pseudo-3D instead. Pseudo-3D data means that the data consists of `options.Nz` number of independent 2D slices.

Storing the observation/system matrix on the device

The entire (full or sparse) observation/system matrix H can be transferred to the chosen device to speed up the computations. However, the matrix H can be so large that it cannot fit into the device memory completely. In such a case, toggling `options.storeData` to false will cause only the current time step to be stored in the device at any given time. This will slow down the computations, but possibly significantly decrease the memory requirements. The observation/system matrix still needs to completely fit into the system memory.

Outputting the a posteriori error covariance

It is possible to output the a posteriori error covariance P_k^+. There are two possible alternatives, first one stores the diagonals of P_k^+ at each time step k and then outputs them as a matrix, or, alternatively, output the final P_k^+ matrix from the very last time step. This can be controlled with `options.storeCovariance`, where 0 means that no covariance is saved, 1 stores the diagonals and 2 the entire (final) matrix.

General settings

These settings are global and affect all the other selections. All these parameters are also compulsory.

Such parameters are the total number of time steps, i.e. the number of KF steps, controlled with `options.Nt`. The final estimate size is controlled with `options.Nx`, `options.Ny` and `options.Nz` with

the total size being the multiplication of all these values. For 1D data, you should only use `options.Nx`, while 2D data should use both `options.Nx` and `options.Ny`.

The total number of measurements in ONE time step is specified with `options.Nm`. The total number of input measurements should then be `options.Nm`×`options.Nt`. If complex data is used, real and imaginary measurements are not considered separate, i.e. do NOT sum the total number of real and imaginary elements. The measurement data is input into `options.m0`.

Due to different dynamic systems, the observation/system matrix can be constant in time, be unique in each time step or be periodic. With `options.matCycles`, the user can control after how many time steps the matrix is used until chronologically the first matrix is used again. For example, if there are a total of six time points and `options.matCycles = 3`, then the time step 1 uses H_1, time step 2 uses H_2 and 3 H_3, while time step 4 again uses H_1, time step 5 H_2 and lastly time step 6 uses H_3.

Inputting matrices

Input matrices/vectors include H, F, Q, R, G and u, as well as the initial values x_0^- and P_0^-. Aside from the initial values, all can be sparse and time-varying. This section specifies possible special requirements in inputting the matrices/vectors.

For observation/system matrix H, the matrix has to be either full or sparse. The matrix, whether full or sparse, can also be complex-valued. The matrix needs to be input into the variable `options.H`. When using full matrices, it is recommended to use single precision matrices, however, the software automatically converts the data to single precision if double precision data is input. For time-varying data, the matrix should be concatenated such that rows contain all required elements along time. For example, if `options.Nm = 4` and `options.Nt = 2`, then the four first rows contain the matrix elements for the first time step and rows 5-8 the elements for the second time step.

Custom state transition matrix F can be input into `options.F`. This can be either full or sparse, real-valued or complex-valued. For time-varying data, use cell matrix, e.g. `F{1}` would contain the matrix for time step 1, `F{2}` for time step 2, etc. Note that kinematic model or SPKF overwrites any input F.

The process noise covariance Q can be input as either a scalar, vector, matrix, or cell matrix into `options.Q`. Scalar should be used when a single value is desired for all estimates at all time steps, i.e. Q is a diagonal matrix where each element is the same. Vector should be used when Q is constant, but each estimate has its own noise variance, i.e. Q is a diagonal matrix. Matrix should be used if a diagonal Q is used that also varies with time, where each column represents its own time step (e.g. column 2 is time step 2, etc.). Cell matrix should be used when Q is either a constant matrix (in which case only `options.Q{1}` exists) or if a time-varying matrices are used. The matrices in the cell matrix can be sparse. All values can be either real-valued or complex-valued.

The observation noise covariance R functions exactly as the process noise covariance Q above. The observation noise covariance is input into `options.R`.

The state transition matrix G for vector u can only be used if u is input as well. G functions as F and is input with `options.G`. For u, use matrix for time-varying data (like with diagonal Q). u is input into `options.u`.

Initial value of the estimate, x_0^-, is input into `options.x0`. The initial value for the estimation error covariance P_0^- is input into `options.P0` and can be scalar, vector or matrix.

Choosing the backend and device

Three different backends are available and can be selected before running the code. The backend is selected with `options.backend`, where 0 uses OpenCL, 1 CUDA and 2 CPU. If you failed to build a certain backend, for example CUDA when using only AMD GPU, that backend cannot be used.

After choosing the backend, you also need to choose the device. It is possible to successfully install a backend, but not have any compatible devices. Available devices for each backend can be queried with the following functions: `ArrayFire_OpenCL_device_info()` for OpenCL, `ArrayFire_CUDA_device_info()` for CUDA and `ArrayFire_CPU_device_info()` for CPU. The device numbers will be shown in MATLAB/Octave command window (or included in `ans` if `;` was inserted). The device can be selected with `options.device`. The default selection of 0 should often work optimally. Multi-device/GPU computation is not supported.

Choosing the correct filter

Since no extensive testing of the various filters have been performed no specific recommendations can be given as to which filter to use. However, the regular KF should be optimal for all cases where the dimensionality is not too large, i.e. no out of memory errors are encountered. Of EnKF, the regular EnKF is not really recommended, however, but has been mainly added due to being the very first EnKF. ETKF and ESTKF are recommended ensemble filters.

Dimension reduction KF can be quite slow initially due to the computation of the SVD for the prior covariance. At the moment there is no way to precompute this process. Note that DRKF might become unstable if used with the kinematic model.

Conjugate gradient KF and CGVKF are very memory efficient, but can get slow when using > 50 iterations. Both, and especially CGKF, are also vulnerable to floating point rounding errors and as such it is recommended to force orthogonalization. CGVKF is more reliable numerically, but can be slower to compute than CGKF.

Spectral KF is still experimental feature and as such should be only used for random walk cases where the number of measurements is small compared to the number of states.

References

[1] V.-V. Wettenhovi, "Image reconstruction in functional MRI and PET," Ph.D. dissertation, University of Eastern Finland, Kuopio, Finland, 6 2021.

[2] D. Simon, *Optimal State Estimation: Kalman, H_∞ and Nonlinear Approaches*. Wiley, 2006, pp. 128–129.

[3] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic Eng.*, vol. 82, no. 1, pp. 35–45, Mar. 1960.

[4] D. Simon, *Optimal State Estimation: Kalman, H_∞ and Nonlinear Approaches*. Wiley, 2006, pp. 156–157.

[5] ——, *Optimal State Estimation: Kalman, H_∞ and Nonlinear Approaches*. Wiley, 2006, p. 131.

[6] G. Evensen, "The ensemble Kalman filter: theoretical formulation and practical implementation," *Ocean Dynamics*, vol. 53, pp. 343–367, 2012.

[7] H. C. Godinez and J. D. Moulton, "An efficient matrix-free algorithm for the ensemble Kalman filter," *Comput Geosci*, vol. 16, pp. 565–575, 2012.

[8] J. Mandel, "Efficient implementation of the ensemble Kalman filter," University of Colorado, Tech. Rep., May 2006.

[9] C. H. Bishop, B. J. Etherton, and S. J. Majumdar, "Adaptive sampling with the ensemble transform Kalman filter. Part I: Theoretical aspects," *Monthly Weather Review*, vol. 129, no. 3, pp. 420 – 436, 2001. [Online]. Available: https://journals.ametsoc.org/view/journals/mwre/129/3/1520-0493_2001_129_0420_aswtet_2.0.co_2.xml

[10] L. Nerger, T. Janjić, J. Schröter, and W. Hiller, "A unification of ensemble square root Kalman filters," *Monthly Weather Review*, vol. 140, no. 7, pp. 2335–2345, 2012.

[11] L. Nerger, S. Schulte, and A. Bunse-Gerstner, "On the influence of model nonlinearity and localization on ensemble kalman smoothing," *Quarterly Journal of the Royal Meteorological Society*, vol. 140, no. 684, pp. 2249–2259, 2014.

[12] J. Hakkarainen, Z. Purisha, A. Solonen, and S. Siltanen, "Undersampled dynamic X-ray tomography with dimension reduction Kalman filter," *IEEE Trans. Comput. Imag.*, vol. 5, no. 3, pp. 492–501, 2019.

[13] J. M. Bardsley, A. Parker, A. Solonen, and M. Howard, "Krylov space approximate Kalman filtering," *Numerical Linear Algebra with Applications*, vol. 20, no. 2, pp. 171–184, 2011. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.805

[14] H. Ghorbanidehno, A. Kokkinaki, J. Y. Li, E. Darve, and P. K. Kitanidis, "Real-time data assimilation for large-scale systems: The spectral Kalman filter," *Advances in Water Resources*, vol. 86, pp. 260–272, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0309170815001645

[15] J. Y. Li, S. Ambikasaran, E. F. Darve, and P. K. Kitanidis, "A kalman filter powered by -matrices for quasi-continuous data assimilation problems," *Water Resources Research*, vol. 50, no. 5, pp. 3734–3749, 2014. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2013WR014607

[16] V.-V. Wettenhovi, V. Kolehmainen, J. Huttunen, M. Kettunen, O. Gröhn, and M. Vauhkonen, "State estimation with structural priors in fMRI," *J. Math. Imaging Vis.*, vol. 60, no. 2, pp. 174–188, 2018.

[17] D. Sbarbaro, M. Vauhkonen, and T. A. Johansen, "State estimation and inverse problems in electrical impedance tomography: observability, convergence and regularization," *Inverse problems*, vol. 31, no. 4, pp. 1–27, Mar. 2015.

[18] V.-V. Wettenhovi, V. Kolehmainen, J. Huttunen, M. Kettunen, O. Gröhn, and M. Vauhkonen, "State estimation in dynamic MRI," in *2018 IEEE Nuclear Science Symposium and Medical Imaging Conference Proceedings (NSS/MIC)*, 2018, pp. 1–6.

[19] J. P. Kaipio and E. Somersalo, *Statistical and computational inverse problems.* Springer, 2005.

[20] C. Byrne, "Block-iterative interior point optimization methods for image reconstruction from limited data," *Inverse Probl.*, vol. 16, no. 5, pp. 1405–1419, Oct. 2000.

[21] M. J. Ehrhardt, P. Markiewicz, M. Liljeroth, A. Barnes, V. Kolehmainen, J. S. Duncan, L. Pizarro, D. Atkinson, B. F. Hutton, S. Ourselin, K. Thielemans, and S. R. Arridge, "PET reconstruction with an anatomical MRI prior using parallel level sets," *IEEE Trans Med Imaging*, vol. 35, no. 9, pp. 2189–2199, Sep. 2016.

[22] Y. Lin, C. R. Schmidtlein, Q. Li, S. Li, and Y. Xu, "A Krasnoselskii-Mann algorithm with an improved EM preconditioner for PET image reconstruction," *IEEE Trans. Med. Imaging*, vol. 38, pp. 2114–2126, 2019.

[23] V. Kolehmainen, M. J. Ehrhardt, and S. R. Arridge, "Incorporating structural prior information and sparsity into EIT using parallel level sets," *Inverse Probl. Imag.*, vol. 13, no. 1930-8337_2019_2_285, p. 285, 2019.

[24] K. Bredies, K. Kunisch, and T. Pock, "Total generalized variation," *SIAM J. Imaging Sci.*, vol. 3, no. 3, pp. 492–526, 2010.

[25] L. Condat, "A primal-dual splitting method for convex optimization involving Lipschitzian, proximable and linear composite terms," *Journal of Optimization Theory and Applications*, vol. 158, no. 2, pp. 460–479, 2013.

[26] J. Qranfal and G. Tanoh, "Regularized Kalman filtering for dynamic SPECT," in *J. Phys.: Conf. Ser.*, vol. 124, no. 012042, 2008.

[27] Y. Qranfal, "Optimal recursive estimation techniques for dynamic medical image reconstruction," Ph.D. dissertation, Simon Fraser University, Burnaby, Canada, 12 2008.

[28] P. Hiltunen, S. Särkkä, I. Nissilä, A. Lajunen, and J. Lampinen, "State space regularization in the nonstationary inverse problem for diffuse optical tomography," *Inverse Problems*, vol. 27, no. 2, p. 025009, jan 2011.

[29] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning.* Cambridge, MA, USA: MIT Press, 2005.

[30] D. Simon, *Optimal State Estimation: Kalman, H_∞ and Nonlinear Approaches.* Wiley, 2006, pp. 208–210.

[31] H. E. Rauch, C. T. Striebel, and F. Tung, "Maximum likelihood estimates of linear dynamic systems," *AIAA Journal*, vol. 3, no. 8, pp. 1445–1450, 1965.

[32] D. Simon, *Optimal State Estimation: Kalman, H_∞ and Nonlinear Approaches*. Wiley, 2006, pp. 293–294.

[33] Y. Bar-Shalom, T. Kirubarajan, and X.-R. Li, *Estimation with Applications to Tracking and Navigation*. New York, NY, USA: John Wiley & Sons, Inc., 2002.

[34] R. Pichè, "Online tests of Kalman filter consistency," *International Journal of Adaptive Control and Signal Processing*, vol. 30, no. 1, pp. 115–124, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/acs.2571

[35] J. L. Crassidis and J. L. Junkins, *Optimal Estimation of Dynamic Systems*, 2nd ed. Chapman & Hall/CRC, 2011.

[36] O.-P. Tossavainen, "Shape estimation in electrical impedance tomography," Ph.D. dissertation, University of Kuopio, Kuopio, Finland, 5 2007.

[37] O.-P. Tossavainen, M. Vauhkonen, V. Kolehmainen, and K. Y. Kim, "Tracking of moving interfaces in sedimentation processes using electrical impedance tomography," *Chem. Eng. Sci.*, vol. 61, pp. 7717–7729, 2006.