Improving precision by using square roots

- The modifications to the basic Kalman filter that you have learned so far are able to ensure symmetric, positive-definite (or at least positive-semidefinite) covariance matrices.
- The filter is still sensitive to finite word length: no longer in the sense of causing divergence, but in the sense of not converging to as good a solution as possible.
- Consider the set of numbers: {1,000,000; 100; 1}.
  - There are six orders of magnitude in the spread between the largest and smallest.
- Now consider a second set of numbers: {1,000; 10; 1}.
  - There are only three orders of magnitude in spread.
- But, the second set is the square root of the first set: We can reduce dynamic range (number of bits needed to implement a desired precision) by using square roots.
- Implication: we can get away with single-precision instead of double-precision math.

Deriving the SR-KF, step 1a

- The place this is most manifest is in the eigenvalue spread of covariance matrices. If we could use square-root matrices instead, that would be better.
- Consider the Cholesky factorization from before.
  - Define $\Sigma^+_{\tilde{x},k} = \mathcal{S}^+_{\tilde{x},k}\left(\mathcal{S}^+_{\tilde{x},k}\right)^T$ and $\Sigma^-_{\tilde{x},k} = \mathcal{S}^-_{\tilde{x},k}\left(\mathcal{S}^-_{\tilde{x},k}\right)^T$.
- We would like to be able to compute the prediction-error covariance time updates and estimation-error measurement updates in terms of $\mathcal{S}^\pm_{\tilde{x},k}$ instead of $\Sigma^\pm_{\tilde{x},k}$.
- Let's take the steps in order ("SR-KF" = "Square-root Kalman filter").

SR-KF step 1a: State prediction time update.

- There is no change in this step from the standard KF (since it does not involve covariances). We compute:

$$\hat{x}^-_k = A_{k-1}\hat{x}^+_{k-1} + B_{k-1}u_{k-1}.$$

Deriving the SR-KF, step 1b

SR-KF step 1b: Prediction-error covariance time update.

- We start with the standard step:

$$\Sigma^-_{\tilde{x},k} = A_{k-1}\Sigma^+_{\tilde{x},k-1}A^T_{k-1} + \Sigma_{\widetilde{w}}.$$

- We would like to write this in terms of Cholesky factors:

$$\mathcal{S}^-_{\tilde{x},k}\left(\mathcal{S}^-_{\tilde{x},k}\right)^T = A_{k-1}\mathcal{S}^+_{\tilde{x},k-1}\left(\mathcal{S}^+_{\tilde{x},k-1}\right)^T A^T_{k-1} + \mathcal{S}_{\widetilde{w}}\mathcal{S}^T_{\widetilde{w}}.$$

- One option is to compute the right side, then take the Cholesky decomposition to compute the factors on the left side. This is computationally too intensive.
- Instead, start by noticing that we can write the equation as:

$$\mathcal{S}^-_{\tilde{x},k}\left(\mathcal{S}^-_{\tilde{x},k}\right)^T = \left[\begin{array}{cc} A_{k-1}\mathcal{S}^+_{\tilde{x},k-1}, & \mathcal{S}_{\widetilde{w}}\end{array}\right]\left[\begin{array}{cc} A_{k-1}\mathcal{S}^+_{\tilde{x},k-1}, & \mathcal{S}_{\widetilde{w}}\end{array}\right]^T = MM^T.$$

- This might at first appear to be exactly what we desire, but the problem is that $\mathcal{S}^-_{\tilde{x},k}$ is an $n \times n$ matrix, whereas M is an $n \times 2n$ matrix.

Deriving the SR-KF, step 1b (cont.)

- So, we have that $\mathcal{S}_{\tilde{x},k}^{-}\left(\mathcal{S}_{\tilde{x},k}^{-}\right)^{T} = MM^{T}$, but that $\mathcal{S}_{\tilde{x},k}^{-}$ and M have different dimensions so are not the same thing.
- This result is not the final answer but it is at least a step in the right direction. Enter the QR matrix decomposition.

QR decomposition: The QR decomposition factors $Z \in \mathbb{R}^{n\times m}$ as $Z = QR$, where $Q \in \mathbb{R}^{n\times n}$ is orthogonal, $R \in \mathbb{R}^{n\times m}$ is "upper-triangular," and $m \geq n$.

- Its importance to our problem is that R is related to the Cholesky factor we seek.
- Specifically, if $\widetilde{R} \in \mathbb{R}^{n\times n}$ is the upper-triangular portion of R, then $\widetilde{R}^{T}$ is the Cholesky factor of $\Sigma = M^{T}M$.
- That is, if $\widetilde{R} = \mathrm{qr}(M^{T})^{T}$, where $\mathrm{qr}(\cdot)$ performs the QR decomposition and returns the upper-triangular portion of R only, then $\widetilde{R}$ is the lower-triangular Cholesky factor of MM^{T}.

Deriving the SR-KF, steps 1b–1c

- Continuing with our derivation, notice that if we form M as above, then compute $\widetilde{R}$, we have our desired result.

$$\mathcal{S}_{\tilde{x},k}^{-} = \mathrm{qr}\left(\left[\begin{array}{cc} A_{k-1}\mathcal{S}_{\tilde{x},k-1}^{+}, & \mathcal{S}_{\widetilde{w}} \end{array}\right]^{T}\right)^{T}.$$

- In Octave code:

```
Sminus = qr([A*Splus,Sw]')';
Sminus = tril(Sminus(1:nx,1:nx));
```

SR-KF step 1c: Predict system output.

- There is no change in this step from the standard KF (since it does not involve covariances). We predict the system output as:

$$\hat{z}_k = C_k\hat{x}_k^{-} + D_k u_k.$$

Deriving the SR-KF, step 2a

SR-KF step 2a: Estimator (Kalman) gain matrix.

- In this step, we must compute $L_k = \Sigma_{\tilde{x}\tilde{z},k}^{-}(\Sigma_{\tilde{z},k})^{-1}$.
- Recall that $\Sigma_{\tilde{x}\tilde{z},k}^{-} = \Sigma_{\tilde{x},k}^{-}C_k^{T}$ and $\Sigma_{\tilde{z},k} = C_k\Sigma_{\tilde{x},k}^{-}C_k^{T} + \Sigma_{\tilde{v}}$.
- We find $\mathcal{S}_{\tilde{z},k}$ using the QR decomposition, as before. And, we already know $\mathcal{S}_{\tilde{x},k}^{-}$.
- So, we can now write $L_k(\mathcal{S}_{\tilde{z},k}\mathcal{S}_{\tilde{z},k}^{T}) = \Sigma_{\tilde{x}\tilde{z},k}^{-}$.
- If z_k is not a scalar, this equation may often be computed most efficiently via back-substitution in two steps.
  - First, $(M)\mathcal{S}_{\tilde{z},k}^{T} = \Sigma_{\tilde{x}\tilde{z},k}^{-}$ is solved for M, and...
  - Then $L_k\mathcal{S}_{\tilde{z},k} = M$ is solved for L_k.
  - Since $\mathcal{S}_{\tilde{z},k}$ is already triangular, no matrix inversion need be done.

Deriving the SR-KF, step 2a–2b

- Note that multiplying out $\Sigma^-_{\tilde{x},k} = \mathcal{S}^-_{\tilde{x},k}\left(\mathcal{S}^-_{\tilde{x},k}\right)^T$ in the computation of $\Sigma^-_{\tilde{x}\tilde{z},k}$ may drop some precision in L_k.
- However, this is not the critical issue.
- The critical issue is keeping $\mathcal{S}^\pm_{\tilde{x},k}$ accurate for whatever L_k is used, which is something that we do manage to accomplish.
- In Octave:

```
Sz = qr([C*Sminus,Sv]')';
Sz = tril(Sz(1:nz,1:nz));
L = (Sminus*Sminus')*C'/Sz'/Sz;
```

SR-KF step 2b: State estimate measurement update.

- There is no change in this step from the standard KF (since it does not involve covariances). We compute the state estimate as:

$$\hat{x}^+_k = \hat{x}^-_k + L_k(z_k - \hat{z}_k).$$

Deriving the SR-KF, step 2c

SR-KF step 2c: Estimation-error covariance meas. update.

- We write step 2c in terms of square-root factors as:

$$\Sigma^+_{\tilde{x},k} = \Sigma^-_{\tilde{x},k} - L_k \Sigma_{\tilde{z},k} L_k^T$$

$$\mathcal{S}^+_{\tilde{x},k}\left(\mathcal{S}^+_{\tilde{x},k}\right)^T = \mathcal{S}^-_{\tilde{x},k}\left(\mathcal{S}^-_{\tilde{x},k}\right)^T - L_k \mathcal{S}_{\tilde{z}} \mathcal{S}_{\tilde{z}}^T L_k^T.$$

- Note that the subtraction prohibits us using the QR decomposition to solve this problem as we did before; instead, we rely on the "Cholesky downdating" procedure.
- In Octave, we downdate for every column of the matrix $L_k \mathcal{S}_{\tilde{z}}$:

```
Sx_ = Sminus'; % Octave wants up-triang Cholesky factor but Sminus is low-tri
% Compute SigmaPlus = SigmaMinus - L*Sigmaz*L';
cov_update_vectors = L*Sz;
for j = 1:length(zhat) % Process each column of L*Sz one at a time
  Sx_ = cholupdate(Sx_,cov_update_vectors(:,j),'-');
end
Splus = Sx_';  % Re-transpose to force Splus to be lower-triangular
```


Octave code for SR-KF loop: Initialization

- We put the steps together to write a SR-KF. We initialize the filter similarly to standard KF, but now also need to compute square-root factors of the noise and initial-state-uncertainty covariance matrices.

```
% Load data from simulation of system dynamic model
load simOut.mat Ad Bd Cd Dd SigmaV SigmaW dT t u x z

% Determine number of states and timesteps
[nx,nt] = size(x);
% Initialize state estimate and covariance
xhat = zeros(nx,1); SigmaX = 1e-10*eye(nx); % SigmaX must be positive definite
% Initialize storage for state/bounds for plotting purposes
xhatstore = zeros(nx,nt); boundstore = zeros(nx,nt);

% Initialize simulation variables
SRSigmaW = chol(SigmaW,'lower'); % Square-root process noise covar
SRSigmaV = chol(SigmaV,'lower'); % Square-root sensor noise covar
SRSigmaX = chol(SigmaX,'lower'); % Square-root initial-state uncertainty
```

Octave code for SR-KF loop: Main loop

- We now enter the main SR-KF loop.

```
for k = 2:length(t)
  % SR-KF Step 1a: State prediction time update
  xhat = Ad*xhat + Bd*u(:,k-1); % use prior value of "u"

  % SR-KF Step 1b: Error covariance time update
  SRSigmaX = qr([Ad*SRSigmaX, SRSigmaW]')';
  SRSigmaX = tril(SRSigmaX(1:nx,1:nx));

  % SR-KF Step 1c: Predict system output
  zhat = Cd*xhat + Dd*u(:,k);

  % SR-KF Step 2a: Compute estimator (Kalman) gain matrix
  % Note: "help mrdivide" to see how "division" is implemented
  SRSigmaZ = qr([Cd*SRSigmaX,SRSigmaV]')';
  SRSigmaZ = tril(SRSigmaZ(1:length(zhat),1:length(zhat)));
  L = (SRSigmaX*SRSigmaX')*Cd'/SRSigmaZ'/SRSigmaZ;

  % SR-KF Step 2b: State estimate measurement update
  xhat = xhat + L*(z(:,k) - zhat);
```


Octave code for SR-KF loop: Main loop (cont.)

- The main program loop concludes, and we the store state-estimate and estimation-error covariance results.[1]

```
  % SR-KF Step 2c: Estimation-error covariance measurement update
  Sx_ = SRSigmaX';
  cov_update_vectors = L*SRSigmaZ;
  for j=1:length(zhat)
    Sx_ = cholupdate(Sx_,cov_update_vectors(:,j),'-');
  end
  SRSigmaX = Sx_';

  % [Store information for evaluation/plotting purposes]
  xhatstore(:,k) = xhat;
  boundstore(:,k) = 3*sqrt(diag(SRSigmaX*SRSigmaX'));
end
```

[1] Note: If you need to implement a SR-KF in a language other than Octave, an excellent discussion regarding finding Cholesky factors, QR factorizations, and Cholesky updating/downdating (with pseudo-code) may be found in: G.W. Stewart, Matrix Algorithms, Volume I: Basic Decompositions, Siam, 1998.

Summary

- You have now learned how to derive a square-root Kalman filter and implement it in Octave. Results are shown below for same data as Lesson 1.4.3. Initialization of $\Sigma^{+}_{\tilde{x},0}$ was slightly different but otherwise the SR-KF outputs are indistinguishable from the KF outputs (as expected!).