KALMAN FILTER GENERALIZATIONS

5.1: Maintaining symmetry of covariance matrices

- The Kalman filter as described so far is theoretically correct, but has known vulnerabilities and limitations in practical implementations.
- In this unit of notes, we consider the following issues:

1. Improving numeric robustness;
2. Sequential measurement processing and square-root filtering;
3. Dealing with auto- and cross-correlated sensor or process noise;
4. Extending the filter to prediction and smoothing;
5. Reduced-order filtering;
6. Using residue analysis to detect sensor faults.

Improving numeric robustness

- Within the filter, the covariance matrices $\Sigma_{\tilde{x},k}^{-}$ and $\Sigma_{\tilde{x},k}^{+}$ must remain

1. Symmetric, and
2. Positive definite (all eigenvalues strictly positive).

- It is possible for both conditions to be violated due to round-off errors in a computer implementation.
- We wish to find ways to limit or eliminate these problems.

Dealing with loss of symmetry

- The cause of covariance matrices becoming asymmetric or non-positive definite must be due to either the time-update or measurement-update equations of the filter.
- Consider first the time-update equation:

$$\Sigma_{\tilde{x},k}^{-} = A\Sigma_{\tilde{x},k-1}^{+}A^{T} + \Sigma_{\widetilde{w}}.$$

  - Because we are adding two positive-definite quantities together, the result must be positive definite.
  - A "suitable implementation" of the products of the matrices will avoid loss of symmetry in the final result.
- Consider next the measurement-update equation:

$$\Sigma_{\tilde{x},k}^{+} = \Sigma_{\tilde{x},k}^{-} - L_k C_k \Sigma_{\tilde{x},k}^{-}.$$

- *Theoretically*, the result is positive definite, but due to the subtraction operation it is possible for round-off errors in an implementation to result in a non-positive-definite solution.
- The problem may be mitigated in part by computing instead

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

  - This may be proven correct via

$$\begin{aligned}
\Sigma_{\tilde{x},k}^{+} &= \Sigma_{\tilde{x},k}^{-} - L_k \Sigma_{\tilde{z},k} L_k^{T} \\
&= \Sigma_{\tilde{x},k}^{-} - L_k \Sigma_{\tilde{z},k} \left(\Sigma_{\tilde{x},k}^{-} C_k^{T} \Sigma_{\tilde{z},k}^{-1}\right)^{T} \\
&= \Sigma_{\tilde{x},k}^{-} - L_k \Sigma_{\tilde{z},k} \Sigma_{\tilde{z},k}^{-1} C_k \Sigma_{\tilde{x},k}^{-} \\
&= \Sigma_{\tilde{x},k}^{-} - L_k C_k \Sigma_{\tilde{x},k}^{-}.
\end{aligned}$$

- With a "suitable implementation" of the products in the $L_k \Sigma_{\tilde{z},k} L_k^T$ term, symmetry can be guaranteed. However, the subtraction may still give a non-positive definite result if there is round-off error.
- A better solution is the Joseph form covariance update.

$$\Sigma_{\tilde{x},k}^{+} = [I - L_k C_k]\,\Sigma_{\tilde{x},k}^{-}\,[I - L_k C_k]^T + L_k \Sigma_{\tilde{v}} L_k^T.$$

  - This may be proven correct via

$$\begin{aligned}
\Sigma_{\tilde{x},k}^{+} &= [I - L_k C_k]\,\Sigma_{\tilde{x},k}^{-}\,[I - L_k C_k]^T + L_k \Sigma_{\tilde{v}} L_k^T \\
&= \Sigma_{\tilde{x},k}^{-} - L_k C_k \Sigma_{\tilde{x},k}^{-} - \Sigma_{\tilde{x},k}^{-} C_k^T L_k^T + L_k C_k \Sigma_{\tilde{x},k}^{-} C_k^T L_k^T + L_k \Sigma_{\tilde{v}} L_k^T \\
&= \Sigma_{\tilde{x},k}^{-} - L_k C_k \Sigma_{\tilde{x},k}^{-} - \Sigma_{\tilde{x},k}^{-} C_k^T L_k^T + L_k \left(C_k \Sigma_{\tilde{x},k}^{-} C_k^T + \Sigma_{\tilde{v}} \right) L_k^T \\
&= \Sigma_{\tilde{x},k}^{-} - L_k C_k \Sigma_{\tilde{x},k}^{-} - \Sigma_{\tilde{x},k}^{-} C_k^T L_k^T + L_k \Sigma_{\tilde{z},k} L^T \\
&= \Sigma_{\tilde{x},k}^{-} - L_k C_k \Sigma_{\tilde{x},k}^{-} - \Sigma_{\tilde{x},k}^{-} C_k^T L_k^T + \left(\Sigma_{\tilde{x},k}^{-} C_k^T \Sigma_{\tilde{z},k}^{-1} \right) \Sigma_{\tilde{z},k} L^T \\
&= \Sigma_{\tilde{x},k}^{-} - L_k C_k \Sigma_{\tilde{x},k}^{-}.
\end{aligned}$$

- Because the subtraction occurs in the "squared" term, this form "guarantees" a positive definite result.
- If we end up with a negative definite matrix (numerics), we can replace it by the nearest symmetric positive semidefinite matrix.[1]
- Omitting the details, the procedure is:
  - Calculate singular-value decomposition: $\Sigma = USV^T$.
  - Compute $H = VSV^T$.
  - Replace Σ with $(\Sigma + \Sigma^T + H + H^T)/4$.

[1] Nicholas J. Higham, "Computing a Nearest Symmetric Positive Semidefinite Matrix," *Linear Algebra and its Applications*, vol. 103, pp. 103–118, 1988.

5.2: Sequential processing of measurements

- There are still improvements that may be made. We can:
  - Reduce the computational requirements of the Joseph form,
  - Increase the precision of the numeric accuracy.
- One of the computationally intensive operations in the Kalman filter is the matrix inverse operation in $L_k = \Sigma_{\tilde{x},k}^{-} C_k^T \Sigma_{\tilde{z},k}^{-1}$.
- Using matrix inversion via Gaussian elimination (the most straightforward approach), is an $\mathcal{O}(m^3)$ operation, where m is the dimension of the measurement vector.
- If there is a single sensor, this matrix inverse becomes a scalar division, which is an $\mathcal{O}(1)$ operation.
- Therefore, if we can break the m measurements into m single-sensor measurements and update the Kalman filter that way, there is opportunity for significant computational savings.

Sequentially processing independent measurements

- We start by assuming that the sensor measurements are independent. That is, that

$$\Sigma_{\tilde{v}} = \text{diag}\left[\begin{array}{ccc} \sigma_{\tilde{v}_1}^2, & \cdots & \sigma_{\tilde{v}_m}^2 \end{array}\right].$$

- We will use colon ":" notation to refer to the measurement number. For example, $z_{k:1}$ is the measurement from sensor 1 at time k.
- Then, the measurement is

$$z_k = \begin{bmatrix} z_{k:1} \\ \vdots \\ z_{k:m} \end{bmatrix} = C_k x_k + v_k = \begin{bmatrix} C_{k:1}^T x_k + v_{k:1} \\ \vdots \\ C_{k:m}^T x_k + v_{k:m} \end{bmatrix},$$

where $C_{k:1}^T$ is the first row of C_k (for example), and $v_{k:1}$ is the sensor noise of the first sensor at time k, for example.

- We will consider this a sequence of scalar measurements $z_{k:1} \ldots z_{k:m}$, and update the state estimate and covariance estimates in m steps.
- We initialize the measurement update process with $\hat{x}_{k:0}^+ = \hat{x}_k^-$ and $\Sigma_{\tilde{x},k:0}^+ = \Sigma_{\tilde{x},k}^-$.
- Consider the measurement update for the ith measurement, $z_{k:i}$

$$\begin{aligned}\hat{x}_{k:i}^+ &= \mathbb{E}[x_k \mid \mathbb{Z}_{k-1}, z_{k:1} \ldots z_{k:i}] \\ &= \mathbb{E}[x_k \mid \mathbb{Z}_{k-1}, z_{k:1} \ldots z_{k:i-1}] + L_{k:i}(z_{k:i} - \mathbb{E}[z_k \mid \mathbb{Z}_{k-1}, z_{k:1} \ldots z_{k:i-1}]) \\ &= \hat{x}_{k:i-1}^+ + L_{k:i}(z_{k:i} - C_{k:i}^T \hat{x}_{k:i-1}^+).\end{aligned}$$

- Generalizing from before

$$L_{k:i} = \mathbb{E}[\tilde{x}_{k:i-1}^+ \tilde{z}_{k:i}^T] \Sigma_{\tilde{z}_{k:i}}^{-1}.$$

- Next, we recognize that the variance of the innovation corresponding to measurement $z_{k:i}$ is

$$\Sigma_{\tilde{z}_{k:i}} = \sigma_{\tilde{z}_{k:i}}^2 = C_{k:i}^T \Sigma_{\tilde{x},k:i-1}^+ C_{k:i} + \sigma_{\tilde{v}_i}^2.$$

- The corresponding gain is $L_{k:i} = \dfrac{\Sigma_{\tilde{x},k:i-1}^+ C_{k:i}}{\sigma_{\tilde{z}_{k:i}}^2}$ and the updated state is

$$\hat{x}_{k:i}^+ = \hat{x}_{k:i-1}^+ + L_{k:i}\left[z_{k:i} - C_{k:i}^T \hat{x}_{k:i-1}^+\right]$$

with covariance

$$\Sigma_{\tilde{x},k:i}^+ = \Sigma_{\tilde{x},k:i-1}^+ - L_{k:i} C_{k:i}^T \Sigma_{\tilde{x},k:i-1}^+.$$

- The covariance update can be implemented as

$$\Sigma_{\tilde{x},k:i}^+ = \Sigma_{\tilde{x},k:i-1}^+ - \frac{\Sigma_{\tilde{x},k:i-1}^+ C_{k:i} C_{k:i}^T \Sigma_{\tilde{x},k:i-1}^+}{C_{k:i}^T \Sigma_{\tilde{x},k:i-1}^+ C_{k:i} + \sigma_{\tilde{v}_i}^2}.$$

- An alternative update is the Joseph form,

$$\Sigma^+_{\tilde{x},k:i} = \left[I - L_{k:i}C^T_{k:i}\right]\Sigma^+_{\tilde{x},k:i}\left[I - L_{k:i}C^T_{k:i}\right]^T + L_{k:i}\sigma^2_{\tilde{v}_i}L^T_{k:i}.$$

- The final measurement update gives $\hat{x}^+_k = \hat{x}^+_{k:m}$ and $\Sigma^+_{\tilde{x},k} = \Sigma^+_{\tilde{x},k:m}$.

Sequentially processing correlated measurements

- The above process must be modified to accommodate the situation where sensor noise is correlated among the measurements.
- Assume that we can factor the matrix $\Sigma_{\tilde{v}} = \mathcal{S}_v\mathcal{S}_v^T$, where $\mathcal{S}_v$ is a lower-triangular matrix (for symmetric positive-definite $\Sigma_{\tilde{v}}$, we can).
  - The factor $\mathcal{S}_v$ is a kind of a matrix square root, and will be important in a number of places in this course.
  - It is known as the "Cholesky" factor of the original matrix.
  - In MATLAB, `Sv = chol(SigmaV,'lower');`
  - Be careful: MATLAB's default answer (without specifying "`lower`") is an upper-triangular matrix, which is not what we're after.
- The Cholesky factor has strictly positive elements on its diagonal (positive eigenvalues), so is guaranteed to be invertible.
- Consider a modification to the output equation of a system having correlated measurements

$$z_k = Cx_k + v_k$$

$$\bar{z}_k = \mathcal{S}_v^{-1}z_k = \mathcal{S}_v^{-1}Cx_k + \mathcal{S}_v^{-1}v_k$$

$$\bar{z}_k = \bar{C}x_k + \bar{v}_k.$$

  - Note that we will use the "bar" decoration $(\bar{\cdot})$ frequently in this chapter of notes.

- It rarely (if ever) indicates the mean of that quantity.
- Rather, it refers to a definition having similar meaning to the original symbol.
- For example, $\bar{z}_k$ is a (computed) output value, similar in interpretation to the measured output value z_k.

■ Consider now the covariance of the modified noise input $\bar{v}_k = \mathcal{S}_v^{-1} v_k$

$$\begin{aligned}\Sigma_{\tilde{\bar{v}}_k} &= \mathbb{E}[\bar{v}_k \bar{v}_k^T] \\ &= \mathbb{E}[\mathcal{S}_v^{-1} v_k v_k^T \mathcal{S}_v^{-T}] \\ &= \mathcal{S}_v^{-1} \Sigma_{\tilde{v}} \mathcal{S}_v^{-T} = I.\end{aligned}$$

■ Therefore, we have identified a transformation that de-correlates (and normalizes) measurement noise.

■ Using this revised output equation, we use the prior method.

■ We start the measurement update process with $\hat{x}_{k:0}^+ = \hat{x}_k^-$ and $\Sigma_{\tilde{x},k:0}^+ = \Sigma_{\tilde{x},k}^-$.

■ The Kalman gain is $\bar{L}_{k:i} = \dfrac{\Sigma_{\tilde{x},k:i-1}^+ \bar{C}_{k:i}}{\bar{C}_{k:i}^T \Sigma_{\tilde{x},k:i-1}^+ \bar{C}_{k:i} + 1}$ and the updated state is

$$\begin{aligned}\hat{x}_{k:i}^+ &= \hat{x}_{k:i-1}^+ + \bar{L}_{k:i}\left[\bar{z}_{k:i} - \bar{C}_{k:i}^T \hat{x}_{k:i-1}^+\right] \\ &= \hat{x}_{k:i-1}^+ + \bar{L}_{k:i}\left[(\mathcal{S}_v^{-1} z_k)_i - \bar{C}_{k:i}^T \hat{x}_{k:i-1}^+\right].\end{aligned}$$

with covariance

$$\Sigma_{\tilde{x},k:i}^+ = \Sigma_{\tilde{x},k:i-1}^+ - \bar{L}_{k:i} \bar{C}_{k:i}^T \Sigma_{\tilde{x},k:i-1}^+$$

(which may also be computed with a Joseph form update, for example).

■ The final measurement update gives $\hat{x}_k^+ = \hat{x}_{k:m}^+$ and $\Sigma_{\tilde{x}:k}^+ = \Sigma_{\tilde{x},k:m}^+$.

LDL updates for correlated measurements

- An alternative to the Cholesky decomposition for factoring the covariance matrix is the LDL decomposition

$$\Sigma_{\tilde{v}} = \mathcal{L}_v \mathcal{D}_v \mathcal{L}_v^T,$$

where $\mathcal{L}_v$ is lower-triangular and $\mathcal{D}_v$ is diagonal (with positive entries).

- In MATLAB,

```
[L,D] = ldl(SigmaV);
```

- The Cholesky decomposition is related to the LDL decomposition via

$$\mathcal{S}_v = \mathcal{L}_v \mathcal{D}_v^{1/2}.$$

- Both texts show how to use the LDL decomposition to perform a sequential measurement update.
- A computational advantage of LDL over Cholesky is that no square-root operations need be taken. (We can avoid finding $\mathcal{D}_v^{1/2}$.)
- A pedagogical advantage of introducing the Colesky decomposition is that we use it later on.

5.3: Square-root filtering

- The modifications to the basic Kalman filter that we have described so far are able to
  - Ensure symmetric, positive-definite covariance matrices;
  - Speed up the operation of a multiple-measurement Kalman filter.
- 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 required to implement a given precision of solution) by using square roots of numbers.
- For example, we can get away with single-precision math instead of double-precision math.
- The place this really shows up is in the eigenvalue spread of covariance matrices. If we can 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 covariance time updates and 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 step 1a: State estimate time update.

- We compute

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

- No change in this step from standard KF.

SR-KF step 1b: Error covariance time update.

- We start with 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

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

- This might at first appear to be exactly what we desire, but the problem is that $\mathcal{S}^-_{\tilde{x},k}$ is and $n \times n$ matrix, whereas M is an $n \times 2n$ matrix.
- But, it is at least a step in the right direction. Enter the QR decomposition.

***QR decomposition*:** The QR decomposition algorithm computes two factors $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{n \times m}$ for a matrix $Z \in \mathbb{R}^{n \times m}$ such that $Z = QR$, Q is orthogonal, R is upper-triangular, and $m \geq n$.

- The property of the QR factorization that is important here is that R is related to the Cholesky factor we wish to find.
- Specifically, if $\tilde{R} \in \mathbb{R}^{n \times n}$ is the upper-triangular portion of R, then $\tilde{R}^T$ is the Cholesky factor of $\Sigma = M^T M$.
- That is, if $\tilde{R} = \mathrm{qr}(M^T)^T$, where $\mathrm{qr}(\cdot)$ performs the QR decomposition and returns the upper-triangular portion of R only, then $\tilde{R}$ is the lower-triangular Cholesky factor of MM^T.
- Continuing with our derivation, notice that if we form M as above, then compute $\tilde{R}$, we have our desired result.

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

- The computational complexity of the QR decomposition is $\mathcal{O}(mn^2)$, whereas the complexity of the Cholesky factor is $\mathcal{O}(n^3/6)$ plus $\mathcal{O}(mn^2)$ to first compute MM^T.
- In MATLAB:

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

SR-KF step 1c: Estimate system output.

- As before, we estimate the system output as

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

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 may 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 $L_k(\mathcal{S}_{\tilde{z},k}\mathcal{S}^T_{\tilde{z},k}) = \Sigma^-_{\tilde{x}\tilde{z},k}$.
- If z_k is not a scalar, this equation may often be computed most efficiently via back-substitution in two steps.
  - First, $(M)\mathcal{S}^T_{\tilde{z},k} = \Sigma^-_{\tilde{x}\tilde{z},k}$ is found, and
  - Then $L_k\mathcal{S}_{\tilde{z},k} = M$ is solved.
  - Back-substitution has complexity $\mathcal{O}(n^2/2)$.
  - Since $\mathcal{S}_{\tilde{z},k}$ is already triangular, no matrix inversion need be done.
- 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 MATLAB:

```
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.

- This is done just as in the standard Kalman filter,

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

SR-KF step 2c: Error covariance measurement update.

- Finally, we update the error covariance matrix.

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

  which can be written as,

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

- Note that the "−" sign prohibits us using the QR decomposition to solve this problem as we did before.
- Instead, we rely on the "Cholesky downdating" procedure.
- In MATLAB,

```
% deal with MATLAB wanting upper-triangular Cholesky factor
Sx_ = Sminus';
% Want SigmaPlus = SigmaMinus - L*Sigmaz*L';
cov_update_vectors = L*Sz;
for j=1:length(zhat),
  Sx_ = cholupdate(Sx_,cov_update_vectors(:,j),'-');
end
% Re-transpose to undo MATLAB's strange Cholesky factor
Splus = Sx_';
```

- If you need to implement this kind of filter in a language other than MATLAB, a really excellent discussion of finding Cholesky factors, QR factorizations, and both Cholesky updating and downdating may be found in: G.W. Stewart, Matrix Algorithms, Volume I: Basic Decompositions, Siam, 1998. Pseudo-code is included.

Summary of the square-root linear Kalman filter

Linear state-space model:
$$x_k = A_{k-1}x_{k-1} + B_{k-1}u_{k-1} + w_{k-1}$$
$$z_k = C_k x_k + D_k u_k + v_k,$$

where w_k and v_k are independent, zero-mean, Gaussian noise processes of covariance matrices $\Sigma_{\widetilde{w}}$ and $\Sigma_{\tilde{v}}$, respectively.

Initialization: For $k = 0$, set

$$\hat{x}_0^+ = \mathbb{E}[x_0] \qquad \mathcal{S}_{\widetilde{w}} = \texttt{chol}(\Sigma_{\widetilde{w}}, \texttt{'lower'}).$$
$$\Sigma_{\tilde{x},0}^+ = \mathbb{E}[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T]. \qquad \mathcal{S}_{\tilde{v}} = \texttt{chol}(\Sigma_{\tilde{v}}, \texttt{'lower'}).$$
$$\mathcal{S}_{\tilde{x},0}^+ = \texttt{chol}(\Sigma_{\tilde{x},0}^+, \texttt{'lower'}).$$

Computation: For $k = 1, 2, \ldots$ compute:

State estimate time update: $\hat{x}_k^- = A_{k-1}\hat{x}_{k-1}^+ + B_{k-1}u_{k-1}.$

Error covariance time update: $\mathcal{S}_{\tilde{x},k}^- = \texttt{cholupdate}\left(\left(A_{k-1}\mathcal{S}_{\tilde{x},k-1}^+\right)^T, \mathcal{S}_{\widetilde{w}}^T\right)^T.$

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

*Estimator gain matrix:** $\mathcal{S}_{\tilde{z},k} = \texttt{cholupdate}\left(\left(C_k\mathcal{S}_{\tilde{x},k}^-\right)^T, \mathcal{S}_{\tilde{v}}^T\right)^T.$

$M\mathcal{S}_{\tilde{z},k}^T = \mathcal{S}_{\tilde{x},k}^-\left(\mathcal{S}_{\tilde{x},k}^-\right)^T C_k^T$, (solved by backsubstitution)

$L_k\mathcal{S}_{\tilde{z},k} = M$, (solved by backsubstitution)

State estimate meas. update: $\hat{x}_k^+ = \hat{x}_k^- + L_k(z_k - \hat{z}_k).$

Error covariance meas. update: $\mathcal{S}_{\tilde{x},k}^+ = \texttt{cholupdate}\left(\left(\mathcal{S}_{\tilde{x},k}^-\right)^T, \left(L_k\mathcal{S}_{\tilde{z},k}\right)^T, \texttt{'-'}\right)^T.$

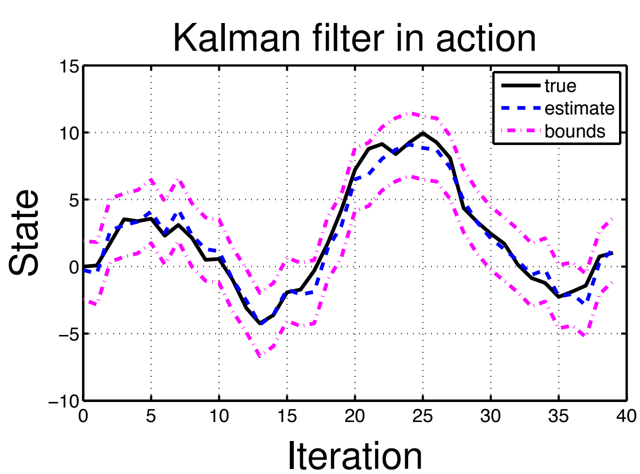
*If a measurement is missed for some reason, then simply skip the measurement update for that iteration. That is, $L_k = 0$ and $\hat{x}_k^+ = \hat{x}_k^-$ and $\mathcal{S}_{\tilde{x},k}^+ = \mathcal{S}_{\tilde{x},k}^-$.

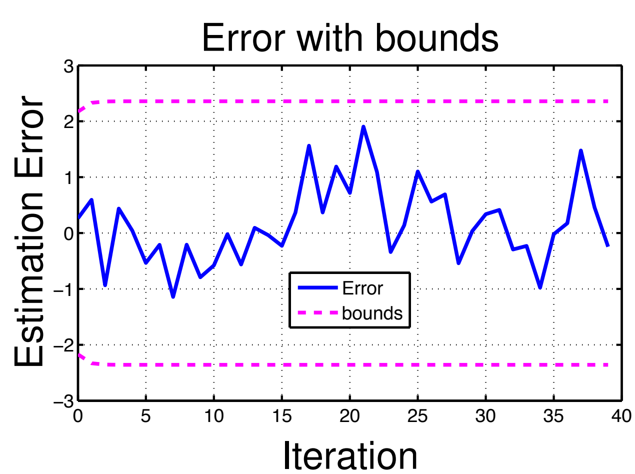
5.4: MATLAB code for the square-root Kalman filter steps

- Coding a square-root Kalman filter in MATLAB is straightforward.

```
% Initialize simulation variables
SRSigmaW = chol(1,'lower'); % Square-root process noise covar
SRSigmaV = chol(1,'lower'); % Square-root sensor noise covar
A = 1; B = 1; C = 1; D = 0; % Plant definition matrices
maxIter = 40;

xtrue = 0; xhat = 0 % Initialize true and estimated system initial state
SigmaX = 0.1;    % Initialize Kalman filter covariance
SRSigmaX = chol(SigmaX,'lower');
u = 0;       % Unknown initial driving input: assume zero

% Reserve storage for variables we might want to plot/evaluate
xstore = zeros(maxIter+1,length(xtrue)); xstore(1,:) = xtrue;
xhatstore = zeros(maxIter,length(xhat));
SigmaXstore = zeros(maxIter,length(xhat)); % store diagonal only

for k = 1:maxIter,
  % SR-KF Step 1a: State estimate time update
  xhat = A*xhat + B*u; % use prior value of "u"

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

  % [Implied operation of system in background, with
  % input signal u, and output signal z]
  u = 0.5*randn(1) + cos(k/pi); % for example... (measured)
  w = SRSigmaW*randn(length(xtrue));
  v = SRSigmaV*randn(length(C*xtrue));
  ztrue = C*xtrue + D*u + v;  % y is based on present x and u
  xtrue = A*xtrue + B*u + w;  % future x is based on present u

  % SR-KF Step 1c: Estimate system output
  zhat = C*xhat + D*u;

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

```
  L = (SRSigmaX*SRSigmaX')*C'/SRSigmaZ'/SRSigmaZ;

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

  % SR-KF Step 2c: 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]
  xstore(k+1,:) = xtrue;   xhatstore(k,:) = xhat;
  SigmaXstore(k,:) = diag(SRSigmaX*SRSigmaX');
end;

figure(1); clf;
plot(0:maxIter-1,xstore(1:maxIter),'k-',0:maxIter-1,xhatstore,'b--', ...
  0:maxIter-1,xhatstore+3*sqrt(SigmaXstore),'m-.',...
  0:maxIter-1,xhatstore-3*sqrt(SigmaXstore),'m-.'); grid;
title('Kalman filter in action'); xlabel('Iteration');
ylabel('State'); legend('true','estimate','bounds');

figure(2); clf;
plot(0:maxIter-1,xstore(1:maxIter)-xhatstore,'b-', ...
  0:maxIter-1,3*sqrt(SigmaXstore),'m--',...
  0:maxIter-1,-3*sqrt(SigmaXstore),'m--');
grid; legend('Error','bounds',0); title('Error with bounds');
xlabel('Iteration'); ylabel('Estimation Error');
```





5.5: Cross-correlated process and measurement noises: Coincident

- The standard KF assumes that $\mathbb{E}[w_k v_j^T] = 0$. But, sometimes we may encounter systems where this is not the case.
- This might happen if both the physical process and the measurement system are affected by the same source of disturbance. Examples are changes of temperature, or inductive electrical interference.
- In this section, we assume that $\mathbb{E}[w_k w_j^T] = \Sigma_{\widetilde{w}} \delta_{kj}$, $\mathbb{E}[v_k v_j^T] = \Sigma_{\tilde{v}} \delta_{kj}$, and $\mathbb{E}[w_k v_j^T] = \Sigma_{\widetilde{w}\tilde{v}} \delta_{kj}$.
- Note that the correlation between noises is memoryless: the only correlation is at identical time instants.
- We can handle this case if we re-write the plant equation so that it has a new process noise that is uncorrelated with the measurement noise.
- Using an arbitrary matrix T (to be determined), we can write

$$\begin{aligned} x_{k+1} &= A_k x_k + B_k u_k + w_k + T\left(z_k - C_k x_k - D_k u_k - v_k\right) \\ &= (A_k - TC_k)x_k + (B_k - TD_k)u_k + w_k - Tv_k + Tz_k. \end{aligned}$$

- Denote the new transition matrix $\bar{A}_k = A_k - TC_k$, new input matrix as $\bar{B}_k = B_k - TD_k$, and the new process noise as $\bar{w}_k = w_k - Tv_k$.
- Further, denote the known (measured/computed) sequence as a new input $\bar{u}_k = Tz_k$.
- Then, we can write a modified state space system

$$x_{k+1} = \bar{A}_k x_k + \bar{B}_k u_k + \bar{u}_k + \bar{w}_k.$$

- We can create a Kalman filter for this system, provided that the cross-correlation between the new process noise $\bar{w}_k$ and the sensor noise v_k is zero. We enforce this:

$$\mathbb{E}[\bar{w}_k v_k^T] = \mathbb{E}\left[[w_k - Tv_k]v_k^T\right] = \Sigma_{\widetilde{w}\tilde{v}} - T\Sigma_{\tilde{v}} = 0.$$

- This gives us that the previously unspecified matrix $T = \Sigma_{\widetilde{w}\tilde{v}}\Sigma_{\tilde{v}}^{-1}$.
- Using the above, the covariance of the new process noise may be found to be

$$\begin{aligned}
\Sigma_{\tilde{\tilde{w}}} &= \mathbb{E}[\bar{w}_k \bar{w}_k^T] \\
&= \mathbb{E}\left[\left[w_k - \Sigma_{\widetilde{w}\tilde{v}}\Sigma_{\tilde{v}}^{-1}v_k\right]\left[w_k - \Sigma_{\widetilde{w}\tilde{v}}\Sigma_{\tilde{v}}^{-1}v_k\right]^T\right] \\
&= \Sigma_{\widetilde{w}} - \Sigma_{\widetilde{w}\tilde{v}}\Sigma_{\tilde{v}}^{-1}\Sigma_{\widetilde{w}\tilde{v}}^T.
\end{aligned}$$

- A new Kalman filter may be generated using these definitions:

$$\begin{aligned}
\bar{A}_k &= A_k - \Sigma_{\widetilde{w}\tilde{v}}\Sigma_{\tilde{v}}^{-1}C_k \\
\Sigma_{\tilde{\tilde{w}}} &= \Sigma_{\widetilde{w}} - \Sigma_{\widetilde{w}\tilde{v}}\Sigma_{\tilde{v}}^{-1}\Sigma_{\widetilde{w}\tilde{v}}^T,
\end{aligned}$$

and

$$\begin{aligned}
x_{k+1} &= (A_k - \Sigma_{\widetilde{w}\tilde{v}}\Sigma_{\tilde{v}}^{-1}C_k)x_k + (B_k - \Sigma_{\widetilde{w}\tilde{v}}\Sigma_{\tilde{v}}^{-1}D_k)u_k + \Sigma_{\widetilde{w}\tilde{v}}\Sigma_{\tilde{v}}^{-1}z_k + \bar{w}_k \\
z_k &= C_k x_k + D_k u_k + v_k.
\end{aligned}$$

Cross-correlated process and measurement noises: Shifted

- A close relation to the above is when the process noise and sensor noise have correlation one timestep apart.
- That is, we assume that $\mathbb{E}[w_k w_j^T] = \Sigma_{\widetilde{w}}\delta_{kj}$, $\mathbb{E}[v_k v_j^T] = \Sigma_{\tilde{v}}\delta_{kj}$, and $\mathbb{E}[w_k v_j^T] = \Sigma_{\widetilde{w}\tilde{v}}\delta_{k,j-1}$. The cross-correlation is nonzero only between w_{k-1} and v_k.
- We can re-derive the KF equations using this assumption. We will find that the differences show up in the state-error covariance terms.

- The state prediction error is

$$\tilde{x}_k^- = x_k - \hat{x}_k^- = A_k \tilde{x}_{k-1}^+ + w_{k-1}.$$

- With the assumptions of this section, the covariance between the state prediction error and the measurement noise is

$$\mathbb{E}[\tilde{x}_k^- v_k^T] = \mathbb{E}\left[[A_k \tilde{x}_{k-1}^+ + w_{k-1}]v_k^T\right] = \Sigma_{\widetilde{w}\widetilde{v}}.$$

- The covariance between the state and the measurement becomes

$$\begin{aligned}\mathbb{E}\left[\tilde{x}_k^- \tilde{z}_k^T \mid \mathbb{Z}_{k-1}\right] &= \mathbb{E}\left[\tilde{x}_k^- \left(C_k \tilde{x}_k^- + v_k\right)^T \mid \mathbb{Z}_{k-1}\right] \\ &= \Sigma_{\tilde{x},k}^- C_k^T + \Sigma_{\widetilde{w}\widetilde{v}}.\end{aligned}$$

- The measurement prediction covariance becomes

$$\begin{aligned}\Sigma_{\tilde{z},k} &= \mathbb{E}\left[\tilde{z}_k \tilde{z}_k^T\right] \\ &= \mathbb{E}\left[[C_k \tilde{x}_k^- + v_k][C_k \tilde{x}_k^- + v_k]^T\right] \\ &= C_k \Sigma_{\tilde{x},k}^- C_k^T + \Sigma_{\tilde{v}} + C_k \Sigma_{\widetilde{w}\widetilde{v}} + \Sigma_{\widetilde{w}\widetilde{v}}^T C_k^T.\end{aligned}$$

- The modified KF gain then becomes,

$$L_k = \left[\Sigma_{\tilde{x},k}^- C_k^T + \Sigma_{\widetilde{w}\widetilde{v}}\right]\left(C_k \Sigma_{\tilde{x},k}^- C_k^T + \Sigma_{\tilde{v}} + C_k \Sigma_{\widetilde{w}\widetilde{v}} + \Sigma_{\widetilde{w}\widetilde{v}}^T C_k^T\right)^{-1}.$$

- Except for the modified filter gain, all of the KF equations are the same as in the standard case.
- Note that since w_{k-1} is the process noise corresponding to the interval $[t_{k-1}, t_k]$ and v_j is the measurement noise at t_j, it can be seen that the first case considered process noise correlated with measurement noise at the beginning of the above interval, and the second case considered process noise correlated with the end of the interval.

Auto-correlated process noise

- Another common situation that contradicts the KF assumptions is that the process noise is correlated in time.
- That is, with the standard assumption that $x_{k+1} = A_k x_k + B_k u_k + w_k$, we do not have zero-mean white-noise w_k.
- Instead, we have $w_k = A_w w_{k-1} + \bar{w}_{k-1}$, where $\bar{w}_{k-1}$ is a zero-mean white-noise process.
- We handle this situation by estimating both the true system state x_k and also the noise state w_k. We have

$$\begin{bmatrix} x_k \\ w_k \end{bmatrix} = \begin{bmatrix} A_{k-1} & I \\ 0 & A_w \end{bmatrix} \begin{bmatrix} x_{k-1} \\ w_{k-1} \end{bmatrix} + \begin{bmatrix} B_{k-1} \\ 0 \end{bmatrix} u_{k-1} + \begin{bmatrix} 0 \\ \bar{w}_{k-1} \end{bmatrix}$$

$$x_k^* = A_{k-1}^* x_{k-1}^* + B_{k-1}^* u_{k-1} + w_{k-1}^*,$$

where the overall process noise covariance is

$$\Sigma_{\widetilde{w}^*} = \mathbb{E}[w_{k-1}^* (w_{k-1}^*)^T] = \begin{bmatrix} 0 & 0 \\ 0 & \mathbb{E}[\bar{w}_{k-1} \, (\bar{w}_{k-1})^T] \end{bmatrix}$$

and the output equation is

$$\begin{aligned} z_k &= \begin{bmatrix} C_k & 0 \end{bmatrix} \begin{bmatrix} x_k \\ w_k \end{bmatrix} + D_k u_k + v_k \\ &= C_k^* x_k^* + D_k u_k + v_k. \end{aligned}$$

- A standard Kalman filter may now be designed using the definitions of x_k^*, A_k^*, B_k^*, C_k^*, D_k, $\Sigma_{\widetilde{w}^*}$, and $\Sigma_{\tilde{v}}$.

Auto-correlated sensor noise

- Similarly, we might encounter situations with auto-correlated sensor noise: $v_k = A_v v_{k-1} + \bar{v}_{k-1}$, where $\bar{v}_k$ is white.

- We take a similar approach. The augmented system is

$$\begin{bmatrix} x_k \\ v_k \end{bmatrix} = \begin{bmatrix} A_{k-1} & 0 \\ 0 & A_v \end{bmatrix} \begin{bmatrix} x_{k-1} \\ v_{k-1} \end{bmatrix} + \begin{bmatrix} B_{k-1} \\ 0 \end{bmatrix} u_{k-1} + \begin{bmatrix} w_{k-1} \\ \bar{v}_{k-1} \end{bmatrix}$$

$$x_k^* = A_{k-1}^* x_{k-1}^* + B_{k-1}^* u_{k-1} + w_{k-1}^*$$

with output equation

$$\begin{aligned} z_k &= \begin{bmatrix} C_k & I \end{bmatrix} \begin{bmatrix} x_k \\ v_k \end{bmatrix} + D_k u_k + 0 \\ &= C_k^* x_k^* + D_k u_k + 0 \end{aligned}$$

- The covariance of the combined process noise is

$$\Sigma_{\widetilde{w}^*} = \mathbb{E}\left[\begin{pmatrix} w_k \\ \bar{v}_k \end{pmatrix} \begin{pmatrix} w_k & \bar{v}_k \end{pmatrix} \right] = \begin{bmatrix} \Sigma_{\widetilde{w}} & 0 \\ 0 & \Sigma_{\tilde{\bar{v}}} \end{bmatrix}.$$

- A Kalman filter may be designed using these new definitions of of x_k^*, A_k^*, B_k^*, C_k^*, D_k, $\Sigma_{\widetilde{w}^*}$, with $\Sigma_{\tilde{v}} = 0$ (the placeholder for measurement noise is zero in the above formulations).

Measurement differencing

- Zero-covariance measurement noise can cause numerical issues.
- A sneaky way to fix this is to introduce an artificial measurement that is computed as a scaled difference between two actual measurements: $\bar{z}_k = z_{k+1} - A_v z_k$.
- KF equations can then be developed using this new "measurement."
- The details are really messy and not conducive to a lecture presentation. I refer you to Bar-Shalom!
- Care must be taken to deal with the "future" measurement z_{k+1} in the update equations, but it works out to a causal solution in the end.

5.6: Kalman-filter prediction and smoothing

- Prediction is the estimation of the system state at a time m beyond the data interval. That is,

$$\hat{x}_{m|k}^{-} = \mathbb{E}[x_m \mid \mathbb{Z}_k],$$

where $m > k$.

- There are three different prediction scenarios:
  - Fixed-point prediction: Find $\hat{x}_{m|k}^{-}$ where m is fixed, but k is changing as more data becomes available;
  - Fixed-lead prediction: Find $\hat{x}_{k+L|k}^{-}$ where L is a fixed lead time;
  - Fixed-interval prediction: Find $\hat{x}_{m|k}^{-}$ where k is fixed, but m can take on multiple future values.

- The desired predictions can be extrapolated from the standard Kalman filter state and estimates.

- The basic approach is to use the relationship (cf. Homework 1)

$$x_m = \left(\prod_{j=0}^{m-k-1} A_{m-1-j}\right) x_k + \sum_{i=k}^{m-1}\left(\prod_{j=0}^{m-i-2} A_{m-1-j}\right) B_i u_i + \sum_{i=k}^{m-1}\left(\prod_{j=0}^{m-i-2} A_{m-1-j}\right) w_i,$$

in the relationship

$$\hat{x}_{m|k}^{-} = \mathbb{E}[x_m \mid \mathbb{Z}_k],$$

with the additional knowledge that $\hat{x}_k^{+} = \mathbb{E}[x_k \mid \mathbb{Z}_k]$ from a standard Kalman filter.

- That is,

$$\begin{aligned}\hat{x}^-_{m|k} &= \mathbb{E}[x_m \mid \mathbb{Z}_k] \\ &= \mathbb{E}\left[\left(\prod_{j=0}^{m-k-1} A_{m-1-i}\right) x_k \mid \mathbb{Z}_k\right] + \mathbb{E}\left[\sum_{i=k}^{m-1}\left(\prod_{j=0}^{m-i-2} A_{m-1-i}\right) B_i u_i \mid \mathbb{Z}_k\right] \\ &\quad + \mathbb{E}\left[\sum_{i=k}^{m-1}\left(\prod_{j=0}^{m-i-2} A_{m-1-j}\right) w_i \mid \mathbb{Z}_k\right] \\ &= \left(\prod_{j=0}^{m-k-1} A_{m-1-i}\right)\hat{x}^+_k + \sum_{i=k}^{m-1}\left(\prod_{j=0}^{m-i-2} A_{m-1-i}\right) B_i \mathbb{E}[u_i \mid \mathbb{Z}_k].\end{aligned}$$

- Note that we often assume that $\mathbb{E}[u_k] = 0$.
- If, furthermore, the system is time invariant,

$$\hat{x}^-_{m|k} = A^{m-k}\hat{x}^+_k.$$

- The covariance of the prediction is

$$\begin{aligned}\Sigma^-_{\tilde{x},m|k} &= \mathbb{E}[(x_m - \hat{x}^-_{m|k})(x_m - \hat{x}^-_{m|k})^T \mid \mathbb{Z}_k] \\ &= \left(\prod_{j=0}^{m-k-1} A_{m-1-i}\right)\Sigma^+_{\tilde{x},k}\left(\prod_{j=0}^{m-k-1} A_{m-1-i}\right)^T \\ &\quad + \sum_{i=k}^{m-1}\left(\prod_{j=0}^{m-i-2} A_{m-1-j}\right)\Sigma_{\widetilde{w}}\left(\prod_{j=0}^{m-i-2} A^T_{m-1-j}\right).\end{aligned}$$

- If the system is time invariant, this reduces to

$$\Sigma^-_{\tilde{x},m|k} = A^{m-k}\Sigma^+_{\tilde{x},k}\left(A^{m-k}\right)^T + \sum_{j=1}^{m-k} A^j \Sigma_{\widetilde{w}}\left(A^j\right)^T.$$

Smoothing

- Smoothing is the estimation of the system state at a time m amid the data interval. That is,

$$\hat{x}^+_{m|N} = \mathbb{E}[x_m \mid \mathbb{Z}_N],$$

  where $m < N$.

- There are three different smoothing scenarios:
  - Fixed-point smoothing: Find $\hat{x}^+_{m|k}$ where m is fixed, but k is changing as more data becomes available;
  - Fixed-lag smoothing: Find $\hat{x}^-_{k-L|k}$ where L is a fixed lag time;
  - Fixed-interval smoothing: Find $\hat{x}^+_{m|N}$ where k is fixed, but m can take on multiple past values.
- Of these, fixed-interval smoothing is the most relevant, and both texts have detailed derivations.
- The others use a variation of this idea.

Fixed interval smoothing

- The algorithm consists of a forward recursive pass followed by a backward pass.
- The forward pass uses a Kalman filter, and saves the intermediate results $\hat{x}^-_k$, $\hat{x}^+_k$, $\Sigma^-_{\tilde{x},k}$, and $\Sigma^+_{\tilde{x},k}$.
- The backward pass starts at time N of the last measurement, and computes the smoothed state estimate using the results obtained from the forward pass.

- Recursive equations (backward sweep)

$$\hat{x}^+_{m|N} = \hat{x}^+_m + \lambda_m\left(\hat{x}^+_{m+1|N} - \hat{x}^-_{m+1}\right)$$

$$\lambda_m = \Sigma^+_{\tilde{x},m} A^T_m \left(\Sigma^-_{\tilde{x},m+1}\right)^{-1}$$

where $m = N-1, N-2, \ldots, 0$. Note, $\hat{x}^+_{N|N} = \hat{x}^+_N$ to start backward pass.

- The error covariance matrix for the smoothed estimate is

$$\Sigma^+_{\tilde{x},m|N} = \Sigma^+_{\tilde{x},m} + \lambda_m\left[\Sigma^+_{\tilde{x},m+1|N} - \Sigma^-_{\tilde{x},m+1}\right]\lambda^T_m,$$

but it is not needed to be able to perform the backward pass.

- Note, the term in the square brackets is negative semi-definite, so the covariance of the smoothed estimate is "smaller" than for the filtered estimate only.

Fixed point smoothing

- Here, m is fixed, and the final point k keeps increasing.

$$\hat{x}^+_{m|k} = \hat{x}^+_{m|k-1} + \mu_k\left(\hat{x}^+_k - \hat{x}^-_k\right)$$

$$\mu_k = \prod_{i=m}^{k-1}\lambda_i,$$

where the product multiplies on the left as i increases.

- For $k = m+1$,

$$\hat{x}^+_{m|m+1} = \hat{x}^+_m + \mu_{m+1}\left(\hat{x}^+_{m+1} - \hat{x}^-_{m+1}\right)$$

$$\mu_{m+1} = \lambda_m = \Sigma^+_{\tilde{x},m} A^T_m \left(\Sigma^-_{\tilde{x},m+1}\right)^{-1}.$$

- For $k = m + 2$,

$$\hat{x}^+_{m|m+2} = \hat{x}^+_{m|m+1} + \mu_{m+2}\left(\hat{x}^+_{m+2} - \hat{x}^-_{m+2}\right)$$
$$\mu_{m+2} = \Sigma^+_{\tilde{x},m+1} A^T_{m+1}\left(\Sigma^-_{\tilde{x},m+2}\right)^{-1}\mu_{k+1},$$

and so forth.

Fixed lag smoothing

- Here, we seek to estimate the state vector at a fixed time interval lagging the time of the current measurement.
- This type of smoothing trades off estimation latency for more accuracy.
- The fixed interval smoothing algorithm could be used to perform fixed-lag smoothing when the number of backward steps equals the time lag
- This is fine as long as the number of backward steps is small.
- Fixed-lag smoothing algorithm has a startup problem: Cannot run until enough data is available.

5.7: Reduced-order Kalman filter

- Why estimate the entire state vector when you are measuring one or more state elements directly?
- Consider partitioning system state into (may require transformation)

$$x_{k:a} : \text{measured state}$$

$$x_{k:b} : \text{to be estimated.}$$

- So,

$$z_k = Cx_k + Du_k + v_k = x_{k:a} + Du_k + v_k$$

$$\hat{x}_{k:a} = z_k - Du_k = x_{k:a} + v_k.$$

- We assume that the measurement is noise-free (otherwise, we would need to estimate $x_{k:a}$ also). So, $\hat{x}_{k:a} = z_k - Du_k = x_{k:a}$.
- In order to design an estimator for $x_{k:b}$, we need to create a suitable state-space model of the dynamics of $x_{k:b}$.
- We begin by writing the equations for the partitioned system

$$\begin{bmatrix} x_{k+1:a} \\ \hdashline x_{k+1:b} \end{bmatrix} = \left[\begin{array}{c:c} A_{aa} & A_{ab} \\ \hdashline A_{ba} & A_{bb} \end{array}\right] \begin{bmatrix} x_{k:a} \\ \hdashline x_{k:b} \end{bmatrix} + \begin{bmatrix} B_a \\ \hdashline B_b \end{bmatrix} u_k + \begin{bmatrix} w_{k:a} \\ \hdashline w_{k:b} \end{bmatrix}$$

$$z_k = \left[\begin{array}{c:c} I & 0 \end{array}\right] \begin{bmatrix} x_{k:a} \\ \hdashline x_{k:b} \end{bmatrix} + Du_k.$$

- We wish to write the $x_{k:b}$ dynamics in the form:

$$x_{k+1:b} = A_{xb}x_{k:b} + B_{xb}m_{k,1} + \bar{w}_k$$

$$m_{k,2} = C_{xb}x_{k:b} + D_{xb}m_{k,1} + \bar{v}_k,$$

where $m_{k,1}$ and $m_{k,2}$ are some measurable inputs, and will be combinations of z_k and u_k.

- Once we have this state-space form, we can create a Kalman-filter state estimator for $x_{k:b}$.

- We start the derivation by finding an output equation for $x_{k:b}$. Consider the dynamics of the measured state:

$$x_{k+1:a} = A_{aa}x_{k:a} + A_{ab}x_{k:b} + B_a u_k + w_{k:a}$$

$$z_{k+1} = A_{aa}z_k + A_{ab}x_{k:b} + B_a u_k + w_{k:a}.$$

- Let

$$m_{k,2} = z_{k+1} - A_{aa}z_k - B_a u_k.$$

- Then

$$m_{k,2} = A_{ab}x_{k:b} + w_{k:a},$$

where $m_{k,2}$ is known/measurable and thus "C_{xb}" is equal to A_{ab}.

- This is our reduced-order estimator output relation.

- We now look for a state equation for $x_{k:b}$. Consider the dynamics of the estimated state:

$$x_{k+1:b} = A_{ba}x_{k:a} + A_{bb}x_{k:b} + B_b u_k + w_{k:b}$$

$$A_{bb}x_{k:b} + A_{ba}z_k + B_b u_k + w_{k:b}.$$

- Let

$$B_{xb}m_{k,1} = A_{ba}z_k + B_b u_k$$

so that the reduced-order recurrence relation is

$$x_{k+1:b} = A_{bb}x_{k:b} + B_{xb}m_{k,1} + w_{k:b}.$$

- This might be accomplished via

$$B_{xb}m_{k,1} = \left[\begin{array}{c:c} A_{ba} & B_b \end{array}\right]\left[\begin{array}{c} z_k \\ \hdashline u_k \end{array}\right]$$

although the details of how this is done do not matter in the end.

- So, for the purpose of designing our estimator, the state-space equations are:

$$x_{k+1:b} = A_{bb}x_{k:b} + B_{xb}m_{k,1} + \bar{w}_k$$

$$m_{k,2} = A_{ab}x_{k:b} + \bar{v}_k,$$

where $\bar{w}_k = w_{k:b}$ and $\bar{v}_k = w_{k:a}$.

- Note that the measurement is non-causal, so the filter output will lag the true output by one sample.
- Another (causal) approach that does not assume noise-free measurements is presented in: D. Simon, "Reduced Order Kalman Filtering without Model Reduction," *Control and Intelligent Systems*, vol. 35, no. 2, pp. 169–174, April 2007.
- This algorithm can end up more complicated than full Kalman filter unless many states are being removed from estimation requirements.

5.8: Measurement validation gating

- Sometimes the systems for which we would like a state estimate have sensors with intermittent faults.
- We would like to detect faulty measurements and discard them (the time update steps of the KF are still implemented, but the measurement update steps are skipped).
- The Kalman filter provides an elegant theoretical means to accomplish this goal. Note:
  - The measurement covariance matrix is $\Sigma_{\tilde{z},k} = C_k \Sigma_{\tilde{x},k}^{-} C_k^T + \Sigma_{\tilde{v}}$;
  - The measurement prediction itself is $\hat{z}_k = C_k \hat{x}_k^- + D_k u_k$;
  - The innovation is $\tilde{z}_k = z_k - \hat{z}_k$.
- A measurement validation gate can be set up around the measurement using normalized estimation error squared (NEES)

$$e_k^2 = \tilde{z}_k^T \Sigma_{\tilde{z},k}^{-1} \tilde{z}_k.$$

- NEES e_k^2 varies as a Chi-squared distribution with m degrees of freedom, where m is the dimension of z_k.
- If e_k^2 is outside the bounding values from the Chi-squared distribution for a desired confidence level, the measurement is discarded.
- Note: If a number of measurements are discarded in a short time interval, it may be that the sensor has truly failed, or that the state estimate and its covariance has gotten "lost."
- It is sometimes helpful to "bump up" the covariance $\Sigma_{\tilde{x},k}^{\pm}$, which simulates additional process noise, to help the Kalman filter to reacquire.

Chi-squared test

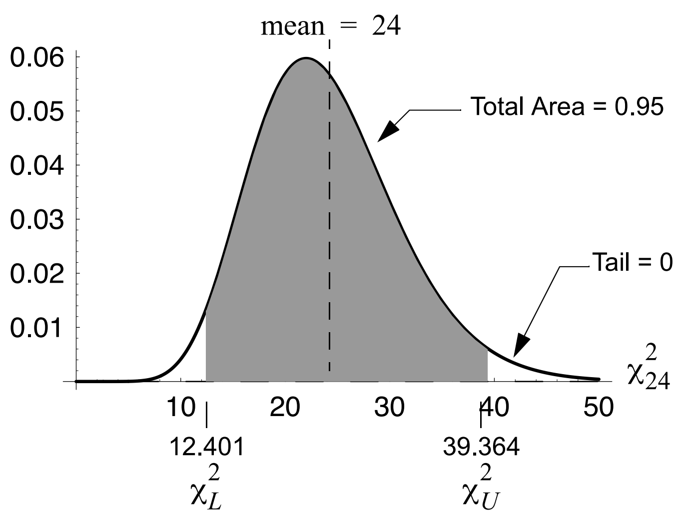
- A chi-square random variable is defined as a sum of squares of independent unit variance zero mean normal random variables.

$$Y = \sum_{i=1}^{n} \left(\frac{X_i - \mathbb{E}[X_i]}{\sigma} \right)^2.$$

- Y is *chi-square* with n degrees of freedom.
- Since it is a sum of squares, it is never negative and is not symmetrical about its mean value.
- The pdf of Y with n degrees of freedom is

$$f_Y(y) = \frac{1}{2^{n/2}\Gamma(n/2)} y^{(n/2-1)} e^{-n/2}.$$

- For confidence interval estimation we need to find two critical values.



Critical values for 95% confidence and χ^2 with 24 degrees of freedom

- If we want $1-\alpha$ confidence that the measurement is valid, we want to make sure that Y is between χ^2_L and χ^2_U where χ^2_L is calculated using $\alpha/2$ and χ^2_U is calculated using $1-\alpha/2$.

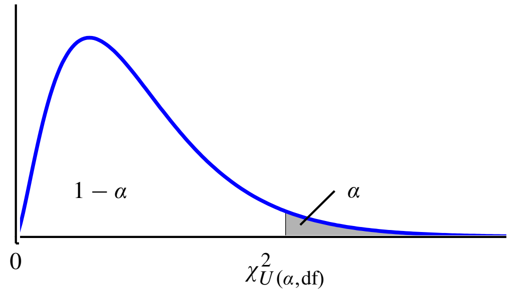
- The χ^2 pdf, cdf, and inverse cdf are available in most analysis software packages (*e.g.*, MATLAB, Mathematica, and even the spreadsheet program Excel).
- For hand calculations a χ^2-table is available on page **??**.

TRICKS WITH MATLAB: MATLAB may also be used to find the χ^2 values if a table is not convenient. This requires the MATLAB *statistics toolbox:*

```
>> help chi2inv

 CHI2INV Inverse of the chi-square cumulative distribution function (cdf).
 X = CHI2INV(P,V)  returns the inverse of the chi-square cdf with V
 degrees of freedom at the values in P. The chi-square cdf with V
 degrees of freedom, is the gamma cdf with parameters V/2 and 2.
 The size of X is the common size of P and V. A scalar input
 functions as a constant matrix of the same size as the other input.

>> % want to compute values for alpha = 0.05
>> chi2inv(1-.025,24) % Tail probability of alpha/2=0.025, n = 24. Upper
 critical value

ans =  39.3641

>> chi2inv(.025,24) % Lower critical value

ans =   12.4012
```

Appendix: Critical Values of χ^2

- For some deg. of freedom, each entry represents the critical value of χ^2 for a specified upper tail area α.



Degrees of Freedom	Upper Tail Areas 0.995	0.99	0.975	0.95	0.90	0.75	0.25	0.10	0.05	0.025	0.01	0.005
1	0.000	0.000	0.001	0.004	0.016	0.102	1.323	2.706	3.841	5.024	6.635	7.879
2	0.010	0.020	0.051	0.103	0.211	0.575	2.773	4.605	5.991	7.378	9.210	10.597
3	0.072	0.115	0.216	0.352	0.584	1.213	4.108	6.251	7.815	9.348	11.345	12.838
4	0.207	0.297	0.484	0.711	1.064	1.923	5.385	7.779	9.488	11.143	13.277	14.860
5	0.412	0.554	0.831	1.145	1.610	2.675	6.626	9.236	11.070	12.833	15.086	16.750
6	0.676	0.872	1.237	1.635	2.204	3.455	7.841	10.645	12.592	14.449	16.812	18.548
7	0.989	1.239	1.690	2.167	2.833	4.255	9.037	12.017	14.067	16.013	18.475	20.278
8	1.344	1.646	2.180	2.733	3.490	5.071	10.219	13.362	15.507	17.535	20.090	21.955
9	1.735	2.088	2.700	3.325	4.168	5.899	11.389	14.684	16.919	19.023	21.666	23.589
10	2.156	2.558	3.247	3.940	4.865	6.737	12.549	15.987	18.307	20.483	23.209	25.188
11	2.603	3.053	3.816	4.575	5.578	7.584	13.701	17.275	19.675	21.920	24.725	26.757
12	3.074	3.571	4.404	5.226	6.304	8.438	14.845	18.549	21.026	23.337	26.217	28.300
13	3.565	4.107	5.009	5.892	7.042	9.299	15.984	19.812	22.362	24.736	27.688	29.819
14	4.075	4.660	5.629	6.571	7.790	10.165	17.117	21.064	23.685	26.119	29.141	31.319
15	4.601	5.229	6.262	7.261	8.547	11.037	18.245	22.307	24.996	27.488	30.578	32.801
16	5.142	5.812	6.908	7.962	9.312	11.912	19.369	23.542	26.296	28.845	32.000	34.267
17	5.697	6.408	7.564	8.672	10.085	12.792	20.489	24.769	27.587	30.191	33.409	35.718
18	6.265	7.015	8.231	9.390	10.865	13.675	21.605	25.989	28.869	31.526	34.805	37.156
19	6.844	7.633	8.907	10.117	11.651	14.562	22.718	27.204	30.144	32.852	36.191	38.582
20	7.434	8.260	9.591	10.851	12.443	15.452	23.828	28.412	31.410	34.170	37.566	39.997
21	8.034	8.897	10.283	11.591	13.240	16.344	24.935	29.615	32.671	35.479	38.932	41.401
22	8.643	9.542	10.982	12.338	14.041	17.240	26.039	30.813	33.924	36.781	40.289	42.796
23	9.260	10.196	11.689	13.091	14.848	18.137	27.141	32.007	35.172	38.076	41.638	44.181
24	9.886	10.856	12.401	13.848	15.659	19.037	28.241	33.196	36.415	39.364	42.980	45.559
25	10.520	11.524	13.120	14.611	16.473	19.939	29.339	34.382	37.652	40.646	44.314	46.928
26	11.160	12.198	13.844	15.379	17.292	20.843	30.435	35.563	38.885	41.923	45.642	48.290
27	11.808	12.879	14.573	16.151	18.114	21.749	31.528	36.741	40.113	43.195	46.963	49.645
28	12.461	13.565	15.308	16.928	18.939	22.657	32.620	37.916	41.337	44.461	48.278	50.993
29	13.121	14.256	16.047	17.708	19.768	23.567	33.711	39.087	42.557	45.722	49.588	52.336
30	13.787	14.953	16.791	18.493	20.599	24.478	34.800	40.256	43.773	46.979	50.892	53.672
31	14.458	15.655	17.539	19.281	21.434	25.390	35.887	41.422	44.985	48.232	52.191	55.003
32	15.134	16.362	18.291	20.072	22.271	26.304	36.973	42.585	46.194	49.480	53.486	56.328
33	15.815	17.074	19.047	20.867	23.110	27.219	38.058	43.745	47.400	50.725	54.776	57.648
34	16.501	17.789	19.806	21.664	23.952	28.136	39.141	44.903	48.602	51.966	56.061	58.964
35	17.192	18.509	20.569	22.465	24.797	29.054	40.223	46.059	49.802	53.203	57.342	60.275
36	17.887	19.233	21.336	23.269	25.643	29.973	41.304	47.212	50.998	54.437	58.619	61.581
37	18.586	19.960	22.106	24.075	26.492	30.893	42.383	48.363	52.192	55.668	59.893	62.883
38	19.289	20.691	22.878	24.884	27.343	31.815	43.462	49.513	53.384	56.896	61.162	64.181
39	19.996	21.426	23.654	25.695	28.196	32.737	44.539	50.660	54.572	58.120	62.428	65.476
40	20.707	22.164	24.433	26.509	29.051	33.660	45.616	51.805	55.758	59.342	63.691	66.766
45	24.311	25.901	28.366	30.612	33.350	38.291	50.985	57.505	61.656	65.410	69.957	73.166
50	27.991	29.707	32.357	34.764	37.689	42.942	56.334	63.167	67.505	71.420	76.154	79.490