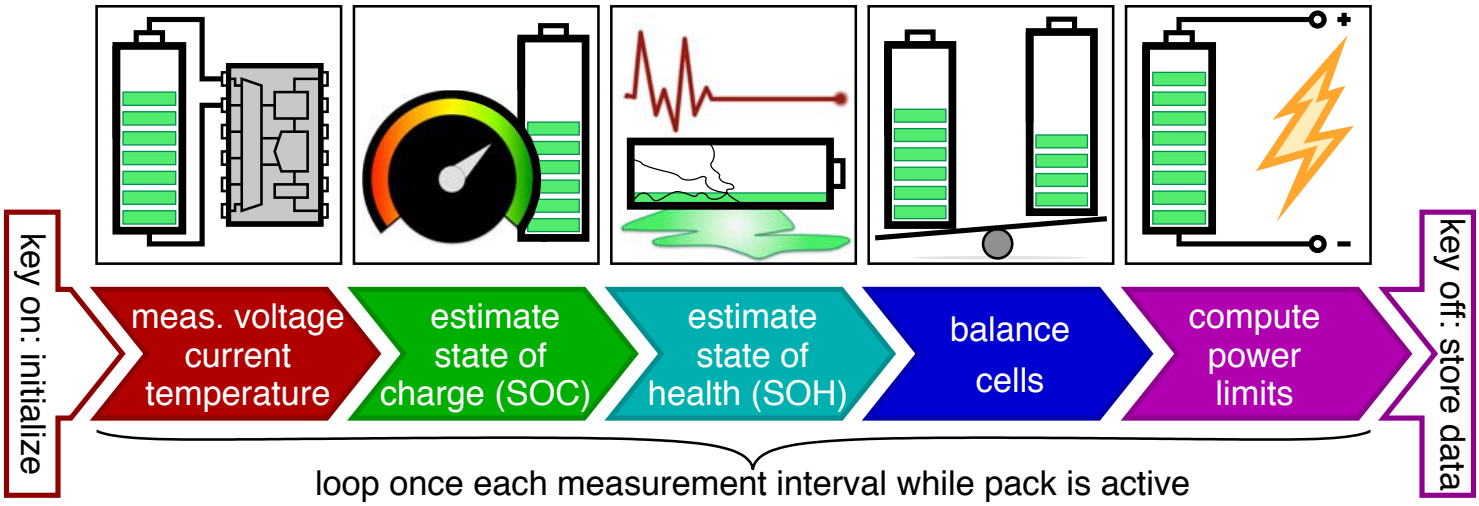
# *Battery State Estimation*

## 3.1: Preliminary definitions

- A battery management system must estimate quantities that
  - Describe the present battery pack condition, but
  - May not be directly measured.
- States are quantities that change quickly (*e.g.*, state-of-charge, diffusion voltage, hysteresis voltage).
- Parameters are quantities that change slowly (*e.g.*, cell capacities, resistances, aging effects).
- These quantities are typically updated in a program loop that looks something like:




- This chapter considers battery state estimation; the next chapter considers battery health estimation.

## State-of-charge (SOC) estimation

- An estimate of all battery-pack cells' SOC is an important input to balancing, energy, and power calculations.
- While we might be interested in estimating the entire battery-model state, we first focus on estimating state-of-charge only.
  - We'll see some simple methods that lack robustness.
  - Then, we examine methods that estimate the entire battery-model state, enabling some more advanced applications.
- Recall, SOC is something like a dashboard fuel gauge that reports a value from "Empty" (0 %) to "Full" (100 %).
- But, while there exist sensors to accurately measure a gasoline level in a tank, there is (presently) no sensor available to measure SOC.
- Further, accurate SOC estimates provide the following benefits:

**Longevity:** If a gas tank is over-filled or run empty, the tank is fine.

- However, over-charging or over-discharging a battery cell may cause permanent damage and result in reduced lifetime.
- An accurate SOC estimate may be used to avoid harming cells by not permitting current to be passed that would cause damage.

**Performance:** Without a good SOC estimator, one must be overly conservative when using the battery pack to avoid over/undercharge due to trusting the poor estimate.

- With a good estimate, especially one with known error bounds, one can aggressively use the entire pack capacity.

**Reliability:** Poor estimators behave differently for different use profiles.

- A good SOC estimator is consistent and dependable for any driving profile, enhancing overall power-system reliability.

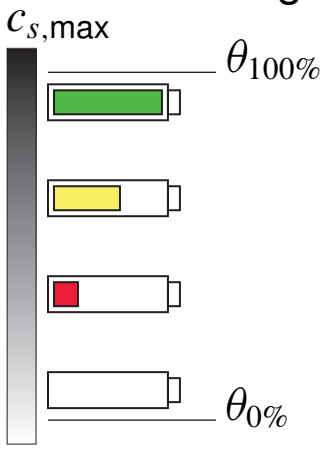
**Density:** Accurate SOC and battery state information allows the battery pack to be used aggressively within the design limits, so the pack does not need to be over-engineered.

- This allows smaller, lighter battery packs.

**Economy:** Smaller battery systems cost less. Warranty service on a reliable system costs less.

## A careful definition of state-of-charge

- Chapter 1 introduced an electrochemical definition of state-of-charge.
- We defined the present lithium concentration stoichiometry as $\theta = c_{s,\text{avg}}/c_{s,\text{max}}$.
- This stoichiometry is intended to remain between $\theta_{0\%}$ and $\theta_{100\%}$.
- Then, cell SOC is computed as: $z_k = (\theta_k - \theta_{0\%})/(\theta_{100\%} - \theta_{0\%})$.




- The issue addressed here is that there is (presently) no direct way to measure the concentrations that would allow us to calculate the SOC.
- So, we must infer or estimate the SOC using measurements of only cell terminal voltage and cell current.
- We've already noticed that while cell OCV is closely related to SOC, the terminal voltage is a poor predictor of OCV unless the cell is in electrochemical equilibrium (and hysteresis is negligible).

- So, how can we know the true SOC to evaluate our estimators? How can we know true SOC for any other purpose?

**KEY POINT:** We are aided by some definitions that can calibrate our tests.

**DEFINITION:** A cell is fully charged when its open circuit voltage (OCV) reaches $v_h(T)$, a manufacturer specified voltage that may be a function of temperature $T$.

- *e.g.*, $v_h(25\,^\circ\text{C}) = 4.2\,\text{V}$ for LMO; $v_h(25\,^\circ\text{C}) = 3.6\,\text{V}$ for LFP.
- A common method to bring a cell to a fully charged state is to execute a constant-current charge profile until the terminal voltage is equal to $v_h(T)$, followed by a constant-voltage profile until the charging current becomes infinitesimal.
- We define the SOC of a fully charged cell to be $100\,\%$.

**DEFINITION:** A cell is fully discharged when its OCV reaches $v_l(T)$, a manufacturer specified voltage that may be a function of temperature.

- *e.g.*, $v_l(25\,^\circ\text{C}) = 3.0\,\text{V}$ for LMO; $v_l(25\,^\circ\text{C}) = 2.0\,\text{V}$ for LFP.
- A cell may be fully discharged by executing a constant-current discharge profile until its terminal voltage is equal to $v_l(T)$, followed by a constant-voltage profile until the discharge current becomes infinitesimal.
- We define the SOC of a fully discharged cell to be $0\,\%$.

**DEFINITION:** The total capacity $Q$ of a cell is the quantity of charge removed from a cell as it is brought from a fully charged state to a fully discharged state.

- While the SI unit for charge is coulombs (C), it is more common in practice to use units of ampere hours (Ah) or milliampere hours (mAh) to measure the total capacity of a battery cell.
- The total capacity of a cell is not a fixed quantity: it generally decays slowly over time as the cell degrades.

**DEFINITION:** The discharge capacity $Q_{[\text{rate}]}$ of a cell is the quantity of charge removed from a cell as it is discharged at a constant rate from a fully charged state until its loaded terminal voltage reaches $v_l(T)$.

- Because the discharge capacity is determined based on loaded terminal voltage rather than open circuit voltage, it is strongly dependent on the cell's internal resistance, which itself is a function of rate and temperature.
- Hence, the discharge capacity of a cell is rate dependent and temperature dependent.
- Because of the resistive $i(t) \times R_0$ drop, discharge capacity is less than total capacity unless the discharge rate is infinitesimal.
- Likewise, the SOC of the cell is nonzero when the terminal voltage reaches $v_l(T)$ at a non-infinitesimal rate.
- The discharge capacity of a cell at a particular rate and temperature is not a fixed quantity: it also generally decays slowly over time as the cell degrades.

**DEFINITION:** The nominal capacity $Q_{\text{nom}}$ of a cell is a manufacturer-specified quantity that is intended to be representative of the 1C-rate discharge capacity $Q_{1C}$ of a particular manufactured lot of cells at room temperature, 25 °C.

- The nominal capacity is a constant value.
- Since the nominal capacity is representative of a lot of cells and the discharge capacity is representative of a single individual cell, $Q_{\text{nom}} \neq Q_{1C}$ in general, even at beginning of life.
- Also, since $Q_{\text{nom}}$ is representative of a discharge capacity and not a total capacity, $Q_{\text{nom}} \neq Q$.

**DEFINITION:** The residual capacity of a cell is the quantity of charge that would be removed from a cell if it were brought from its present state to a fully discharged state.

**DEFINITION:** The state-of-charge of the cell is the ratio of the residual capacity to the total capacity of the cell.

- These definitions are consistent with the relationships

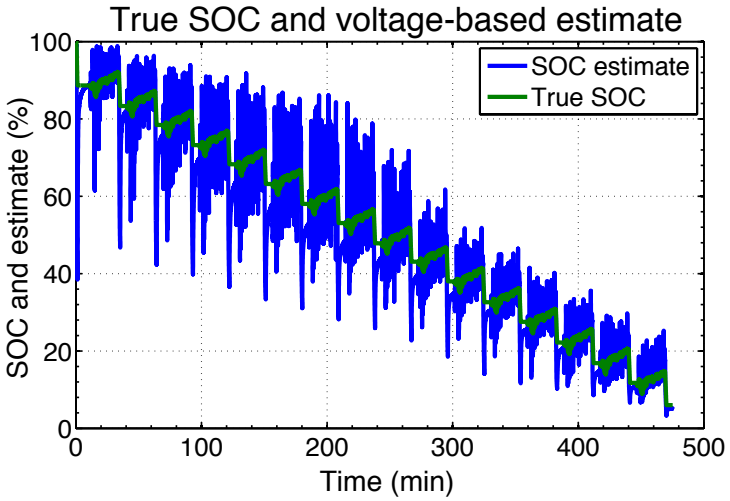
$$z(t) = z(0) - \frac{1}{Q}\int_0^t \eta(t) i(t)\,\mathrm{d}t, \quad \text{and} \quad z_{k+1} = z_k - \eta_k i_k \Delta t / Q$$

that we have already seen.

## 3.2: Some approaches to estimate state of charge

*Poor, voltage-based methods to estimate SOC*

- Measure cell terminal voltage under load $v(t)$ and look up on "SOC versus OCV" curve
  - Misses effects of $i(t) \times R_0$ losses, diffusion voltages, hysteresis
  - Wide flat areas of OCV curve dilute accuracy of estimate
- The "Tino" method assumes a cell model $v(t) = \text{OCV}(z(t)) - i(t)R_0$ and then looks up $v(t) + i(t)R_0$ on "SOC versus OCV" curve
  - Better, but still misses diffusion voltages, hysteresis




- Example shows that Tino estimate is very noisy.
- Filtering helps, but adds delay, which must be accounted for.
- Hysteresis is another complicating factor.
- Even though its estimates are noisy, we'll find an application for the Tino method in the next chapter of notes.

*Poor, current-based method to estimate SOC*

- Coulomb counting keeps track of charge in, out of cells via

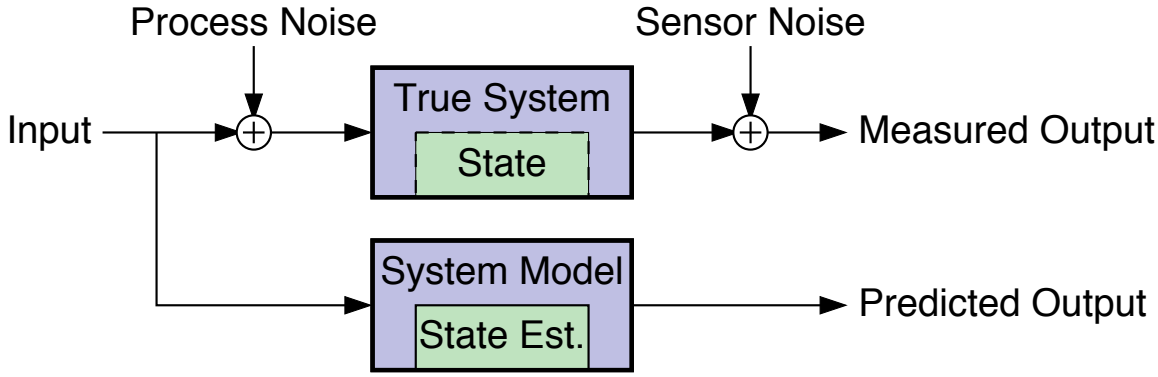
$$\hat{z}(t) = \hat{z}(0) - \frac{1}{\widehat{Q}} \int_0^t i_{\text{meas}}(\tau)\, \mathrm{d}\tau$$

$$i_{\text{meas}}(t) = i_{\text{true}}(t) + i_{\text{noise}}(t) + i_{\text{bias}}(t) + i_{\text{nonlin}}(t) + i_{\text{sd}}(t) + i_{\text{leakage}}(t).$$

  - Okay for short periods of operation when initial conditions are known or can be frequently “reset”;
  - Subject to drift due to current sensor’s fluctuations, current-sensor bias, incorrect capacity estimate, other losses;
- Uncertainty/error bounds grow over time, increasing without bound until estimate is “reset”.

*Model-based state estimation*

- An alternative to a voltage-only method or a current-only method is to somehow combine the approaches.
- Model-based state estimators implement algorithms that use sensor measurements to infer the internal hidden state of a dynamic system.




  - A mathematical model of the system is assumed known.
  - Same input propagated through true system and model.
  - Measured and predicted outputs compared; error used to update model’s estimate of the true state:
    - Output error due to: state, measurement, model errors;
    - Update must be done carefully to account for all of these.
- Under some specific conditions, the Kalman filter (a special case of sequential probabilistic inference) gives the optimal state estimate.

- We will look at the linear Kalman filter and some of its variants throughout the remainder of this chapter.

## *Sequential probabilistic inference*

- We start by assuming a general, possibly nonlinear, model

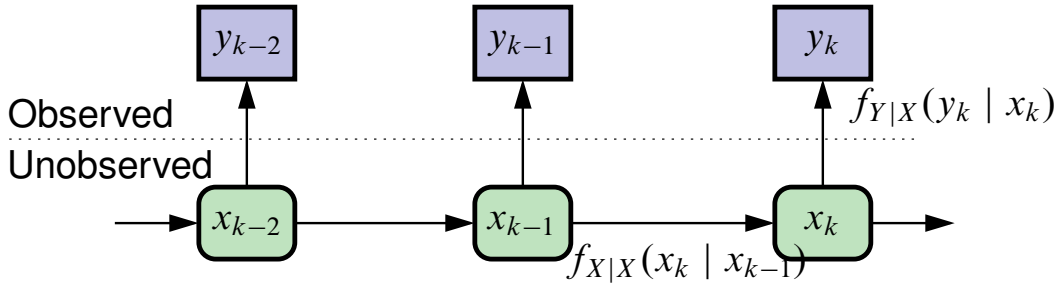
$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1})$$

$$y_k = h(x_k, u_k, v_k),$$

  where $u_k$ is a known (deterministic/measured) input signal, $w_k$ is a process-noise random input, and $v_k$ is a sensor-noise random input.

- We note that $f(\cdot)$ and $h(\cdot)$ may be time-varying, but we generally omit the time dependency from the notation for ease of understanding.

**SEQUENTIAL PROBABILISTIC INFERENCE:** Estimate the present state $x_k$ of a dynamic system using all measurements $\mathbb{Y}_k = \{y_0, y_1, \cdots, y_k\}$.




- The observations allow us to "peek" at what is happening in the true system. Based on observations and our model, we estimate the state.

- However, process-noise and sensor-noise randomness cause us never to be able to compute the state exactly.

- So, to be able to talk about the sequential-probabilistic-inference solution, we first must look at some topics in vector random variables and random processes.

## 3.3: Review of probability

- By definition, noise is not deterministic—it is random in some sense.
- So, to discuss the impact of noise on the system dynamics, we must understand "random variables" (RVs).
  - Cannot predict exactly what we will get each time we measure or sample the random variable, but
  - We can characterize the probability of each sample value by the "probability density function" (pdf).
- For a brief review, define random vector $X$ and sample vector $x_0$ as

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}, \qquad x_0 = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix},$$

  where $X_1$ through $X_n$ are scalar random variables and $x_1$ through $x_n$ are scalar constants.
- $X$ described by (scalar function) joint pdf $f_X(x)$ of vector $X$.
  - $f_X(x_0)$ means $f_X(X_1 = x_1, X_2 = x_2, \cdots, X_n = x_n)$.
  - That is, $f_X(x_0)\,\mathrm{d}x_1\,\mathrm{d}x_2\,\cdots\,\mathrm{d}x_n$ is the probability that $X$ is between $x_0$ and $x_0 + \mathrm{d}x$.
- Properties of joint pdf $f_X(x)$:
  1. $f_X(x) \geq 0 \quad \forall \quad x.$
  2. $\displaystyle\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} f_X(x)\,\mathrm{d}x_1\,\mathrm{d}x_2\cdots\mathrm{d}x_n = 1.$
  3. $\displaystyle\bar{x} = \mathbb{E}[X] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} x f_X(x)\,\mathrm{d}x_1\,\mathrm{d}x_2\cdots\mathrm{d}x_n.$

4. Correlation matrix:

$$\Sigma_X = \mathbb{E}[XX^T] \qquad \text{(outer product)}$$
$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} xx^T f_X(x)\,\mathrm{d}x_1\,\mathrm{d}x_2\cdots\mathrm{d}x_n.$$

5. Covariance matrix: Define $\widetilde{X} = X - \bar{x}$. Then,

$$\Sigma_{\widetilde{X}} = \mathbb{E}[(X-\bar{x})(X-\bar{x})^T]$$
$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} (x-\bar{x})(x-\bar{x})^T f_X(x)\,\mathrm{d}x_1\,\mathrm{d}x_2\cdots\mathrm{d}x_n.$$

$\Sigma_{\widetilde{X}}$ is symmetric and positive-semi-definite (psd). This means

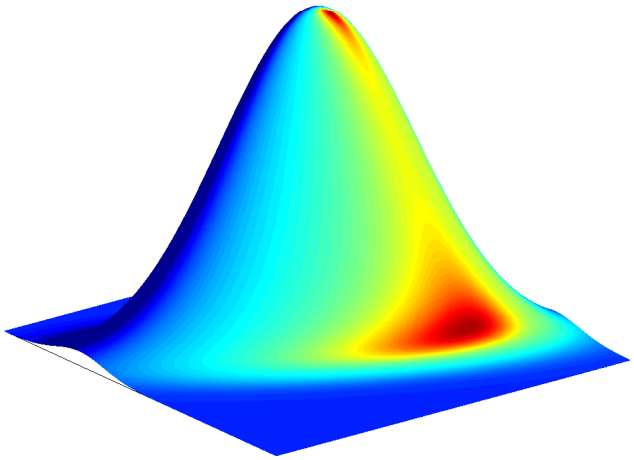
$$y^T\Sigma_{\widetilde{X}}y \geq 0 \quad \forall \quad y.$$

- Notice that correlation = covariance for zero-mean random vectors.
- The covariance entries have specific meaning:

$$(\Sigma_{\widetilde{X}})_{ii} = \sigma^2_{X_i}$$
$$(\Sigma_{\widetilde{X}})_{ij} = \rho_{ij}\sigma_{X_i}\sigma_{X_j} = (\Sigma_{\widetilde{X}})_{ji}.$$

  - The diagonal entries are the variances of each vector component;
  - The correlation coefficient $\rho_{ij}$ is a measure of linear dependence between $X_i$ and $X_j$. $|\rho_{ij}| \leq 1$.

- There are infinite variety in pdfs.
- However, we use only the (multivariable) Gaussian pdf when defining the Kalman filter.
- All noises and the state vector itself are assumed to be Gaussian random vectors.



- The Gaussian or normal pdf is defined as, where we say $X \sim \mathcal{N}(\bar{x}, \Sigma_{\widetilde{X}})$

$$f_X(x) = \frac{1}{(2\pi)^{n/2}\,|\Sigma_{\widetilde{X}}|^{1/2}} \exp\left(-\frac{1}{2}(x-\bar{x})^T \Sigma_{\widetilde{X}}^{-1}(x-\bar{x})\right).$$

$$|\Sigma_{\widetilde{X}}| = \det(\Sigma_{\widetilde{X}}), \quad \Sigma_{\widetilde{X}}^{-1} \text{ requires positive-definite } \Sigma_{\widetilde{X}}.$$

- Contours of constant $f_X(x)$ are hyper-ellipsoids, centered at $\bar{x}$, directions governed by $\Sigma_{\widetilde{X}}$.

*Properties of jointly-distributed RVs*

**INDEPENDENCE:** Iff jointly-distributed RVs are independent, then

$$f_X(x_1, x_2, \ldots, x_n) = f_{X_1}(x_1) f_{X_2}(x_2) \cdots f_{X_n}(x_n).$$

- Joint distribution can be split up into the product of individual distributions for each RV.
  - "The particular value of the random variable $X_1$ has no impact on what value we would obtain for the random variable $X_2$."

**UNCORRELATED:** Two jointly-distributed R.V.s $X_1$ and $X_2$ are uncorrelated if their second moments are finite and

$$\mathrm{cov}(X_1, X_2) = \mathbb{E}[(X_1 - \bar{x}_1)(X_2 - \bar{x}_2)] = 0$$

which implies $\rho_{12} = 0$.

- Uncorrelated means that there is no linear relationship between RVs.

**MAIN POINT #1:** If jointly-distributed RVs $X_1$ and $X_2$ are independent then they must also be uncorrelated. Independence implies uncorrelation. However, uncorrelated RVs are not necessarily independent.

**MAIN POINT #2:** If jointly normally distributed RVs are uncorrelated, then they are independent. This is a (very) special case.

**MAIN POINT #3:** We can define a conditional pdf

$$f_{X|Y}(x|y) = \frac{f_{X,Y}(x,y)}{f_Y(y)}$$

as the probability that $X = x$ given that $Y = y$ has happened.

**NOTE:** The marginal probability $f_Y(y)$ may be calculated as

$$f_Y(y) = \int_{-\infty}^{\infty} f_{X,Y}(x,y)\,\mathrm{d}x.$$

For each $y$, integrate out the effect of $X$.

**DIRECT EXTENSION:**

$$\begin{aligned} f_{X,Y}(x,y) &= f_{X|Y}(x|y) f_Y(y) \\ &= f_{Y|X}(y|x) f_X(x), \end{aligned}$$

Therefore,

$$f_{X|Y}(x|y) = \frac{f_{Y|X}(y|x) f_X(x)}{f_Y(y)}.$$

- This is known as Bayes' rule. It relates the posterior probability to the prior probability.
- It forms a key step in the Kalman filter derivation.

**MAIN POINT #4:** We can define conditional expectation as what we expect the value of $X$ to be given that $Y = y$ has happened

$$\mathbb{E}[X = x|Y = y] = \mathbb{E}[X|Y] = \int_{-\infty}^{\infty} x f_{X|Y}(x|Y)\,\mathrm{d}x.$$

- Note: Conditional expectation is critical. The Kalman filter is an algorithm to compute $\mathbb{E}[x_k \mid \mathbb{Y}_k]$, where we define $\mathbb{Y}_k$ later.

**MAIN POINT #5:** Central Limit Theorem.

- If $Y = \sum_i X_i$ and the $X_i$ are independent and identically distributed (IID), and the $X_i$ have finite mean and variance, then $Y$ will be approximately normally distributed.
- The approximation improves as the number of summed RVs gets large.
- Since the state of our dynamic system adds up the effects of lots of independent random inputs, it is reasonable to assume that the distribution of the state tends to the normal distribution.
- This leads to the key assumptions for the derivation of the Kalman filter, as we will see:
  - We will assume that the state $x_k$ is a normally distributed random vector;
  - We will assume that the process noise $w_k$ is a normally distributed random vector;
  - We will assume that the sensor noise $v_k$ is a normally distributed random vector;
  - We will assume that $w_k$ and $v_k$ are uncorrelated with each other.
- Even when these assumptions are broken in practice, the Kalman filter works quite well.
- Exceptions tend to be with very highly nonlinear systems, for which particle filters must sometimes be employed to get good estimates.

**MAIN POINT #6:** A linear combination of Gaussian RVs results in a Gaussian RV.

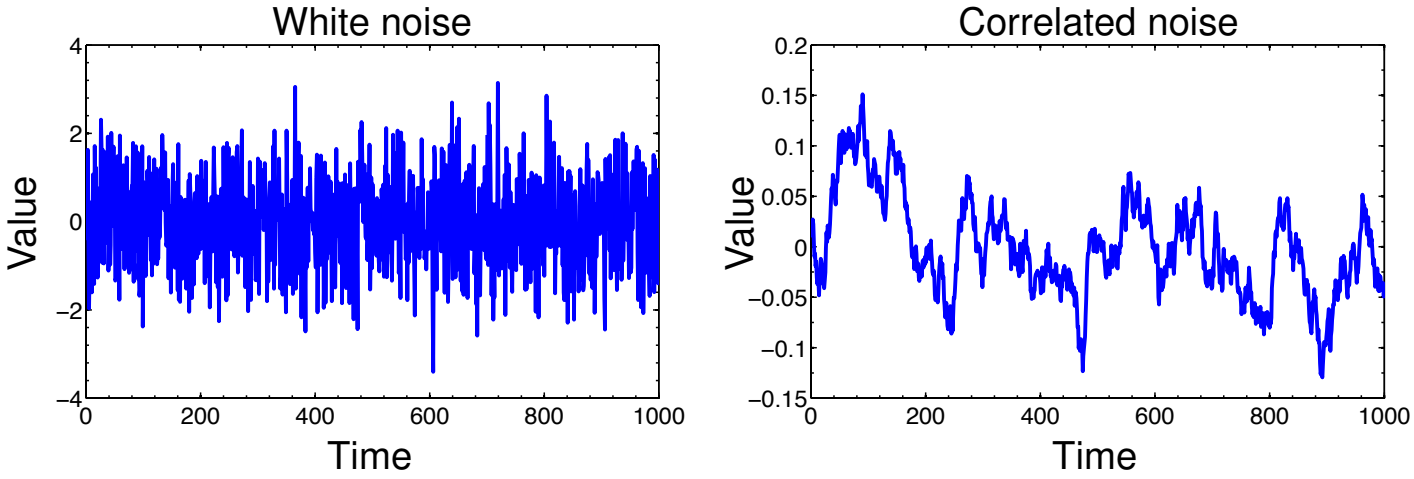
## 3.4: Overview of vector random (stochastic) processes

- A stochastic or random process is a family of random vectors indexed by a parameter set ("time" in our case).
  - For example, we might refer to a random process $X_k$ for generic $k$.
  - The value of the random process at any specific time $k = m$ is a random variable $X_m$.
- Usually assume stationarity.
  - The statistics (*i.e.,* pdf) of the RV are time-shift invariant.
  - Therefore, $\mathbb{E}[X_k] = \bar{x}$ for all $k$ and $\mathbb{E}[X_{k_1} X_{k_2}^T] = R_X(k_1 - k_2)$.

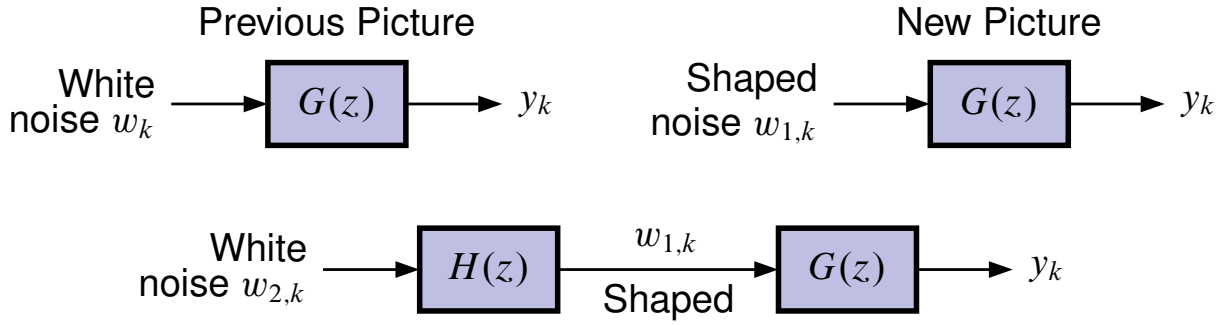
*Properties and important points*

1. Autocorrelation: $R_X(k_1, k_2) = \mathbb{E}[X_{k_1} X_{k_2}^T]$. If stationary,
$$R_X(\tau) = \mathbb{E}[X_k X_{k+\tau}^T].$$
   - Provides a measure of correlation between elements of the process having time displacement $\tau$.
   - $R_X(0) = \sigma_X^2$ for zero-mean $X$.
   - $R_X(0)$ is always the maximum value of $R_X(\tau)$.
2. Autocovariance: $C_X(k_1, k_2) = \mathbb{E}[(X_{k_1} - \mathbb{E}[X_{k_1}])(X_{k_2} - \mathbb{E}[X_{k_2}])^T]$. If stationary,
$$C_X(\tau) = \mathbb{E}[(X_k - \bar{x})(X_{k+\tau} - \bar{x})^T].$$
3. White noise: Some processes have a unique autocorrelation:
   (a) Zero mean,
   (b) $R_X(\tau) = \mathbb{E}[X_k X_{k+\tau}^T] = S_X \delta(\tau)$ where $\delta(\tau)$ is the Dirac delta. $\delta(\tau) = 0 \,\forall\, \tau \neq 0$.

- Therefore, the process is uncorrelated in time.
- Clearly an abstraction, but proves to be a very useful one.




4. Shaping filters: We will assume that the noise inputs to the dynamic systems are white (Gaussian) processes.
   - Pretty limiting assumption, but one that can be easily fixed ➠ Can use second linear system to "shape" the noise as desired.




- Therefore, we can drive our linear system with noise that has a desired characteristics by introducing a shaping filter $H(z)$ that itself is driven by white noise.
- The combined system $GH(z)$ looks exactly the same as before, *but* the system $G(z)$ is not driven by pure white noise any more.

- Analysis *augments* original system model with filter states. Original system has

$$x_{k+1} = Ax_k + B_w w_{1,k}$$

$$y_k = Cx_k.$$

- Shaping filter with white input and desired output statistics has

$$x_{s,k+1} = A_s x_{s,k} + B_s w_{2,k}$$

$$w_{1,k} = C_s x_{s,k}.$$

- Combine into one system:

$$\left[\begin{array}{c:c} x_{k+1} \\ \hdashline x_{s,k+1} \end{array}\right] = \left[\begin{array}{c:c} A & B_w C_s \\ \hdashline 0 & A_s \end{array}\right] \left[\begin{array}{c:c} x_k \\ \hdashline x_{s,k} \end{array}\right] + \left[\begin{array}{c:c} 0 \\ \hdashline B_s \end{array}\right] w_{2,k}$$

$$y_k = \left[\begin{array}{c:c} C & 0 \end{array}\right] \left[\begin{array}{c:c} x_k \\ \hdashline x_{s,k} \end{array}\right].$$

- Augmented system just a larger-order system driven by white noise.

5. Gaussian processes: We will work with Gaussian noises to a large extent, which are uniquely defined by the first- and second central moments of the statistics ➠ Gaussian assumption not essential.
   - Our filters will always track only the first two moments.

**NOTATION:** Until now, we have always used capital letters for random variables. The state of a system driven by a random process is a random vector, so we could now call it $X_k$. However, it is more common to retain the standard notation $x_k$ and understand from the context that we are discussing an RV.

## 3.5 Sequential-probabilistic-inference solution

- In the notation that follows,
  - The superscript "$-$" indicates a predicted quantity based only on past measurements.
  - The superscript "$+$" indicates an estimated quantity based on both past and present measurements.
  - The decoration "$\hat{}$" indicates a predicted or estimated quantity.
  - The decoration "$\tilde{}$" indicates an error: the difference between a true and predicted or estimated quantity.
  - The symbol "$\Sigma$" is used to denote the correlation between the two arguments in its subscript (autocorrelation if only one is given).

$$\Sigma_{xy} = \mathbb{E}[xy^T] \qquad \text{and} \qquad \Sigma_x = \mathbb{E}[xx^T].$$

  - Furthermore, if the arguments are zero mean (as they often are in the quantities we talk about), then this represents covariance.

$$\begin{aligned}\Sigma_{\tilde{x}\tilde{y}} &= \mathbb{E}[\tilde{x}\tilde{y}^T] \\ &= \mathbb{E}[(\tilde{x} - \mathbb{E}[\tilde{x}])(\tilde{y} - \mathbb{E}[\tilde{y}])^T],\end{aligned}$$

    for zero-mean $\tilde{x}$ and $\tilde{y}$.

- We choose to find a state estimate that minimizes the "mean-squared error"

$$\begin{aligned}\hat{x}_k^{\text{MMSE}}(\mathbb{Y}_k) &= \arg\min_{\hat{x}_k} \left( \mathbb{E}\left[ \left\| x_k - \hat{x}_k^+ \right\|_2^2 \mid \mathbb{Y}_k \right] \right) \\ &= \arg\min_{\hat{x}_k} \left( \mathbb{E}\left[ (x_k - \hat{x}_k^+)^T (x_k - \hat{x}_k^+) \mid \mathbb{Y}_k \right] \right) \\ &= \arg\min_{\hat{x}_k} \left( \mathbb{E}\left[ x_k^T x_k - 2x_k^T \hat{x}_k^+ + (\hat{x}_k^+)^T \hat{x}_k^+ \mid \mathbb{Y}_k \right] \right).\end{aligned}$$

- We solve for $\hat{x}_k^+$ by differentiating the cost function and setting the result to zero

$$0 = \frac{\mathrm{d}}{\mathrm{d}\hat{x}_k^+}\mathbb{E}\Big[x_k^T x_k - 2x_k^T \hat{x}_k^+ + (\hat{x}_k^+)\hat{x}_k^+ \mid \mathbb{Y}_k\Big].$$

- To do so, note the following identities from vector calculus,

$$\frac{\mathrm{d}}{\mathrm{d}X}Y^T X = Y, \qquad \frac{\mathrm{d}}{\mathrm{d}X}X^T Y = Y, \quad \text{and} \quad \frac{\mathrm{d}}{\mathrm{d}X}X^T AX = (A + A^T)X.$$

- Then,

$$0 = \mathbb{E}\Big[-2(x_k - \hat{x}_k^+) \mid \mathbb{Y}_k\Big] = 2\hat{x}_k^+ - 2\mathbb{E}\Big[x_k \mid \mathbb{Y}_k\Big]$$

$$\hat{x}_k^+ = \mathbb{E}\Big[x_k \mid \mathbb{Y}_k\Big].$$

- We proceed by assuming that all RVs have a Gaussian distribution. We will find that the result has a predict/correct mechanism.

- So, with malice aforethought, we define prediction error $\tilde{x}_k^- = x_k - \hat{x}_k^-$ where $\hat{x}_k^- = \mathbb{E}\Big[x_k \mid \mathbb{Y}_{k-1}\Big]$.
  - Error is always "truth minus prediction" or "truth minus estimate."
  - We can't compute error in practice, since truth value is not known.
  - But, we can prove statistical results using this definition that give an algorithm for estimating the truth using measurable values.

- Also, define the measurement innovation (what is new or unexpected in the measurement) as $\tilde{y}_k = y_k - \hat{y}_k$ where $\hat{y}_k = \mathbb{E}\Big[y_k \mid \mathbb{Y}_{k-1}\Big]$.

- Both $\tilde{x}_k^-$ and $\tilde{y}_k$ can be shown to be zero mean using the method of iterated expectation: $\mathbb{E}\Big[\mathbb{E}\Big[X \mid Y\Big]\Big] = \mathbb{E}\Big[X\Big]$.

$$\mathbb{E}\Big[\tilde{x}_k^-\Big] = \mathbb{E}\Big[x_k\Big] - \mathbb{E}\Big[\mathbb{E}\Big[x_k \mid \mathbb{Y}_{k-1}\Big]\Big] = \mathbb{E}\Big[x_k\Big] - \mathbb{E}\Big[x_k\Big] = 0$$

$$\mathbb{E}\Big[\tilde{y}_k\Big] = \mathbb{E}\Big[y_k\Big] - \mathbb{E}\Big[\mathbb{E}\Big[y_k \mid \mathbb{Y}_{k-1}\Big]\Big] = \mathbb{E}\Big[y_k\Big] - \mathbb{E}\Big[y_k\Big] = 0.$$

- Note also that $\tilde{x}_k^-$ is uncorrelated with past measurements as they have already been incorporated into $\hat{x}_k^-$

$$\mathbb{E}\big[\tilde{x}_k^- \mid \mathbb{Y}_{k-1}\big] = \mathbb{E}\big[x_k - \mathbb{E}\big[x_k \mid \mathbb{Y}_{k-1}\big] \mid \mathbb{Y}_{k-1}\big] = 0 = \mathbb{E}\big[\tilde{x}_k^-\big].$$

- Therefore, on one hand we can write the relationship

$$\mathbb{E}\big[\tilde{x}_k^- \mid \mathbb{Y}_k\big] = \underbrace{\mathbb{E}\big[x_k \mid \mathbb{Y}_k\big]}_{\hat{x}_k^+} - \underbrace{\mathbb{E}\big[\hat{x}_k^- \mid \mathbb{Y}_k\big]}_{\hat{x}_k^-}.$$

- On the other hand, we can write

$$\mathbb{E}\big[\tilde{x}_k^- \mid \mathbb{Y}_k\big] = \mathbb{E}\big[\tilde{x}_k^- \mid \mathbb{Y}_{k-1}, y_k\big] = \mathbb{E}\big[\tilde{x}_k^- \mid y_k\big].$$

- So,

$$\hat{x}_k^+ = \hat{x}_k^- + \mathbb{E}\big[\tilde{x}_k^- \mid y_k\big],$$

  which is a predict/correct sequence of steps, as promised.

- But, what is $\mathbb{E}\big[\tilde{x}_k^- \mid y_k\big]$? We can show that, generically, when $x$ and $y$ are jointly Gaussian distributed,

$$\mathbb{E}\big[x \mid y\big] = \mathbb{E}\big[x\big] + \Sigma_{\tilde{x}\tilde{y}}\Sigma_{\tilde{y}}^{-1}\big(y - \mathbb{E}\big[y\big]\big).$$

- Applying this to our problem, when $y_k = \tilde{y}_k + \hat{y}_k$, we get

$$\begin{aligned}
\mathbb{E}\big[\tilde{x}_k^- \mid y_k\big] &= \mathbb{E}\big[\tilde{x}_k^-\big] + \Sigma_{\tilde{x}\tilde{y},k}^-\Sigma_{\tilde{y},k}^{-1}\big(y_k - \mathbb{E}\big[y_k\big]\big) \\
&= \mathbb{E}\big[\tilde{x}_k^-\big] + \Sigma_{\tilde{x}\tilde{y},k}^-\Sigma_{\tilde{y},k}^{-1}\big(\tilde{y}_k + \hat{y}_k - \mathbb{E}\big[\tilde{y}_k + \hat{y}_k\big]\big) \\
&= 0 + \Sigma_{\tilde{x}\tilde{y},k}^-\Sigma_{\tilde{y},k}^{-1}\big(\tilde{y}_k + \hat{y}_k - (0 + \hat{y}_k)\big) \\
&= \underbrace{\Sigma_{\tilde{x}\tilde{y},k}^-\Sigma_{\tilde{y},k}^{-1}}_{L_k}\tilde{y}_k.
\end{aligned}$$

- Putting all of the pieces together, we get the general update equation:

$$\hat{x}_k^+ = \hat{x}_k^- + L_k\tilde{y}_k.$$

- Note that $L_k$ is a function of $\Sigma^+_{\tilde{x},k}$, which may be computed as

$$\begin{aligned}
\Sigma^+_{\tilde{x},k} &= \mathbb{E}\big[(x_k - \hat{x}_k^+)(x_k - \hat{x}_k^+)^T\big] \\
&= \mathbb{E}\big[\big\{(x_k - \hat{x}_k^-) - L_k\tilde{y}_k\big\}\big\{(x_k - \hat{x}_k^-) - L_k\tilde{y}_k\big\}^T\big] \\
&= \mathbb{E}\big[\big\{\tilde{x}_k^- - L_k\tilde{y}_k\big\}\big\{\tilde{x}_k^- - L_k\tilde{y}_k\big\}^T\big] \\
&= \Sigma^-_{\tilde{x},k} - L_k\underbrace{\mathbb{E}\big[\tilde{y}_k(\tilde{x}_k^-)^T\big]}_{\Sigma_{\tilde{y},k}L_k^T} - \underbrace{\mathbb{E}\big[\tilde{x}_k^-\tilde{y}_k^T\big]}_{L_k\Sigma_{\tilde{y},k}}L_k^T + L_k\Sigma_{\tilde{y},k}L_k^T \\
&= \Sigma^-_{\tilde{x},k} - L_k\Sigma_{\tilde{y},k}L_k^T.
\end{aligned}$$

## *Forest and trees*

- To be perfectly clear, the output of this process has two parts:
  1. The state estimate. At the end of every iteration, we have computed our best guess of the present state value, which is $\hat{x}_k^+$.
  2. The covariance estimate. The covariance matrix $\Sigma^+_{\tilde{x},k}$ gives the uncertainty of $\hat{x}_k^+$, and can be used to compute error bounds.

- Summarizing, the generic Gaussian sequential probabilistic inference recursion becomes:

$$\begin{aligned}
\hat{x}_k^+ &= \hat{x}_k^- + L_k\big(y_k - \hat{y}_k\big) = \hat{x}_k^- + L_k\tilde{y}_k \\
\Sigma^+_{\tilde{x},k} &= \Sigma^-_{\tilde{x},k} - L_k\Sigma_{\tilde{y},k}L_k^T,
\end{aligned}$$

where

$$\begin{aligned}
\hat{x}_k^- &= \mathbb{E}\big[x_k \mid \mathbb{Y}_{k-1}\big] & \Sigma^-_{\tilde{x},k} &= \mathbb{E}\big[(x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T\big] = \mathbb{E}\big[(\tilde{x}_k^-)(\tilde{x}_k^-)^T\big] \\
\hat{x}_k^+ &= \mathbb{E}\big[x_k \mid \mathbb{Y}_k\big] & \Sigma^+_{\tilde{x},k} &= \mathbb{E}\big[(x_k - \hat{x}_k^+)(x_k - \hat{x}_k^+)^T\big] = \mathbb{E}\big[(\tilde{x}_k^+)(\tilde{x}_k^+)^T\big] \\
\hat{z}_k &= \mathbb{E}\big[z_k \mid \mathbb{Y}_{k-1}\big] & \Sigma_{\tilde{y},k} &= \mathbb{E}\big[(y_k - \hat{y}_k)(y_k - \hat{y}_k)^T\big] = \mathbb{E}\big[(\tilde{y}_k)(\tilde{y}_k)^T\big] \\
L_k &= \mathbb{E}\big[(x_k - \hat{x}_k^-)(y_k - \hat{y}_k)^T\big]\Sigma^{-1}_{\tilde{y},k} = \Sigma^-_{\tilde{x}\tilde{y},k}\Sigma^{-1}_{\tilde{y},k}.
\end{aligned}$$

- Note that this is a linear recursion, even if the system is nonlinear(!)

## 3.6: The six-step process

- Generic Gaussian probabilistic inference solution can be divided into two main steps, each having three sub-steps.

**General step 1a:** State prediction time update.

- Each time step, compute an updated prediction of the present value of $x_k$, based on prior information and the system model

$$\hat{x}_k^- = \mathbb{E}\big[x_k \mid \mathbb{Y}_{k-1}\big] = \mathbb{E}\big[f(x_{k-1}, u_{k-1}, w_{k-1}) \mid \mathbb{Y}_{k-1}\big].$$

**General step 1b:** Error covariance time update.

- Determine the predicted state-estimate error covariance matrix $\Sigma_{\tilde{x},k}^-$ based on prior information and the system model.
- We compute $\Sigma_{\tilde{x},k}^- = \mathbb{E}\big[(\tilde{x}_k^-)(\tilde{x}_k^-)^T\big]$ , where $\tilde{x}_k^- = x_k - \hat{x}_k^-$.

**General step 1c:** Predict system output $y_k$.

- Predict the system's output using prior information

$$\hat{y}_k = \mathbb{E}\big[y_k \mid \mathbb{Y}_{k-1}\big] = \mathbb{E}\big[h(x_k, u_k, v_k) \mid \mathbb{Y}_{k-1}\big].$$

**General step 2a:** Estimator gain matrix $L_k$.

- Compute the estimator gain matrix $L_k$ by evaluating $L_k = \Sigma_{\tilde{x}\tilde{y},k}^- \Sigma_{\tilde{y},k}^{-1}$.

**General step 2b:** State estimate measurement update.

- Compute the posterior state estimate by updating the prediction using the $L_k$ and the innovation $y_k - \hat{y}_k$

$$\hat{x}_k^+ = \hat{x}_k^- + L_k(y_k - \hat{y}_k).$$

**General step 2c:** Error covariance measurement update.

- Compute the posterior error covariance matrix

$$\begin{aligned}\Sigma^{+}_{\tilde{x},k} &= \mathbb{E}\big[(\tilde{x}^{+}_k)(\tilde{x}^{+}_k)^T\big] \\ &= \Sigma^{-}_{\tilde{x},k} - L_k \Sigma_{\tilde{y},k} L_k^T.\end{aligned}$$

**KEY POINT:** The estimator output comprises the state estimate $\hat{x}^{+}_k$ and error covariance estimate $\Sigma^{+}_{\tilde{x},k}$.

- That is, we have high confidence that the truth lies within $\hat{x}^{+}_k \pm 3\sqrt{\text{diag}\left(\Sigma^{+}_{\tilde{x},k}\right)}$.
- The estimator then waits until the next sample interval, updates $k$, and proceeds to step 1a.

Step 1a: State estimate time update

$$\hat{x}^{-}_k = \mathbb{E}\big[x_k \mid \mathbb{Y}_{k-1}\big] = \mathbb{E}\big[f(x_{k-1}, u_{k-1}, w_{k-1}) \mid \mathbb{Y}_{k-1}\big].$$

Step 1b: Error covariance time update

$$\Sigma^{-}_{\tilde{x},k} = \mathbb{E}\big[(\tilde{x}^{-}_k)(\tilde{x}^{-}_k)^T\big] = \mathbb{E}\big[(x_k - \hat{x}^{-}_k)(x_k - \hat{x}^{-}_k)^T\big].$$

Step 1c: Estimate system output

$$\hat{y}_k = \mathbb{E}\big[y_k \mid \mathbb{Y}_{k-1}\big] = \mathbb{E}\big[h(x_k, u_k, v_k) \mid \mathbb{Y}_{k-1}\big].$$

Prediction

Step 2a: Estimator gain matrix

$$L_k = \Sigma^{-}_{\tilde{x}\tilde{y},k} \Sigma^{-1}_{\tilde{y},k}.$$

Step 2b: State estimate measurement update

$$\hat{x}^{+}_k = \hat{x}^{-}_k + L_k(y_k - \hat{y}_k).$$

Step 2c: Covariance estimate measurement update

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

Correction

## *Optimal application to linear systems: The Kalman filter*

- In the next section, we take the general solution and apply it to the specific case where the system dynamics are linear.
- Linear systems have the desirable property that all pdfs do in fact remain Gaussian if the stochastic inputs are Gaussian, so the assumptions made in deriving the filter steps hold exactly.
- The linear Kalman filter assumes that the system being modeled can be represented in the "state-space" form

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

$$y_k = C_k x_k + D_k u_k + v_k.$$

- We assume that $w_k$ and $v_k$ are mutually uncorrelated white Gaussian random processes, with zero mean and covariance matrices with known value:

$$\mathbb{E}[w_n w_k^T] = \begin{cases} \Sigma_{\widetilde{w}}, & n = k; \\ 0, & n \neq k. \end{cases} \qquad \mathbb{E}[v_n v_k^T] = \begin{cases} \Sigma_{\tilde{v}}, & n = k; \\ 0, & n \neq k, \end{cases}$$

  and $\mathbb{E}[w_k x_0^T] = 0$ for all $k$.
- The assumptions on the noise processes $w_k$ and $v_k$ and on the linearity of system dynamics are rarely (never) met in practice, but the consensus of the literature and practice is that the method still works very well.

## 3.7: Deriving the linear Kalman filter

- We now apply the general solution to the linear case, and derive the linear Kalman filter.
- An attempt to aid intuition is also given as we proceed.

**KF step 1a:** State estimate time update.

- Here, we compute the predicted state

$$\begin{aligned}\hat{x}_k^- &= \mathbb{E}\Big[f(x_{k-1}, u_{k-1}, w_{k-1}) \mid \mathbb{Y}_{k-1}\Big] \\ &= \mathbb{E}\Big[A_{k-1}x_{k-1} + B_{k-1}u_{k-1} + w_{k-1} \mid \mathbb{Y}_{k-1}\Big] \\ &= \mathbb{E}\Big[A_{k-1}x_{k-1} \mid \mathbb{Y}_{k-1}\Big] + \mathbb{E}\Big[B_{k-1}u_{k-1} \mid \mathbb{Y}_{k-1}\Big] + \mathbb{E}\Big[w_{k-1} \mid \mathbb{Y}_{k-1}\Big] \\ &= A_{k-1}\hat{x}_{k-1}^+ + B_{k-1}u_{k-1},\end{aligned}$$

  by the linearity of expectation, noting that $w_{k-1}$ is zero-mean.

  **INTUITION:** When predicting the present state given only past measurements, the best we can do is to use the most recent state estimate and system model, propagating the mean forward in time.

**KF step 1b:** Error covariance time update.

- First, we note that the prediction error is $\tilde{x}_k^- = x_k - \hat{x}_k^-$, so

$$\begin{aligned}\tilde{x}_k^- &= x_k - \hat{x}_k^- \\ &= A_{k-1}x_{k-1} + B_{k-1}u_{k-1} + w_{k-1} - A_{k-1}\hat{x}_{k-1}^+ - B_{k-1}u_{k-1} \\ &= A_{k-1}\tilde{x}_{k-1}^+ + w_{k-1}.\end{aligned}$$

- Therefore, the covariance of the prediction error is

$$\Sigma_{\tilde{x}_k}^- = \mathbb{E}\Big[(\tilde{x}_k^-)(\tilde{x}_k^-)^T\Big]$$

$$
\begin{aligned}
&= \mathbb{E}\Big[(A_{k-1}\tilde{x}^+_{k-1} + w_{k-1})(A_{k-1}\tilde{x}^+_{k-1} + w_{k-1})^T\Big] \\
&= \mathbb{E}\Big[A_{k-1}\tilde{x}^+_{k-1}(\tilde{x}^+_{k-1})^T A^T_{k-1} + w_{k-1}(\tilde{x}^+_{k-1})^T A^T_{k-1} \\
&\qquad + A_{k-1}\tilde{x}^+_{k-1} w^T_{k-1} + w_{k-1} w^T_{k-1}\Big] \\
&= A_{k-1}\Sigma^+_{\tilde{x},k-1} A^T_{k-1} + \Sigma_{\widetilde{w}}.
\end{aligned}
$$

- The cross terms drop out of the final result since the white process noise $w_{k-1}$ is not correlated with the state at time $k-1$.

  **INTUITION:** When estimating the error covariance of state prediction,

  - The best we can do is to use the most recent covariance estimate and propagate it forward in time.
  - For stable systems, $A_{k-1}\Sigma^+_{\tilde{x},k-1}A^T_{k-1}$ is contractive, meaning that the covariance gets "smaller." The state of stable systems always decays toward zero in the absence of input, or toward a known trajectory if $u_k \neq 0$. As time goes on, this term tells us that we tend to get more and more certain of the state estimate.
  - On the other hand, $\Sigma_{\widetilde{w}}$ adds to the covariance. Unmeasured inputs add uncertainty to our estimate because they perturb the trajectory away from the known trajectory based only on $u_k$.

**KF step 1c:** Predict system output.

- We predict the system output as

$$
\begin{aligned}
\hat{y}_k &= \mathbb{E}\Big[h(x_k, u_k, v_k) \mid \mathbb{Y}_{k-1}\Big] \\
&= \mathbb{E}\Big[C_k x_k + D_k u_k + v_k \mid \mathbb{Y}_{k-1}\Big] \\
&= \mathbb{E}\Big[C_k x_k \mid \mathbb{Y}_{k-1}\Big] + \mathbb{E}\Big[D_k u_k \mid \mathbb{Y}_{k-1}\Big] + \mathbb{E}\Big[v_k \mid \mathbb{Y}_{k-1}\Big] \\
&= C_k \hat{x}^-_k + D_k u_k,
\end{aligned}
$$

since $v_k$ is zero-mean.

**INTUITION:** $\hat{y}_k$ is our best guess of the system output, given only past measurements.

- The best we can do is to predict the output given the output equation of the system model, and our best guess of the system state at the present time.

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

- To compute the Kalman gain matrix, we first need to compute several covariance matrices: $L_k = \Sigma^-_{\tilde{x}\tilde{y},k}\Sigma^{-1}_{\tilde{y},k}$. We first find $\Sigma_{\tilde{y},k}$.

$$\begin{aligned}
\tilde{y}_k &= y_k - \hat{y}_k \\
&= C_k x_k + D_k u_k + v_k - C_k \hat{x}^-_k - D_k u_k \\
&= C_k \tilde{x}^-_k + v_k \\
\Sigma_{\tilde{y},k} &= \mathbb{E}\big[(C_k \tilde{x}^-_k + v_k)(C_k \tilde{x}^-_k + v_k)^T\big] \\
&= \mathbb{E}\big[C_k \tilde{x}^-_k (\tilde{x}^-_k)^T C_k^T + v_k (\tilde{x}^-_k)^T C_k^T + C_k \tilde{x}^-_k v_k^T + v_k v_k^T\big] \\
&= C_k \Sigma^-_{\tilde{x},k} C_k^T + \Sigma_{\tilde{v}}.
\end{aligned}$$

- Again, the cross terms are zero since $v_k$ is uncorrelated with $\tilde{x}^-_k$.
- Similarly,

$$\begin{aligned}
\mathbb{E}[\tilde{x}^-_k \tilde{y}_k^T] &= \mathbb{E}\big[\tilde{x}^-_k (C_k \tilde{x}^-_k + v_k)^T\big] \\
&= \mathbb{E}\big[\tilde{x}^-_k (\tilde{x}^-_k)^T C_k^T + \tilde{x}^-_k v_k^T\big] \\
&= \Sigma^-_{\tilde{x},k} C_k^T.
\end{aligned}$$

- Combining,

$$L_k = \Sigma^-_{\tilde{x},k} C_k^T [C_k \Sigma^-_{\tilde{x},k} C_k^T + \Sigma_{\tilde{v}}]^{-1}.$$

**INTUITION:** Note that the computation of $L_k$ is the most critical aspect of Kalman filtering that distinguishes it from a number of other estimation methods.

- The whole reason for calculating covariance matrices is to be able to update $L_k$.
- $L_k$ is time-varying. It adapts to give the best update to the state estimate based on present conditions.
- Recall that we use $L_k$ in the equation $\hat{x}_k^+ = \hat{x}_k^- + L_k(y_k - \hat{y}_k)$.
- The first component to $L_k$, $\Sigma_{\tilde{x}\tilde{y},k}^-$, indicates relative need for correction to $\hat{x}_k$ and how well individual states within $\hat{x}_k$ are coupled to the measurements.
- We see this clearly in $\Sigma_{\tilde{x}\tilde{y},k}^- = \Sigma_{\tilde{x},k}^- C_k^T$.
- $\Sigma_{\tilde{x},k}^-$ tells us about state uncertainty at the present time, which we hope to reduce as much as possible.
  - A large entry in $\Sigma_{\tilde{x},k}^-$ means that the corresponding state is very uncertain and therefore would benefit from a large update.
  - A small entry in $\Sigma_{\tilde{x},k}^-$ means that the corresponding state is very well known already and does not need as large an update.
- The $C_k^T$ term gives the coupling between state and output.
  - Entries that are zero indicate that a particular state has no direct influence on a particular output and therefore an output prediction error should not directly update that state.
  - Entries that are large indicate that a particular state is highly coupled to an output so has a large contribution to any measured output prediction error; therefore, that state would benefit from a large update.

- $\Sigma_{\tilde{y}}$ tells us how certain we are that the measurement is reliable.
  - If $\Sigma_{\tilde{y}}$ is "large," we want small, slow updates.
  - If $\Sigma_{\tilde{y}}$ is "small," we want big updates.
  - This explains why we divide the Kalman gain matrix by $\Sigma_{\tilde{y}}$.
- The form of $\Sigma_{\tilde{y}}$ can also be explained.
  - The $C_k \Sigma_{\tilde{x}}^{-} C_k^T$ part indicates how error in the state contributes to error in the output estimate.
  - The $\Sigma_{\tilde{v}}$ term indicates the uncertainty in the sensor reading due to sensor noise.
  - Since sensor noise is assumed independent of the state, the uncertainty in $\tilde{y}_k = y_k - \hat{y}_k$ adds the uncertainty in $y_k$ to the uncertainty in $\hat{y}_k$.

**KF step 2b:** State estimate measurement update.

- This step computes the *a posteriori* state estimate by updating the *a priori* estimate using the estimator gain and the output prediction error $y_k - \hat{y}_k$

$$\hat{x}_k^{+} = \hat{x}_k^{-} + L_k(y_k - \hat{y}_k).$$

**INTUITION:** The variable $\hat{y}_k$ is what we expect the measurement to be, based on our state estimate at the moment.

- Therefore, $y_k - \hat{y}_k$ is what is unexpected or new in the measurement.
- We call this term the innovation. The innovation can be due to a bad system model, state error, or sensor noise.
- So, we want to use this new information to update the state, but must be careful to weight it according to the value of the information it contains.

- $L_k$ is the optimal blending factor, as we have already discussed.

**KF step 2c:** Error covariance measurement update.

- Finally, we update the error covariance matrix.

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

**INTUITION:** The measurement update has decreased our uncertainty in the state estimate.

- A covariance matrix is positive semi-definite, and $L_k \Sigma_{\tilde{y},k} L_k^T$ is also a positive-semi-definite form, and we are subtracting this from the predicted-state covariance matrix; therefore, the resulting covariance is "lower" than the pre-measurement covariance.

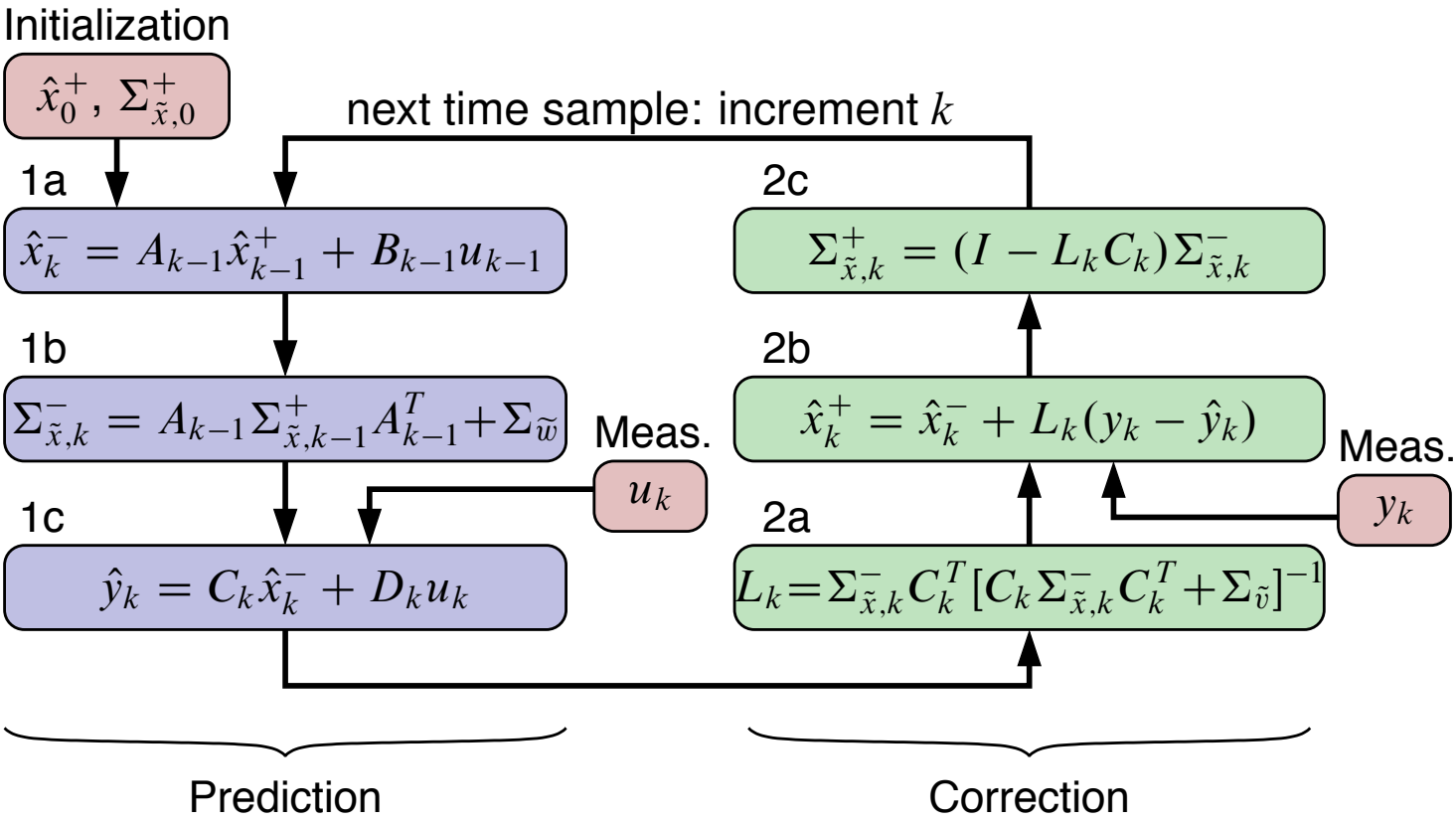
**COMMENT:** If a measurement is missed for some reason, then skip steps 2a–c for that iteration. That is, set $L_k = 0$ and $\hat{x}^+_k = \hat{x}^-_k$ and $\Sigma^+_{\tilde{x},k} = \Sigma^-_{\tilde{x},k}$.

**KEY POINT:** Repeating from before, recall that the estimator output comprises the state estimate $\hat{x}^+_k$ and error covariance estimate $\Sigma^+_{\tilde{x},k}$.

- That is, we have high confidence that the truth lies within $\hat{x}^+_k \pm 3\sqrt{\text{diag}\left(\Sigma^+_{\tilde{x},k}\right)}$.

## 3.8: Visualizing the Kalman filter

- The Kalman filter equations naturally form the following recursion

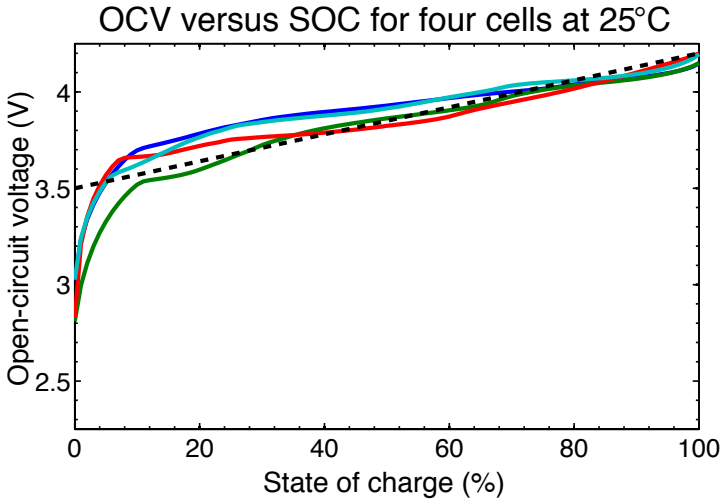



- "Simple" to perform on a digital computer. However, note that our cell models are nonlinear, so we cannot apply the (linear) Kalman filter to them directly.

- To demonstrate the KF steps, we'll develop and use a crude cell model

$$z_{k+1} = 1 \cdot z_k - \frac{1}{3600 \cdot Q} i_k$$

$$\text{volt}_k = 3.5 + 0.7 \times z_k - R_0 i_k.$$



- Notice that we have linearized the OCV relationship, and omitted the diffusion and hysteresis voltages.
- This model still isn't linear because of the "3.5" in the output equation, so we debias the measurement via $y_k = \text{volt}_k - 3.5$ and use the model

$$z_{k+1} = 1 \cdot z_k - \frac{1}{3600 \cdot Q} i_k$$

$$y_k = 0.7 \times z_k - R_0 i_k.$$

- Define state $x_k \equiv z_k$ and input $u_k \equiv i_k$.
- For the sake of example, we will use $Q = 10000/3600$ and $R_0 = 0.01$.
- This yields a state-space description with $A = 1$, $B = -1 \times 10^{-4}$, $C = 0.7$, and $D = -0.01$. We also model $\Sigma_{\widetilde{w}} = 10^{-5}$, and $\Sigma_{\tilde{v}} = 0.1$.
- We assume no initial uncertainty so $\hat{x}_0^+ = 0.5$ and $\Sigma_{\tilde{x},0}^+ = 0$.

*Iteration 1: (let $i_0 = 1$, $i_1 = 0.5$ and $v_1 = 3.85$)*

| | |
|---|---|
| $\hat{x}_k^- = A_{k-1}\hat{x}_{k-1}^+ + B_{k-1}u_{k-1}$ | $\hat{x}_1^- = 1 \times 0.5 - 10^{-4} \times 1 = 0.4999$ |
| $\Sigma_{\tilde{x},k}^- = A_{k-1}\Sigma_{\tilde{x},k-1}^+ A_{k-1}^T + \Sigma_{\widetilde{w}}$ | $\Sigma_{\tilde{x},1}^- = 1 \times 0 \times 1 + 10^{-5} = 10^{-5}$ |
| $\hat{y}_k = C_k\hat{x}_k^- + D_k u_k$ | $\hat{y}_1 = 0.7 \times 0.4999 - 0.01 \times 0.5 = 0.34493$ |
| $L_k = \Sigma_{\tilde{x},k}^- C_k^T [C_k \Sigma_{\tilde{x},k}^- C_k^T + \Sigma_{\tilde{v}}]^{-1}$ | $L_1 = 10^{-5} \times 0.7[0.7^2 \times 10^{-5} + 0.1]^{-1}$ $= 6.99966 \times 10^{-5}$ |
| $\hat{x}_k^+ = \hat{x}_k^- + L_k(y_k - \hat{y}_k)$ (where $y_k = 3.85 - 3.5$) | $\hat{x}_1^+ = 0.4999 + 6.99966 \times 10^{-5}(0.35 - 0.34493)$ $= 0.4999004$ |
| $\Sigma_{\tilde{x},k}^+ = (I - L_k C_k)\Sigma_{\tilde{x},k}^-$ | $\Sigma_{\tilde{x},1}^+ = (1 - 6.99966 \times 10^{-5} \times 0.7) \times 10^{-5}$ $= 9.9995 \times 10^{-6}$ |

- Output: $\hat{q} = 0.4999 \pm 3\sqrt{9.9995 \times 10^{-6}} = 0.4999 \pm 0.0094866$.

*Iteration 2: (let $i_1 = 0.5$, $i_2 = 0.25$, and $v_2 = 3.84$)*

$\hat{x}_k^- = A_{k-1}\hat{x}_{k-1}^+ + B_{k-1}u_{k-1}$ $\quad \hat{x}_1^- = 0.4999004 - 10^{-4} \times 0.5 = 0.49985$

$\Sigma_{\tilde{x},k}^- = A_{k-1}\Sigma_{\tilde{x},k-1}^+ A_{k-1}^T + \Sigma_{\tilde{w}}$ $\quad \Sigma_{\tilde{x},2}^- = 9.9995\times10^{-6} + 10^{-5} = 1.99995\times10^{-5}$

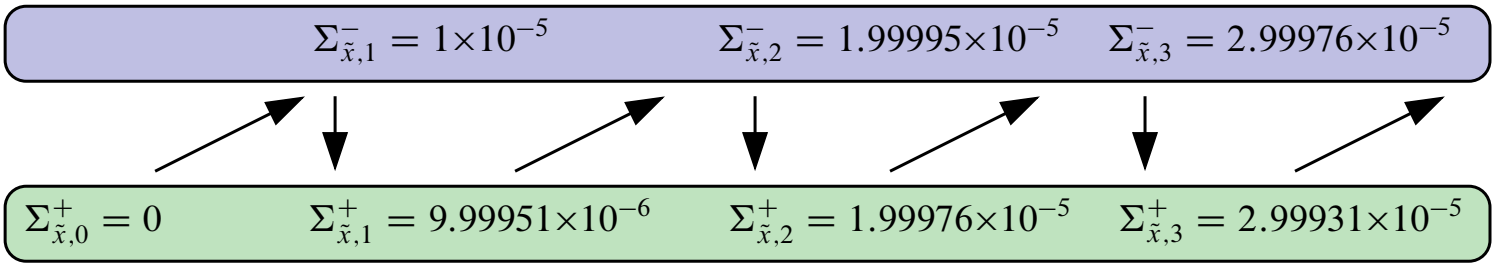
$\hat{y}_k = C_k\hat{x}_k^- + D_k u_k$ $\quad \hat{y}_2 = 0.7\times0.49985 - 0.01 \times 0.25 = 0.347395$

$L_k = \Sigma_{\tilde{x},k}^- C_k^T [C_k \Sigma_{\tilde{x},k}^- C_k^T + \Sigma_{\tilde{v}}]^{-1}$ $\quad L_2 = 1.99995\times10^{-5} \times 0.7[1.99995\times10^{-5} \times 0.7^2 + 0.1]^{-1}$
$= 0.00013998$

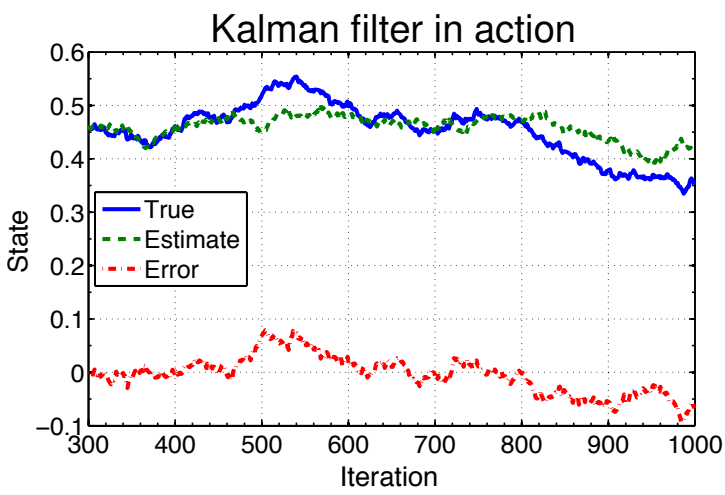
$\hat{x}_k^+ = \hat{x}_k^- + L_k(y_k - \hat{y}_k)$ $\quad \hat{x}_2^+ = 0.49985 + 0.00013998(0.34 - 0.347395)$
(where $y_k = 3.84 - 3.5$) $\quad = 0.499849$

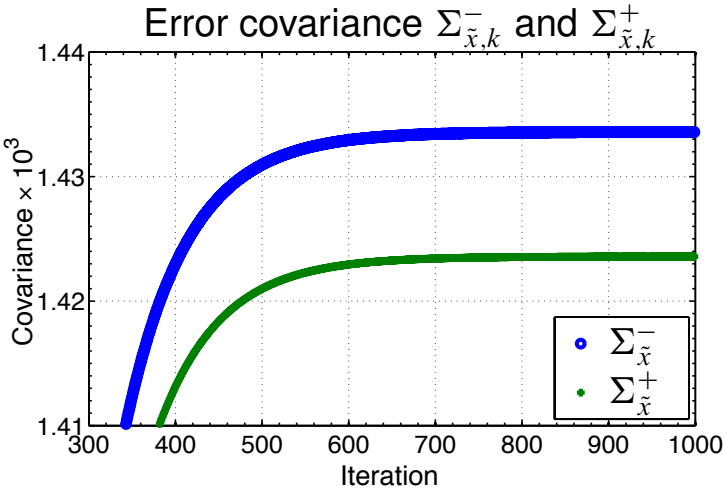
$\Sigma_{\tilde{x},k}^+ = (I - L_k C_k)\Sigma_{\tilde{x},k}^-$ $\quad \Sigma_{\tilde{x},2}^+ = (1 - 0.00013998 \times 0.7) \times 1.99995\times10^{-5}$
$= 1.99976\times10^{-5}$

- Output: $\hat{q} = 0.4998 \pm 3\sqrt{1.99976 \times 10^{-5}} = 0.4998 \pm 0.013416.$
- Note that covariance (uncertainty) converges, but it can take time




- Covariance increases during propagation and is then reduced by each measurement.
- Covariance still oscillates at steady state between $\Sigma_{\tilde{x},ss}^-$ and $\Sigma_{\tilde{x},ss}^+$.
- Estimation error bounds are $\pm 3\sqrt{\Sigma_{\tilde{x},k}^+}$ for $99^+$ % assurance of accuracy of estimate.
- The plots below show a sample of the Kalman filter operating. We shall look at how to write code to evaluate this example shortly.




- Note that Kalman filter does not perform especially well since $\Sigma_{\tilde{v}}$ is quite large.
- However, these are the best-possible results, since the KF is the optimum MMSE estimator.

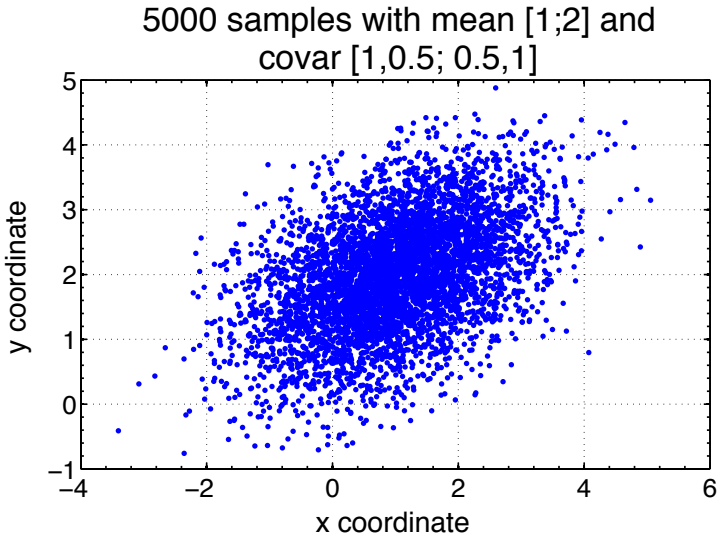
## 3.9: Matlab code for the Kalman filter steps

- It is straightforward to convert the Kalman filter steps to MATLAB. However, great care must be taken to ensure that all "$k$" and "$k+1$" indices (etc) are kept synchronized.
- In the example, below, we simulate the true system and implement a Kalman filter on its input–output data.
- To simulate the true system, we must be able to create non-zero mean Gaussian noise with covariance $\Sigma_{\widetilde{Y}}$. How to do so?
- That is, we want $Y \sim \mathcal{N}(\bar{y}, \Sigma_{\widetilde{Y}})$ but `randn.m` returns $X \sim \mathcal{N}(0, I)$.
- Try $y = \bar{y} + A^T x$ where $A$ is square with the same dimension as $\Sigma_{\widetilde{Y}}$; $A^T A = \Sigma_{\widetilde{Y}}$. ($A$ is the Cholesky decomposition of positive-definite symmetric matrix $\Sigma_{\widetilde{Y}}$).

```
ybar = [1; 2];
covar = [1, 0.5; 0.5, 1];
A = chol(covar);
x = randn([2, 1]);
y = ybar + A'*x;
```

- When $\Sigma_{\widetilde{Y}}$ is non-positive definite (but also non-negative definite)

```
[L,D] = ldl(covar);
x = randn([2,5000]);
y = ybar(:,ones([1 5000]))
  +(L*sqrt(D))*x;
```




- The code below is in MATLAB. Coding a KF in another language is no more challenging, except that you will need to write (or find) code to do the matrix manipulations.

```
% Initialize simulation variables
SigmaW = 1; % Process noise covariance
```

```
SigmaV = 1; % Sensor noise covariance
A = 1; B = 1; C = 1; D = 0; % Plant definition matrices
maxIter = 40;

xtrue = 0;  % Initialize true system initial state
xhat = 0;   % Initialize Kalman filter initial estimate
SigmaX = 0; % Initialize Kalman filter covariance
u = 0;      % Unknown initial driving input: assume zero

% Reserve storage for variables we  want to plot/evaluate
xstore = zeros(length(xtrue),maxIter+1); xstore(:,1) = xtrue;
xhatstore = zeros(length(xhat),maxIter);
SigmaXstore = zeros(length(xhat)^2,maxIter);

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

  % KF Step 1b: Error covariance time update
  SigmaX = A*SigmaX*A' + SigmaW;

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

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

  % KF Step 2a: Compute Kalman gain matrix
  SigmaY = C*SigmaX*C' + SigmaV;
  L = SigmaX*C'/SigmaY;

  % KF Step 2b: State estimate measurement update
  xhat = xhat + L*(ytrue - yhat);

  % KF Step 2c: Error covariance measurement update
  SigmaX = SigmaX - L*SigmaY*L';
```

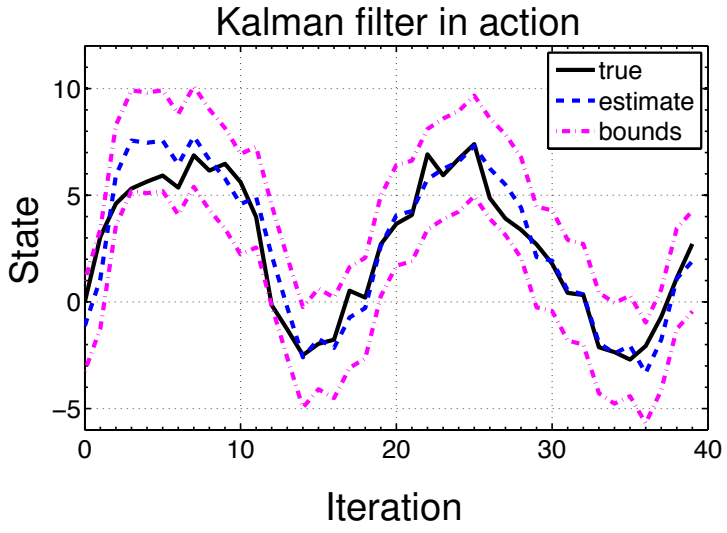
```
  % [Store information for evaluation/plotting purposes]
  xstore(:,k+1) = xtrue; xhatstore(:,k) = xhat;
  SigmaXstore(:,k) = SigmaX(:);
end

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;
legend('true','estimate','bounds');
title('Kalman filter in action');
xlabel('Iteration'); ylabel('State');

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');
```

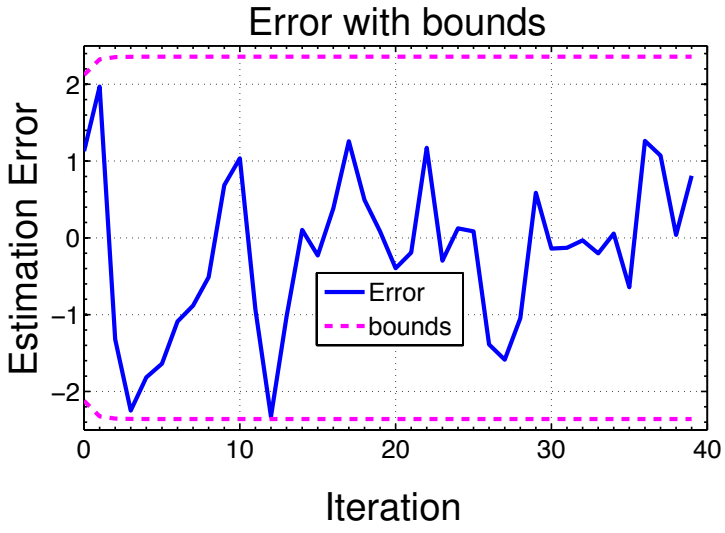
- The plots below show a sample of the Kalman filter operating for an example where

$$x_k = x_{k-1} + u_k + w_k$$

$$y_k = x_k + v_k$$

and $\Sigma_{\tilde{w}} = \Sigma_{\tilde{v}} = 1$.



## 3.10: Practical considerations

*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 un-symmetric 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:

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

  - 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.

- 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

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

- Because the subtraction occurs in the "squared" term, this form guarantees a positive definite result.
- If we still 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 $\Sigma$ with $(\Sigma + \Sigma^T + H + H^T)/4$.
- However, 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.
  - ECE5550 goes into more detail, incl. "square-root" Kalman filters.

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

## *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 $\Sigma_{\tilde{y},k} = C_k \Sigma^-_{\tilde{x},k} C_k^T + \Sigma_{\tilde{v}}$;
  - The measurement prediction itself is $\hat{y}_k = C_k \hat{x}_k^- + D_k u_k$;
  - The innovation is $\tilde{y}_k = y_k - \hat{y}_k$.
- A measurement validation gate can be set up around the measurement using normalized estimation error squared (NEES)

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

- NEES $e_k^2$ varies as a Chi-squared distribution with $m$ degrees of freedom, where $m$ is the dimension of $y_k$.
- If $e_k^2$ is outside of the bounding values from the Chi-squared distribution for a desired confidence level, then the measurement is discarded. (See appendix for Chi-squared test.)
- 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 $\Sigma^{\pm}_{\tilde{x},k}$, which simulates additional process noise, to help the Kalman filter to reacquire.

## *Nonlinear Kalman filters*

- Our cell models are nonlinear, so the standard Kalman-filter recursion does not apply directly.
- We now generalize to the nonlinear case, with system dynamics described as

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1})$$

$$y_k = h(x_k, u_k, v_k),$$

  where $u_k$ is a known (deterministic/measured) input signal, $w_k$ is a process-noise random input, and $v_k$ is a sensor-noise random input.
- We note that $f(\cdot)$ and $h(\cdot)$ may be time-varying, but we generally omit the time dependency from the notation for ease of understanding.
- There are three basic generalizations to KF to estimate the state of a nonlinear system
  - Extended Kalman filter (EKF): Analytic linearization of the model at each point in time. Problematic, but still popular.
  - Sigma-point (Unscented) Kalman filter (SPKF/UKF): Statistical/ empirical linearization of the model at each point in time. Much better than EKF, at same computational complexity.
  - Particle filters: The most precise, but often thousands of times more computations required than either EKF/SPKF. Does not assume Gaussian distributions but approximates distributions via histograms and uses Monte-Carlo integration techniques to find probabilities, expectations, and uncertainties.
- In this section, we present the EKF and SPKF. Particle filters are beyond the scope of this course.

## 3.11: The extended Kalman filter (EKF)

- The EKF makes two simplifying assumptions when adapting the general sequential inference equations to a nonlinear system:
  - When computing estimates of the output of a nonlinear function, EKF assumes $\mathbb{E}[\text{fn}(x)] \approx \text{fn}(\mathbb{E}[x])$, which is not true in general;
  - When computing covariance estimates, EKF uses Taylor-series expansion to linearize the system equations around the present operating point.
- Here, we will show how to apply these approximations and assumptions to derive the EKF equations from the general six steps.

**EKF step 1a:** State prediction time update.

- The state prediction step is approximated as

$$\begin{aligned}\hat{x}_k^- &= \mathbb{E}[f(x_{k-1}, u_{k-1}, w_{k-1}) \mid \mathbb{Y}_{k-1}] \\ &\approx f(\hat{x}_{k-1}^+, u_{k-1}, \bar{w}_{k-1}),\end{aligned}$$

  where $\bar{w}_{k-1} = \mathbb{E}[w_{k-1}]$. (Often, $\bar{w}_{k-1} = 0$.)
- That is, we approximate the expected value of the new state by assuming that it is reasonable to simply propagate $\hat{x}_{k-1}^+$ and $\bar{w}_{k-1}$ through the state equation.

**EKF step 1b:** Error covariance time update.

- The covariance prediction step is accomplished by first making an approximation for $\tilde{x}_k^-$.

$$\begin{aligned}\tilde{x}_k^- &= x_k - \hat{x}_k^- \\ &= f(x_{k-1}, u_{k-1}, w_{k-1}) - f(\hat{x}_{k-1}^+, u_{k-1}, \bar{w}_{k-1}).\end{aligned}$$

- The first term is expanded as a Taylor series around the prior operating "point" which is the set of values $\{\hat{x}_{k-1}^{+}, u_{k-1}, \bar{w}_{k-1}\}$

$$x_k \approx f(\hat{x}_{k-1}^{+}, u_{k-1}, \bar{w}_{k-1}) + \underbrace{\left.\frac{\mathrm{d}f(x_{k-1}, u_{k-1}, w_{k-1})}{\mathrm{d}x_{k-1}}\right|_{x_{k-1}=\hat{x}_{k-1}^{+}}}_{\text{Defined as } \hat{A}_{k-1}} (x_{k-1} - \hat{x}_{k-1}^{+})$$

$$+ \underbrace{\left.\frac{\mathrm{d}f(x_{k-1}, u_{k-1}, w_{k-1})}{\mathrm{d}w_{k-1}}\right|_{w_{k-1}=\bar{w}_{k-1}}}_{\text{Defined as } \hat{B}_{k-1}} (w_{k-1} - \bar{w}_{k-1}).$$

- This gives $\tilde{x}_k^{-} \approx \left(\hat{A}_{k-1}\tilde{x}_{k-1}^{+} + \hat{B}_{k-1}\widetilde{w}_{k-1}\right)$.
- Substituting this to find the prediction-error covariance:

$$\begin{aligned}\Sigma_{\tilde{x},k}^{-} &= \mathbb{E}[(\tilde{x}_k^{-})(\tilde{x}_k^{-})^T] \\ &\approx \hat{A}_{k-1}\Sigma_{\tilde{x},k-1}^{+}\hat{A}_{k-1}^T + \hat{B}_{k-1}\Sigma_{\widetilde{w}}\hat{B}_{k-1}^T.\end{aligned}$$

- Note, by the chain rule of total differentials,

$$\begin{aligned}\mathrm{d}f(x_{k-1}, u_{k-1}, w_{k-1}) = &\frac{\partial f(x_{k-1}, u_{k-1}, w_{k-1})}{\partial x_{k-1}}\mathrm{d}x_{k-1} + \\ &\frac{\partial f(x_{k-1}, u_{k-1}, w_{k-1})}{\partial u_{k-1}}\mathrm{d}u_{k-1} + \\ &\frac{\partial f(x_{k-1}, u_{k-1}, w_{k-1})}{\partial w_{k-1}}\mathrm{d}w_{k-1}\end{aligned}$$

$$\begin{aligned}\frac{\mathrm{d}f(x_{k-1}, u_{k-1}, w_{k-1})}{\mathrm{d}x_{k-1}} = &\frac{\partial f(x_{k-1}, u_{k-1}, w_{k-1})}{\partial x_{k-1}} + \\ &\frac{\partial f(x_{k-1}, u_{k-1}, w_{k-1})}{\partial u_{k-1}}\underbrace{\frac{\mathrm{d}u_{k-1}}{\mathrm{d}x_{k-1}}}_{0} +\end{aligned}$$

$$\frac{\partial f(x_{k-1}, u_{k-1}, w_{k-1})}{\partial w_{k-1}} \underbrace{\frac{\mathrm{d}w_{k-1}}{\mathrm{d}x_{k-1}}}_{0}$$

$$= \frac{\partial f(x_{k-1}, u_{k-1}, w_{k-1})}{\partial x_{k-1}}.$$

- Similarly,

$$\frac{\mathrm{d}f(x_{k-1}, u_{k-1}, w_{k-1})}{\mathrm{d}w_{k-1}} = \frac{\partial f(x_{k-1}, u_{k-1}, w_{k-1})}{\partial w_{k-1}}.$$

- The distinction between the total differential and the partial differential is not critical at this point, but will be in the next section of notes when we look at parameter estimation using extended Kalman filters.

**EKF step 1c:** Output estimate.

- The system output is estimated to be

$$\hat{y}_k = \mathbb{E}[h(x_k, u_k, v_k) \mid \mathbb{Y}_{k-1}]$$

$$\approx h(\hat{x}_k^-, u_k, \bar{v}_k),$$

where $\bar{v}_k = \mathbb{E}[v_k]$.

- That is, it is assumed that propagating $\hat{x}_k^-$ and the mean sensor noise is the best approximation to estimating the output.

**EKF step 2a:** Estimator gain matrix.

- The output prediction error may then be approximated

$$\tilde{y}_k = y_k - \hat{y}_k = h(x_k, u_k, v_k) - h(\hat{x}_k^-, u_k, \bar{v}_k)$$

using again a Taylor-series expansion on the first term.

$$y_k \approx h(\hat{x}_k^-, u_k, \bar{v}_k) + \underbrace{\left.\frac{\mathrm{d}h(x_k, u_k, v_k)}{\mathrm{d}x_k}\right|_{x_k=\hat{x}_k^-}}_{\text{Defined as } \hat{C}_k} (x_k - \hat{x}_k^-)$$

$$+ \underbrace{\left.\frac{\mathrm{d}h(x_k, u_k, v_k)}{\mathrm{d}v_k}\right|_{v_k=\bar{v}_k}}_{\text{Defined as } \hat{D}_k} (v_k - \bar{v}_k).$$

- Note, much like we saw in Step 1b,

$$\frac{\mathrm{d}h(x_k, u_k, v_k)}{\mathrm{d}x_k} = \frac{\partial h(x_k, u_k, v_k)}{\partial x_k}$$
$$\frac{\mathrm{d}h(x_k, u_k, v_k)}{\mathrm{d}v_k} = \frac{\partial h(x_k, u_k, v_k)}{\partial v_k}.$$

- From this, we can compute such necessary quantities as

$$\Sigma_{\tilde{y},k} \approx \hat{C}_k \Sigma^-_{\tilde{x},k} \hat{C}_k^T + \hat{D}_k \Sigma_{\tilde{v}} \hat{D}_k^T,$$
$$\Sigma^-_{\tilde{x}\tilde{y},k} \approx \mathbb{E}[(\tilde{x}^-_k)(\hat{C}_k \tilde{x}^-_k + \hat{D}_k \tilde{v}_k)^T]$$
$$= \Sigma^-_{\tilde{x},k} \hat{C}_k^T.$$

- These terms may be combined to get the Kalman gain

$$L_k = \Sigma^-_{\tilde{x},k} \hat{C}_k^T \left[\hat{C}_k \Sigma^-_{\tilde{x},k} \hat{C}_k^T + \hat{D}_k \Sigma_{\tilde{v}} \hat{D}_k^T\right]^{-1}.$$

**EKF step 2b:** State estimate measurement update.

- This step computes the posterior state estimate by updating the prediction using the estimator gain and the innovation $y_k - \hat{y}_k$

$$\hat{x}^+_k = \hat{x}^-_k + L_k(y_k - \hat{y}_k).$$

**EKF step 2c:** Error covariance measurement update.

- Finally, the updated covariance is computed as

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

## 3.12: An EKF example, with code

- Consider an example of EKF in action, with the following dynamics:

$$x_{k+1} = f(x_k, u_k, w_k) = \sqrt{5 + x_k} + w_k$$

$$y_k = h(x_k, u_k, v_k) = x_k^3 + v_k$$

with $\Sigma_{\tilde{w}} = 1$ and $\Sigma_{\tilde{v}} = 2$.

- To implement EKF, we must determine $\hat{A}_k$, $\hat{B}_k$, $\hat{C}_k$, and $\hat{D}_k$.

$$\hat{A}_k = \left.\frac{\partial f(x_k, u_k, w_k)}{\partial x_k}\right|_{x_k=\hat{x}_k^+} = \left.\frac{\partial\left(\sqrt{5 + x_k} + w_k\right)}{\partial x_k}\right|_{x_k=\hat{x}_k^+} = \frac{1}{2\sqrt{5 + \hat{x}_k^+}}$$

$$\hat{B}_k = \left.\frac{\partial f(x_k, u_k, w_k)}{\partial w_k}\right|_{w_k=\bar{w}_k} = \left.\frac{\partial\left(\sqrt{5 + x_k} + w_k\right)}{\partial w_k}\right|_{w_k=\bar{w}_k} = 1$$

$$\hat{C}_k = \left.\frac{\partial h(x_k, u_k, v_k)}{\partial x_k}\right|_{x_k=\hat{x}_k^-} = \left.\frac{\partial\left(x_k^3 + v_k\right)}{\partial x_k}\right|_{x_k=\hat{x}_k^-} = 3(\hat{x}_k^-)^2$$

$$\hat{D}_k = \left.\frac{\partial h(x_k, u_k, v_k)}{\partial v_k}\right|_{v_k=\bar{v}_k} = \left.\frac{\partial\left(x_k^3 + v_k\right)}{\partial v_k}\right|_{v_k=\bar{v}_k} = 1.$$

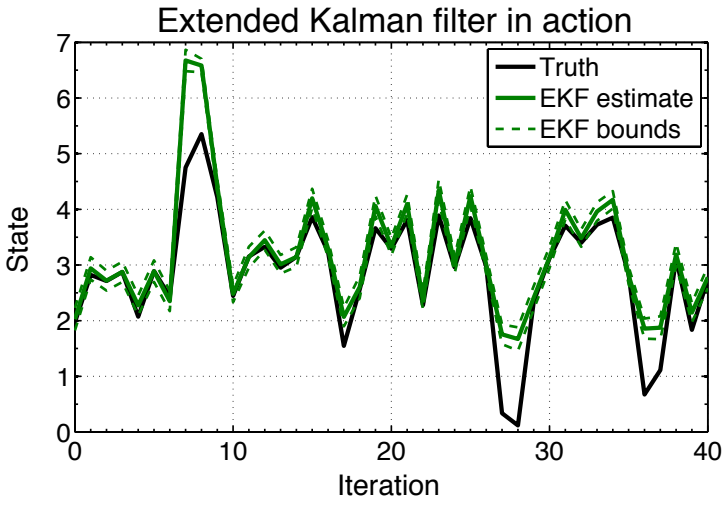
- The following is some sample code to implement an EKF.
  - Note that the steps for calculating the plant and the $\hat{A}$, $\hat{B}$, $\hat{C}$, and $\hat{D}$ matrices will depend on the nonlinear system underlying the estimation problem.

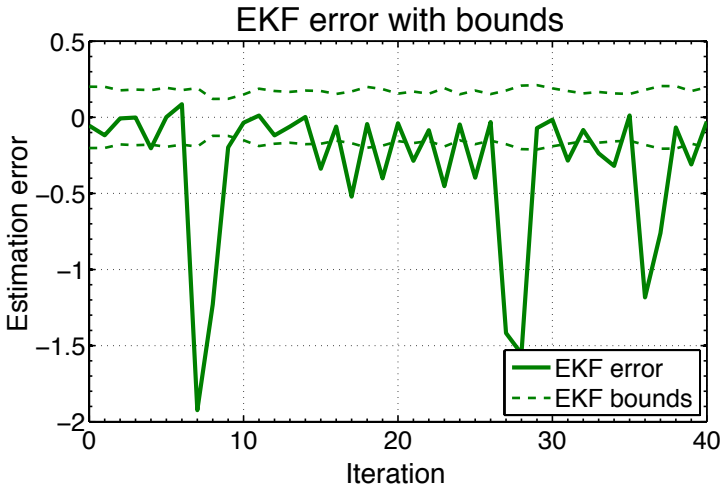
```
% Initialize simulation variables
SigmaW = 1; % Process noise covariance
SigmaV = 2; % Sensor noise covariance
maxIter = 40;

xtrue = 2 + randn(1); % Initialize true system initial state
```

```
xhat = 2;                % Initialize Kalman filter initial estimate
SigmaX = 1;              % Initialize Kalman filter covariance
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)^2);

for k = 1:maxIter,
  % EKF Step 0: Compute Ahat, Bhat
  % Note: For this example, x(k+1) = sqrt(5+x(k)) + w(k)
  Ahat = 0.5/sqrt(5+xhat); Bhat = 1;

  % EKF Step 1a: State estimate time update
  % Note: You need to insert your system's f(...) equation here
  xhat = sqrt(5+xhat);

  % EKF Step 1b: Error covariance time update
  SigmaX = Ahat*SigmaX*Ahat' + Bhat*SigmaW*Bhat';

  % [Implied operation of system in background, with
  % input signal u, and output signal y]
  w = chol(SigmaW)'*randn(1);
  v = chol(SigmaV)'*randn(1);
  ytrue = xtrue^3 + v;  % y is based on present x and u
  xtrue = sqrt(5+xtrue) + w;  % future x is based on present u

  % EKF Step 1c: Estimate system output
  % Note: You need to insert your system's h(...) equation here
  Chat = 3*xhat^2; Dhat = 1;
  yhat = xhat^3;

  % EKF Step 2a: Compute Kalman gain matrix
  SigmaY = Chat*SigmaX*Chat' + Dhat*SigmaV*Dhat';
  L = SigmaX*Chat'/SigmaY;

  % EKF Step 2b: State estimate measurement update
  xhat = xhat + L*(ytrue - yhat);
  xhat = max(-5,xhat); % don't get square root of negative xhat!

  % EKF Step 2c: Error covariance measurement update
```

```
  SigmaX = SigmaX - L*SigmaY*L';
  [~,S,V] = svd(SigmaX);
  HH = V*S*V';
  SigmaX = (SigmaX + SigmaX' + HH + HH')/4; % Help to keep robust

  % [Store information for evaluation/plotting purposes]
  xstore(k+1,:) = xtrue; xhatstore(k,:) = xhat;
  SigmaXstore(k,:) = (SigmaX(:))';
end;

figure(1); clf; t = 0:maxIter-1;
plot(t,xstore(1:maxIter),'k-',t,xhatstore,'b--', ...
  t,xhatstore+3*sqrt(SigmaXstore),'m-.',...
  t,xhatstore-3*sqrt(SigmaXstore),'m-.'); grid;
legend('true','estimate','bounds');
xlabel('Iteration'); ylabel('State');
title('Extended Kalman filter in action');

figure(2); clf;
plot(t,xstore(1:maxIter)-xhatstore,'b-',t, ...
  3*sqrt(SigmaXstore),'m--',t,-3*sqrt(SigmaXstore),'m--');
grid; legend('Error','bounds',0);
title('EKF Error with bounds');
xlabel('Iteration'); ylabel('Estimation error');
```



## 3.13: Preparing to implement EKF on ESC model

- To implement the EKF for battery-cell state estimation using the ESC model, we must know the $\hat{A}_k$, $\hat{B}_k$, $\hat{C}_k$ and $\hat{D}_k$ matrices. We first examine the components of the state equation to find $\hat{A}_k$ and $\hat{B}_k$.
- Suppose that the process noise models current-sensor measurement error. That is, the true cell current is $i_k + w_k$, but we measure $i_k$ only.
- Also assume we can simplify model with $\eta_k = 1$, and have adaptivity of EKF handle the small error introduced by this assumption.
- Then, the state-of-charge equation can be written as

$$z_{k+1} = z_k - \frac{\Delta t}{Q}\left(i_k + w_k\right).$$

- The two derivatives that we need for this term are:

$$\left.\frac{\partial z_{k+1}}{\partial z_k}\right|_{z_k=\hat{z}_k^+} = 1, \qquad \text{and} \qquad \left.\frac{\partial z_{k+1}}{\partial w_k}\right|_{w_k=\bar{w}} = -\frac{\Delta t}{Q},$$

  remembering that $Q$ is measured in ampere-seconds.

- If $\tau_j = \exp(-\Delta t/(R_j C_j))$, then the resistor-currents state equation can be written as

$$i_{R,k+1} = \underbrace{\begin{bmatrix} \tau_1 & 0 & \cdots \\ 0 & \tau_2 & \\ \vdots & & \ddots \end{bmatrix}}_{A_{RC}} i_{R,k} + \underbrace{\begin{bmatrix} 1-\tau_1 \\ 1-\tau_2 \\ \vdots \end{bmatrix}}_{B_{RC}} \left(i_k + w_k\right).$$

- The two derivatives can be found to be

$$\left.\frac{\partial i_{R,k+1}}{\partial i_{R,k}}\right|_{i_{R,k}=\hat{i}_{R,k}^+} = A_{RC}, \qquad \text{and} \qquad \left.\frac{\partial i_{R,k+1}}{\partial w_k}\right|_{w_k=\bar{w}} = B_{RC}.$$

- If we define $A_{H,k} = \exp\left(-\left|\dfrac{(i_k + w_k)\,\gamma\Delta t}{Q}\right|\right)$, then hysteresis state

$$h_{k+1} = A_{H,k} h_k + (A_{H,k} - 1)\,\mathrm{sgn}\,(i_k + w_k)\,.$$

- Taking partial with respect to the state and evaluating at the setpoint (noting that $w_k = \bar{w}$ is a member of the setpoint),

$$\left.\frac{\partial h_{k+1}}{\partial h_k}\right|_{\substack{h_k = \hat{h}_k^+ \\ w_k = \bar{w}}} = \exp\left(-\left|\frac{(i_k + \bar{w}_k)\,\gamma\Delta t}{Q}\right|\right) = \bar{A}_{H,k}.$$

- Next, we must find $\partial h_{k+1}/\partial w_k$. However, the absolute-value and sign functions are not differentiable at $i_k + w_k = 0$. Ignoring this detail,
  - If we assume that $i_k + w_k > 0$,

$$\frac{\partial h_{k+1}}{\partial w_k} = -\left|\frac{\gamma\Delta t}{Q}\right| \exp\left(-\left|\frac{\gamma\Delta t}{Q}\right| |(i_k + w_k)|\right)(1 + h_k)\,.$$

  - If we assume that $i_k + w_k < 0$,

$$\frac{\partial h_{k+1}}{\partial w_k} = -\left|\frac{\gamma\Delta t}{Q}\right| \exp\left(-\left|\frac{\gamma\Delta t}{Q}\right| |(i_k + w_k)|\right)(1 - h_k)\,.$$

- Overall, evaluating at Taylor-series linearization setpoint,

$$\left.\frac{\partial h_{k+1}}{\partial w_k}\right|_{\substack{h_k = \hat{h}_k^+ \\ w_k = \bar{w}}} = -\left|\frac{\gamma\Delta t}{Q}\right| \bar{A}_{H,k}\left(1 + \mathrm{sgn}(i_k + \bar{w})\hat{h}_k^+\right).$$

- The zero-state hysteresis equation is defined as

$$s_{k+1} = \begin{cases} \mathrm{sgn}(i_k + w_k), & |i_k + w_k| > 0, \\ s_k, & \text{else.} \end{cases}$$

- If we consider $i_k + w_k = 0$ to be a zero-probability event, then

$$\frac{\partial s_{k+1}}{\partial s_k} = 0, \qquad \text{and} \qquad \frac{\partial s_{k+1}}{\partial w_k} = 0.$$

- We now look at the components that determine $\hat{C}_k$ and $\hat{D}_k$.
- The ESC-model output equation is

$$y_k = \text{OCV}(z_k) + M_0 s_k + M h_k - \sum_j R_j i_{R_j,k} - R_0 i_k + v_k,$$

  no longer considering $i_k$ to have $w_k$ noise added to it (this would add correlation between process noise and sensor noise).
- We have

$$\left.\frac{\partial y_k}{\partial s_k}\right| = M_0, \quad \left.\frac{\partial y_k}{\partial h_k}\right| = M, \quad \left.\frac{\partial y_k}{\partial i_{R_j,k}}\right| = -R_j, \quad \left.\frac{\partial y_k}{\partial v_k}\right| = 1.$$

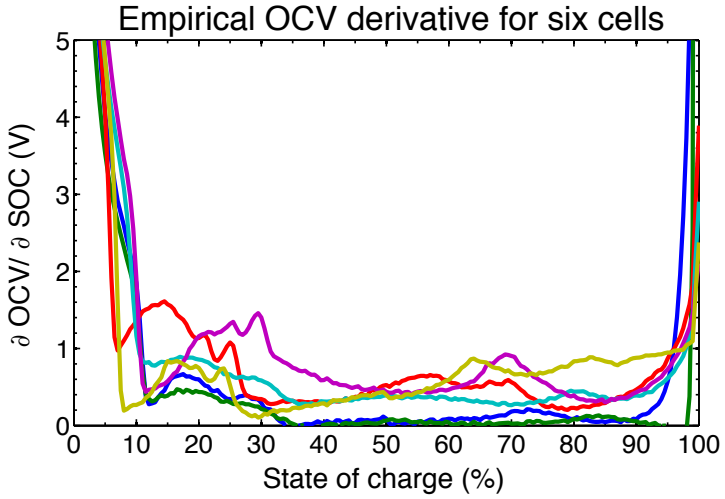
- We also require

$$\left.\frac{\partial y_k}{\partial z_k}\right|_{z_k=\hat{z}_k^-} = \left.\frac{\partial \text{OCV}(z_k)}{\partial z_k}\right|_{z_k=\hat{z}_k^-},$$

  which can be approximated from open-circuit-voltage data. If SOC is a vector of evenly-spaced SOC points with corresponding OCV,

```
% Find dOCV/dz at SOC = z from {SOC,OCV} data
function dOCVz = dOCVfromSOC(SOC,OCV,z)
  dZ = SOC(2) - SOC(1); % Find spacing of SOC vector
  dUdZ = diff(OCV)/dZ;  % Scaled forward finite difference
  dOCV = ([dUdZ(1) dUdZ] + [dUdZ dUdZ(end)])/2; % Avg of fwd/bkwd diffs
  dOCVz = interp1(SOC,dOCV,z); % Could make more efficient than this...
```

- The figure shows empirical OCV derivative relationships for six different lithium-ion cells.
- There is a little noise, which could be filtered (with a zero-phase filter!), but it's not really necessary to do so.



# 3.14: Implementing EKF on ESC model

- We refactor the code to better represent a real BMS implementation.
- There is an initialization routine (`initEKF.m`), called once at startup.
- An update routine (`iterEKF.m`) is called every sample interval.
- "Wrapper" code, which coordinates the entire simulation process, is:

```
load CellModel            % loads "model" of cell

% Load cell-test data. Contains variable "DYNData" of which the field
% "script1" is of interest. It has sub-fields time, current, voltage, soc.
load('Cell_DYN_P25');     % loads data from cell test
T = 25; % Test temperature

time    = DYNData.script1.time(:);   deltat = time(2)-time(1);
time    = time-time(1); % start time at 0
current = DYNData.script1.current(:); % discharge > 0; charge < 0.
voltage = DYNData.script1.voltage(:);
soc     = DYNData.script1.soc(:);

% Reserve storage for computed results, for plotting
sochat = zeros(size(soc));
socbound = zeros(size(soc));

% Covariance values
SigmaX0 = diag([1e-3 1e-3 1e-2]); % uncertainty of initial state
SigmaV = 2e-1; % uncertainty of voltage sensor, output equation
SigmaW = 1e1;  % uncertainty of current sensor, state equation

% Create ekfData structure and initialize variables using first
% voltage measurement and first temperature measurement
ekfData = initEKF(voltage(1),T,SigmaX0,SigmaV,SigmaW,model);

% Now, enter loop for remainder of time, where we update the EKF
% once per sample interval
hwait = waitbar(0,'Computing...');
for k = 1:length(voltage),
  vk = voltage(k); % "measure" voltage
  ik = current(k); % "measure" current
```

```
  Tk = T;           % "measure" temperature

  % Update SOC (and other model states)
  [sochat(k),socbound(k),ekfData] = iterEKF(vk,ik,Tk,deltat,ekfData);
  % update waitbar periodically, but not too often (slow procedure)
  if mod(k,1000)==0, waitbar(k/length(current),hwait); end;
end
close(hwait);

figure(1); clf; plot(time/60,100*sochat,time/60,100*soc); hold on
h = plot([time/60; NaN; time/60],...
         [100*(sochat+socbound); NaN; 100*(sochat-socbound)]);
title('SOC estimation using EKF');
xlabel('Time (min)'); ylabel('SOC (%)');
legend('Estimate','Truth','Bounds'); grid on

fprintf('RMS SOC estimation error = %g%%\n',...
        sqrt(mean((100*(soc-sochat)).^2)));

figure(2); clf; plot(time/60,100*(soc-sochat)); hold on
h = plot([time/60; NaN; time/60],[100*socbound; NaN; -100*socbound]);
title('SOC estimation errors using EKF');
xlabel('Time (min)'); ylabel('SOC error (%)'); ylim([-4 4]);
legend('Estimation error','Bounds'); grid on
print -deps2c EKF2.eps

ind = find(abs(soc-sochat)>socbound);
fprintf('Percent of time error outside bounds = %g%%\n',...
        length(ind)/length(soc)*100);
```

- The initialization code is:

```
function ekfData = initEKF(v0,T0,SigmaX0,SigmaV,SigmaW,model)

  % Initial state description
  ir0   = 0;                          ekfData.irInd = 1;
  hk0   = 0;                          ekfData.hkInd = 2;
  SOC0  = SOCfromOCVtemp(v0,T0,model); ekfData.zkInd = 3;
  ekfData.xhat  = [ir0 hk0 SOC0]'; % initial state

  % Covariance values
  ekfData.SigmaX = SigmaX0;
```

```
  ekfData.SigmaV = SigmaV;
  ekfData.SigmaW = SigmaW;
  ekfData.Qbump = 5;

  % previous value of current
  ekfData.priorI = 0;
  ekfData.signIk = 0;

  % store model data structure too
  ekfData.model = model;
end
```

- The iteration code is:

```
function [zk,zkbnd,ekfData] = iterEKF(vk,ik,Tk,deltat,ekfData)
  model = ekfData.model;
  % Load the cell model parameters
  Q  = getParamESC('QParam',Tk,model);
  G  = getParamESC('GParam',Tk,model);
  M  = getParamESC('MParam',Tk,model);
  M0 = getParamESC('M0Param',Tk,model);
  RC = exp(-deltat./abs(getParamESC('RCParam',Tk,model)))';
  R  = getParamESC('RParam',Tk,model)';
  R0 = getParamESC('R0Param',Tk,model);
  eta = getParamESC('etaParam',Tk,model);
  if ik<0, ik=ik*eta; end;

  % Get data stored in ekfData structure
  I = ekfData.priorI;
  SigmaX = ekfData.SigmaX;
  SigmaV = ekfData.SigmaV;
  SigmaW = ekfData.SigmaW;
  xhat = ekfData.xhat;
  irInd = ekfData.irInd;
  hkInd = ekfData.hkInd;
  zkInd = ekfData.zkInd;
  if abs(ik)>Q/100, ekfData.signIk = sign(ik); end;
  signIk = ekfData.signIk;

  % EKF Step 0: Compute Ahat[k-1], Bhat[k-1]
  nx = length(xhat); Ahat = zeros(nx,nx); Bhat = zeros(nx,1);
  Ahat(zkInd,zkInd) = 1; Bhat(zkInd) = -deltat/(3600*Q);
```

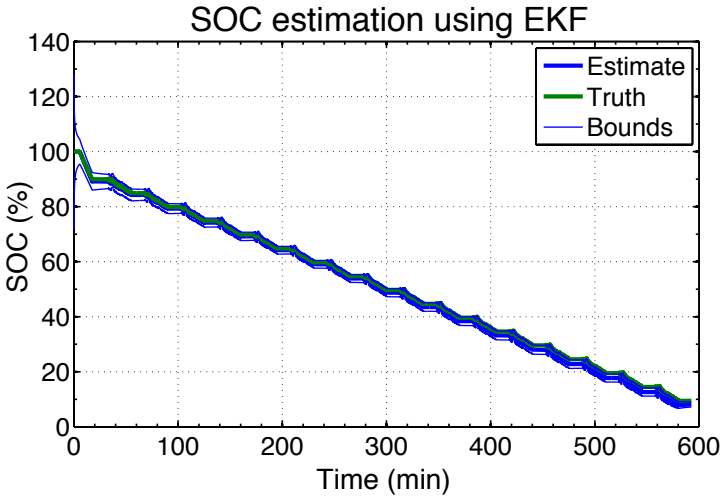
 

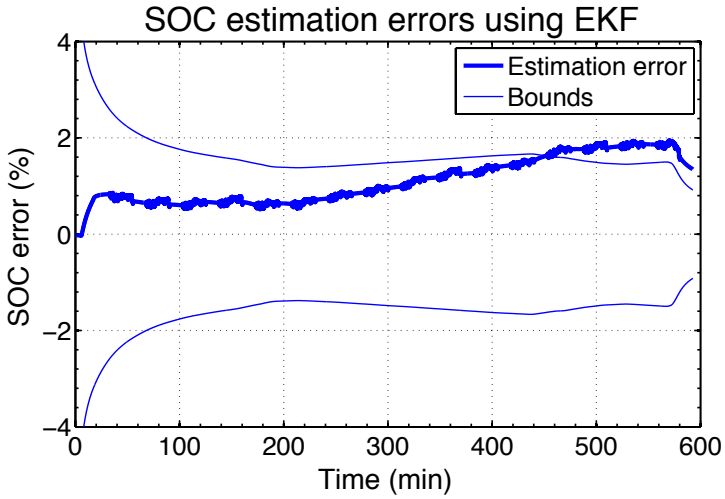
```
Ahat(irInd,irInd) = diag(RC); Bhat(irInd) = 1-RC(:);
Ah  = exp(-abs(I*G*deltat/(3600*Q)));  % hysteresis factor
Ahat(hkInd,hkInd) = Ah;
B = [Bhat, 0*Bhat];
Bhat(hkInd) = -abs(G*deltat/(3600*Q))*Ah*(1+sign(I)*xhat(hkInd));
B(hkInd,2) = 1-Ah;

% Step 1a: State estimate time update
xhat = Ahat*xhat + B*[I; sign(I)];

% Step 1b: Error covariance time update
%          sigmaminus(k) = Ahat(k-1)*sigmaplus(k-1)*Ahat(k-1)' + ...
%                          Bhat(k-1)*sigmawtilde*Bhat(k-1)'
SigmaX = Ahat*SigmaX*Ahat' + Bhat*SigmaW*Bhat';

% Step 1c: Output estimate
yhat = OCVfromSOCtemp(xhat(zkInd),Tk,model) + M0*signIk + ...
       M*xhat(hkInd) - R*xhat(irInd) - R0*ik;

% Step 2a: Estimator gain matrix
Chat = zeros(1,nx);
Chat(zkInd) = dOCVfromSOCtemp(xhat(zkInd),Tk,model);
Chat(hkInd) = M;
Chat(irInd) = -R;
Dhat = 1;
SigmaY = Chat*SigmaX*Chat' + Dhat*SigmaV*Dhat';
L = SigmaX*Chat'/SigmaY;

% Step 2b: State estimate measurement update
r = vk - yhat; % residual.  Use to check for sensor errors...
if r^2 > 100*SigmaY, L(:)=0.0; end
xhat = xhat + L*r;
xhat(hkInd) = min(1,max(-1,xhat(hkInd))); % Help maintain robustness
xhat(zkInd) = min(1.05,max(-0.05,xhat(zkInd)));

% Step 2c: Error covariance measurement update
SigmaX = SigmaX - L*SigmaY*L';
%   % Q-bump code
if r^2 > 4*SigmaY, % bad voltage estimate by 2 std. devs, bump Q
  fprintf('Bumping SigmaX\n');
  SigmaX(zkInd,zkInd) = SigmaX(zkInd,zkInd)*ekfData.Qbump;
end
```

```
  [~,S,V] = svd(SigmaX);
  HH = V*S*V';
  SigmaX = (SigmaX + SigmaX' + HH + HH')/4; % Help maintain robustness

  % Save data in ekfData structure for next time...
  ekfData.priorI = ik;
  ekfData.SigmaX = SigmaX;
  ekfData.xhat = xhat;
  zk = xhat(zkInd);
  zkbnd = 3*sqrt(SigmaX(zkInd,zkInd));
end
```

- For the following example, the EKF was executed for a test having dynamic profiles from 100 % SOC down to around 10 % SOC.
  - RMS SOC estimation error = 1.20 %
  - Percent of time error outside bounds = 23.79 %.



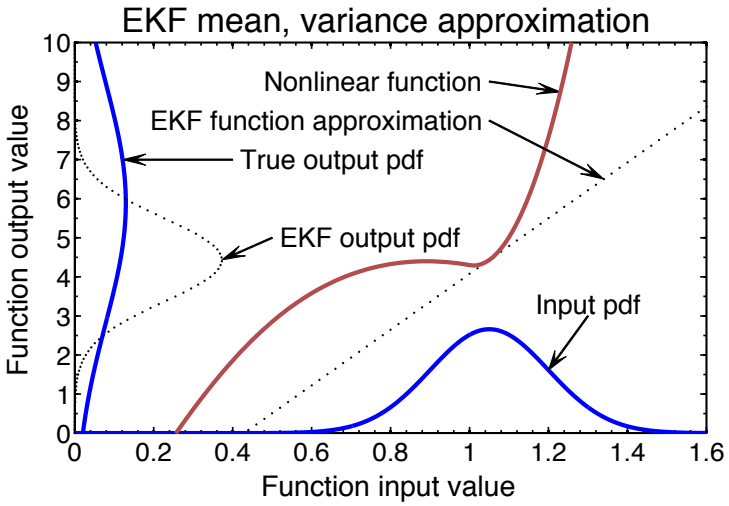

 

## 3.15: Problems with EKF, improved with sigma-point methods

- The EKF is the best known and most used nonlinear Kalman filter.
- However, it has serious flaws that can be remedied fairly easily.

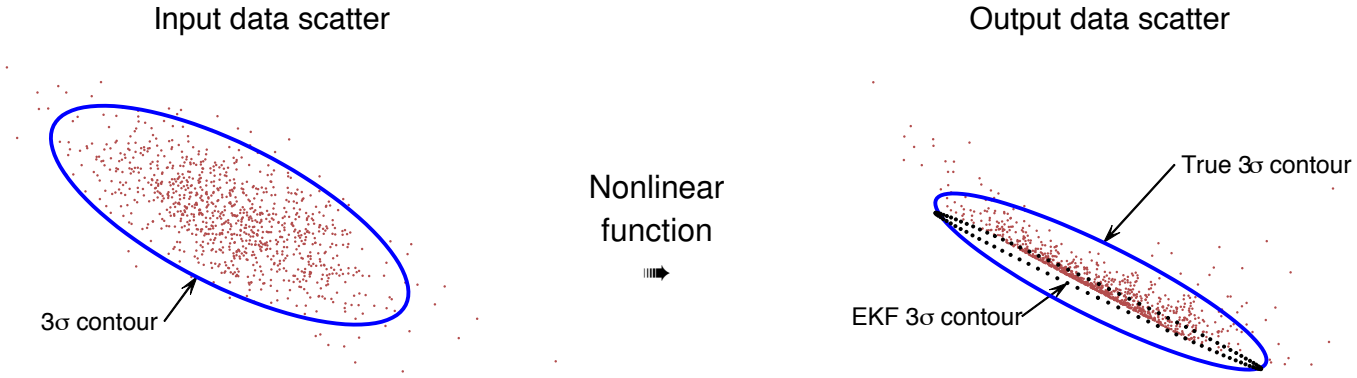
**ISSUE:** How input mean and covariance are propagated through static nonlinear function to create output mean and covariance estimates.

- Recall that the EKF, when computing mean estimates in Steps 1a and 1c, makes the simplification $\mathbb{E}[\text{fn}(x)] \approx \text{fn}(\mathbb{E}[x])$.
  - This is not true in general, and not necessarily even close to true (depending on "how nonlinear" the function $\text{fn}(\cdot)$ is).
- Also, in EKF Steps 1b and 2a, a Taylor-series expansion is performed as part of the calculation of output-variable covariance.
  - Nonlinear terms are dropped, resulting in a loss of accuracy.
- A simple one-dimensional example illustrates these two effects. Consider the figure:
- The nonlinear function is drawn, and the input random-variable PDF is shown on the horizontal axis, with mean 1.05.




- The straight dotted line is the linearized approximation used by the EKF to find the output mean and covariance.
- The EKF-approximated PDF is compared to a Gaussian PDF having same mean and variance of the true data on the vertical axis.

- We notice significant differences between the means and variances: EKF approach is not producing an accurate estimate of either.
- For a two-dimensional example, consider the following figure.




  - Left frame shows a cloud of Gaussian-distributed random points used as input to this function, and
  - Right frame shows the transformed set of output points.
- The actual 95 % confidence interval (indicative of a contour of the Gaussian PDF describing the output covariance and mean) is compared to EKF-estimated confidence interval.
  - Again, EKF is very far from the truth.
- We can improve on mean and covariance propagation through the state and output equations using a “sigma-point” approach.

## Approximating statistics with sigma points

- We now look at a different approach to characterizing the mean and covariance of the output of a nonlinear function.
- We avoid Taylor-series expansion; instead, a number of function evaluations are performed whose results are used to compute estimated mean and covariance matrices.

- This has several advantages:
  1. Derivatives do not need to be computed (which is one of the most error-prone steps when implementing EKF), also implying
  2. The original functions do not need to be differentiable, and
  3. Better covariance approximations are usually achieved, relative to EKF, allowing for better state estimation,
  4. All with comparable computational complexity to EKF.
- A set of sigma points $\mathcal{X}$ is chosen so that the (possibly weighted) mean and covariance of the points exactly matches the mean $\bar{x}$ and covariance $\Sigma_{\tilde{x}}$ of the *a priori* random variable being modeled.
- These points are then passed through the nonlinear function, resulting in a transformed set of points $\mathcal{Y}$.
- The *a posteriori* mean $\bar{y}$ and covariance $\Sigma_{\tilde{y}}$ are then approximated by the mean and covariance of these transformed points $\mathcal{Y}$.
- Note that the sigma points comprise a fixed small number of vectors that are calculated deterministically—not like particle filter methods.
- Specifically, if input RV $x$ has dimension $L$, mean $\bar{x}$, and covariance $\Sigma_{\tilde{x}}$, then $p+1 = 2L+1$ sigma points are generated as the set

$$\mathcal{X} = \left\{\bar{x}, \bar{x} + \gamma\sqrt{\Sigma_{\tilde{x}}}, \bar{x} - \gamma\sqrt{\Sigma_{\tilde{x}}}\right\},$$

  with members of $\mathcal{X}$ indexed from $0$ to $p$, and where the matrix square root $R = \sqrt{\Sigma}$ computes a result such that $\Sigma = RR^T$.
  - Usually, the efficient *Cholesky decomposition* is used, resulting in lower-triangular $R$. (Take care: MATLAB, by default, returns an upper-triangular matrix that must be transposed.)

- The weighted mean and covariance of $\mathcal{X}$ are equal to the original mean and covariance of $x$ for some $\{\gamma, \alpha^{(\mathrm{m})}, \alpha^{(\mathrm{c})}\}$ if we compute

$$\bar{x} = \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \mathcal{X}_i \qquad \text{and} \qquad \Sigma_{\tilde{x}} = \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} (\mathcal{X}_i - \bar{x})(\mathcal{X}_i - \bar{x})^T,$$

  where $\mathcal{X}_i$ is the $i$th member of $\mathcal{X}$, and both $\alpha_i^{(m)}$ and $\alpha_i^{(\mathrm{c})}$ are real scalars where $\alpha_i^{(m)}$ and $\alpha_i^{(\mathrm{c})}$ must both sum to one.

  - The various sigma-point methods differ only in the choices taken for these weighting constants.
  - Values used by the *Unscented Kalman Filter* (UKF) and the *Central Difference Kalman Filter* (CDKF):

| Method | $\gamma$ | $\alpha_0^{(\mathrm{m})}$ | $\alpha_k^{(\mathrm{m})}$ | $\alpha_0^{(\mathrm{c})}$ | $\alpha_k^{(\mathrm{c})}$ |
|---|---|---|---|---|---|
| UKF | $\sqrt{L+\lambda}$ | $\dfrac{\lambda}{L+\lambda}$ | $\dfrac{1}{2(L+\lambda)}$ | $\dfrac{\lambda}{L+\lambda} + (1-\alpha^2+\beta)$ | $\dfrac{1}{2(L+\lambda)}$ |
| CDKF | $h$ | $\dfrac{h^2-L}{h^2}$ | $\dfrac{1}{2h^2}$ | $\dfrac{h^2-L}{h^2}$ | $\dfrac{1}{2h^2}$ |

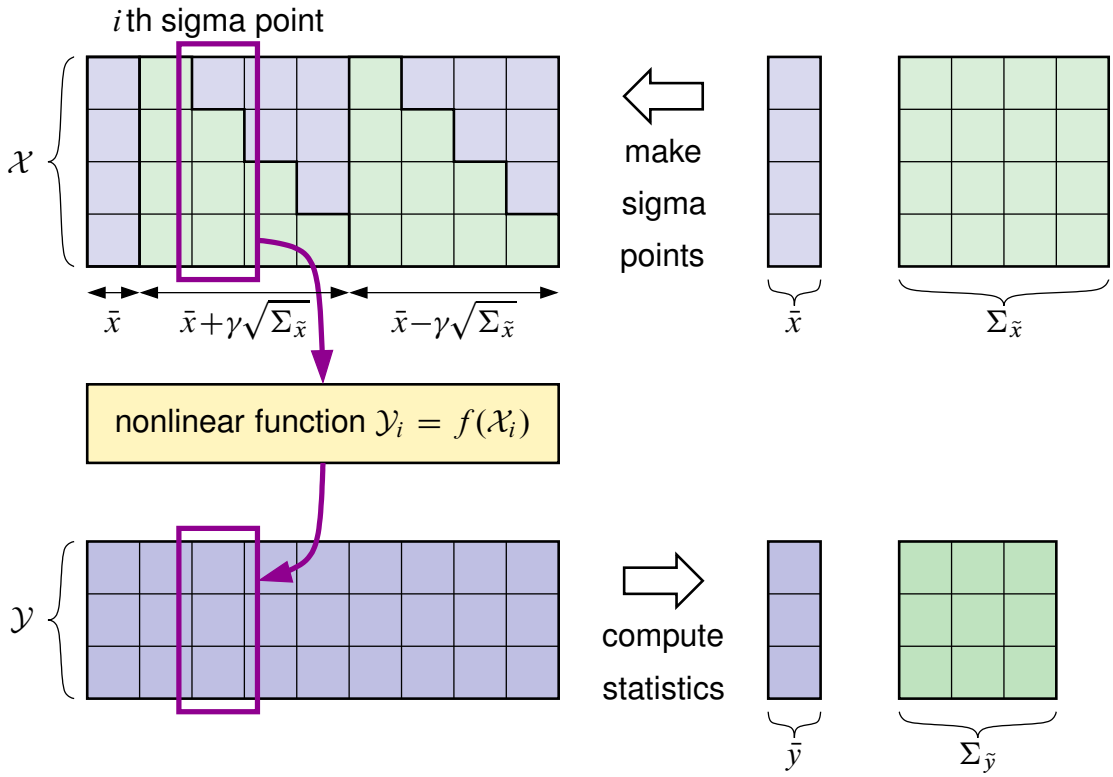
$\lambda = \alpha^2(L+\kappa) - L$ is a scaling parameter, with $(10^{-2} \leq \alpha \leq 1)$. Note that this $\alpha$ is different from $\alpha^{(\mathrm{m})}$ and $\alpha^{(\mathrm{c})}$. $\kappa$ is either 0 or $3-L$. $\beta$ incorporates prior information. For Gaussian RVs, $\beta = 2$. $h$ may take any positive value. For Gaussian RVs, $h = \sqrt{3}$.

  - UKF and CDKF are derived quite differently, but the final methods are essentially identical.
  - CDKF has only one "tuning parameter" $h$, so implementation is simpler. It also has marginally higher theoretic accuracy than UKF.

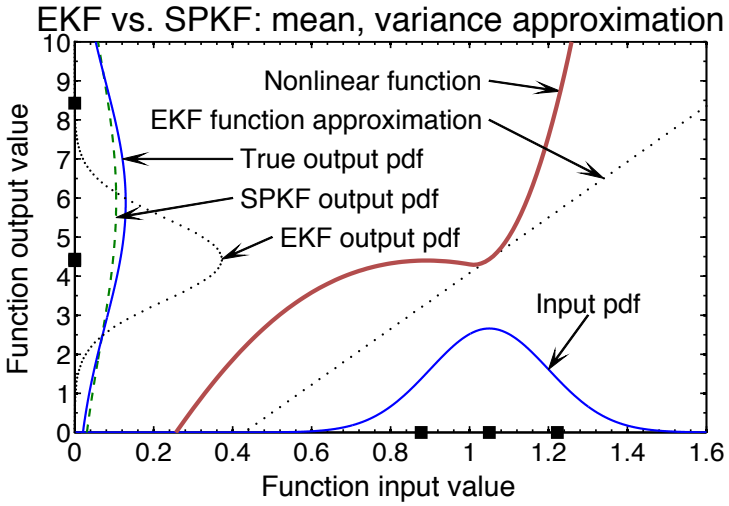
- Output sigma points are computed: $\mathcal{Y}_i = f(\mathcal{X}_i)$. Then, the output mean and covariance are computed as well:

$$\bar{y} = \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \mathcal{Y}_i \qquad \text{and} \qquad \Sigma_{\tilde{y}} = \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} (\mathcal{Y}_i - \bar{y})(\mathcal{Y}_i - \bar{y})^T.$$

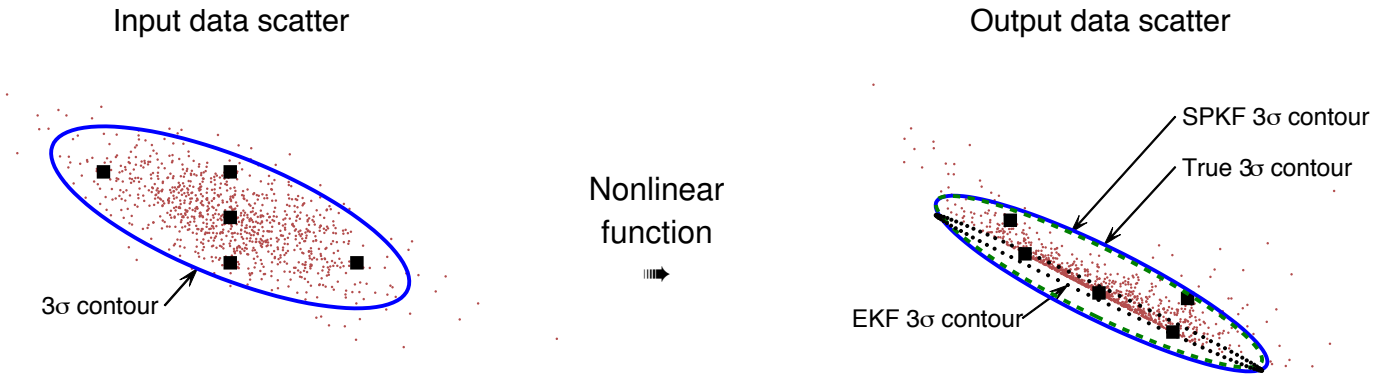
- The diagram illustrates the overall process, with the sets $\mathcal{X}$ and $\mathcal{Y}$ stored compactly with each set member a column in a matrix:




- Before introducing the SPKF algorithm, we re-examine the prior 1D/2D examples using sigma-point methods.
- In the 1D example, three input sigma points are needed and map to the output three sigma points shown.




- The mean and variance of the sigma-point method is shown as a dashed-line PDF and closely matches the true mean and variance.
- For the 2D example, five sigma points represent the input random-variable PDF (on left).




- These five points are transformed to five output points (frame (b)).
- We see that the mean and covariance of the output sigma points (dashed ellipse) closely match the true mean and covariance.
- Will the sigma-point method always be so much better than EKF?
  - The answer depends on the degree of nonlinearity of the state and output equations—the more nonlinear the better SPKF should be with respect to EKF.

## 3.16: The SPKF Steps

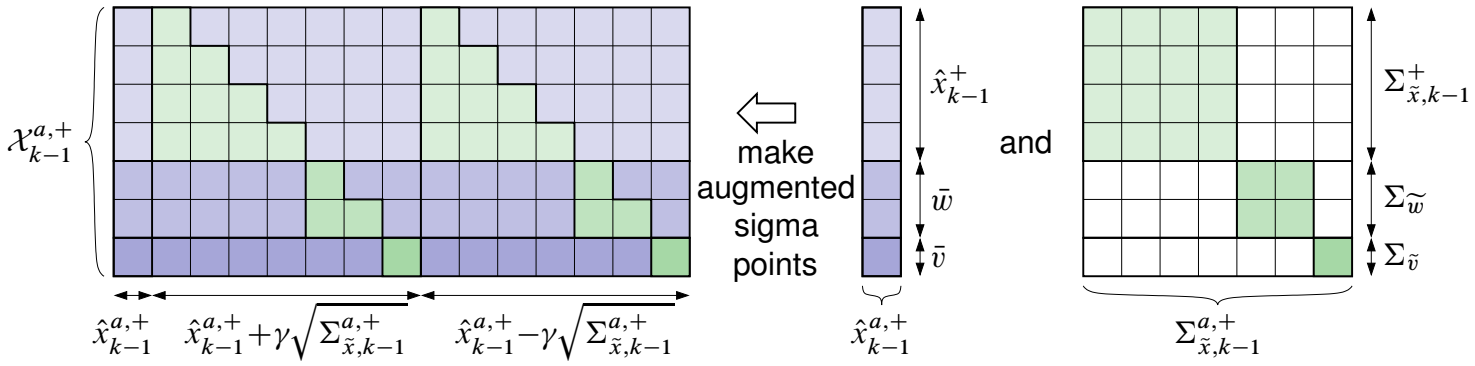
- We now apply the sigma-point approach of propagating statistics through a nonlinear function to the state-estimation problem.
- These sigma-points must jointly model all randomness: uncertainty of the state, process noise, and sensor noise.
- So we first define an augmented random vector $x_k^a$ that combines these random factors at time index $k$.
- This augmented vector is used in the estimation process as described below.

**SPKF step 1a:** State estimate time update.

- First, form the augmented *a posteriori* state estimate vector for the previous time interval: $\hat{x}_{k-1}^{a,+} = \left[(\hat{x}_{k-1}^{+})^T, \bar{w}, \bar{v}\right]^T$, and the augmented *a posteriori* covariance estimate: $\Sigma_{\tilde{x},k-1}^{a,+} = \mathrm{diag}\left(\Sigma_{\tilde{x},k-1}^{+}, \Sigma_{\tilde{w}}, \Sigma_{\tilde{v}}\right)$.
- These factors are used to generate the $p+1$ augmented sigma points

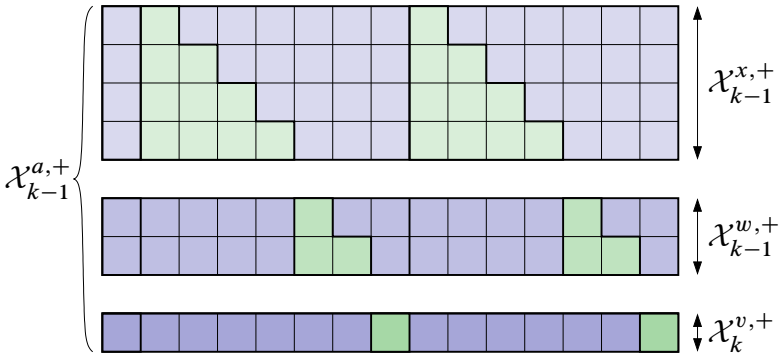
$$\mathcal{X}_{k-1}^{a,+} = \left\{ \hat{x}_{k-1}^{a,+},\ \hat{x}_{k-1}^{a,+} + \gamma\sqrt{\Sigma_{\tilde{x},k-1}^{a,+}},\ \hat{x}_{k-1}^{a,+} - \gamma\sqrt{\Sigma_{\tilde{x},k-1}^{a,+}} \right\}.$$

- Can be organized in convenient matrix form:

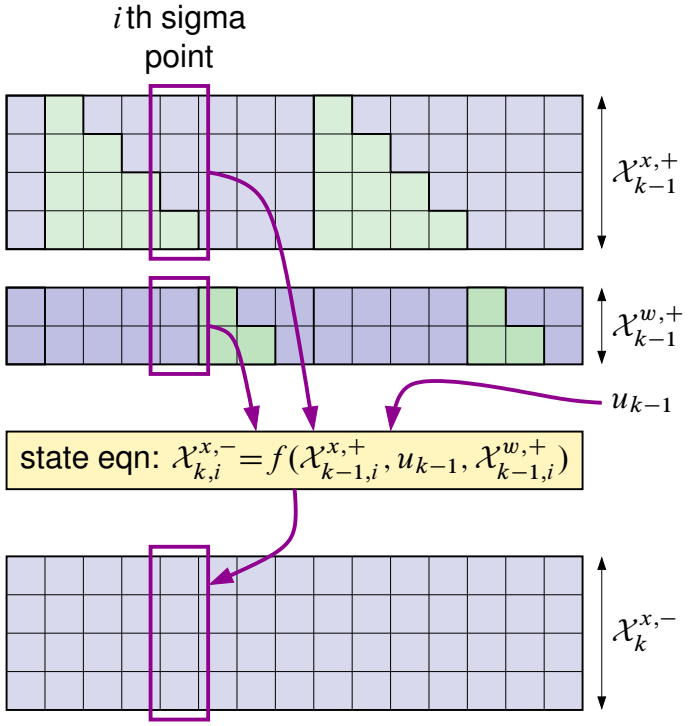


- Split augmented sigma points $\mathcal{X}_{k-1}^{a,+}$ into state portion $\mathcal{X}_{k-1}^{x,+}$, process-noise portion $\mathcal{X}_{k-1}^{w,+}$, and sensor-noise portion $\mathcal{X}_{k}^{v,+}$.




- Evaluate state equation using all pairs of $\mathcal{X}_{k-1,i}^{x,+}$ and $\mathcal{X}_{k-1,i}^{w,+}$ (where subscript $i$ denotes that the $i$th vector is being extracted from the original set), yielding the *a priori* sigma points $\mathcal{X}_{k,i}^{x,-}$.
- That is, compute $\mathcal{X}_{k,i}^{x,-} = f(\mathcal{X}_{k-1,i}^{x,+}, u_{k-1}, \mathcal{X}_{k-1,i}^{w,+})$.

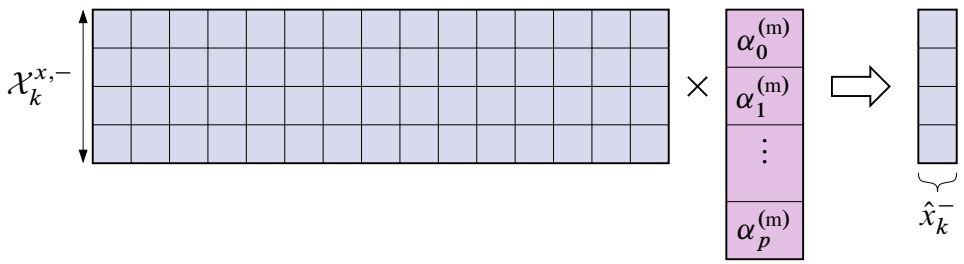



- Finally, the *a priori* state estimate is computed as

$$\hat{x}_k^- = \mathbb{E}\left[ f(x_{k-1}, u_{k-1}, w_{k-1}) \mid \mathbb{Y}_{k-1} \right] \approx \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} f(\mathcal{X}_{k-1,i}^{x,+}, u_{k-1}, \mathcal{X}_{k-1,i}^{w,+})$$

$$= \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \mathcal{X}_{k,i}^{x,-}.$$

- Can be computed with a simple matrix multiply operation.



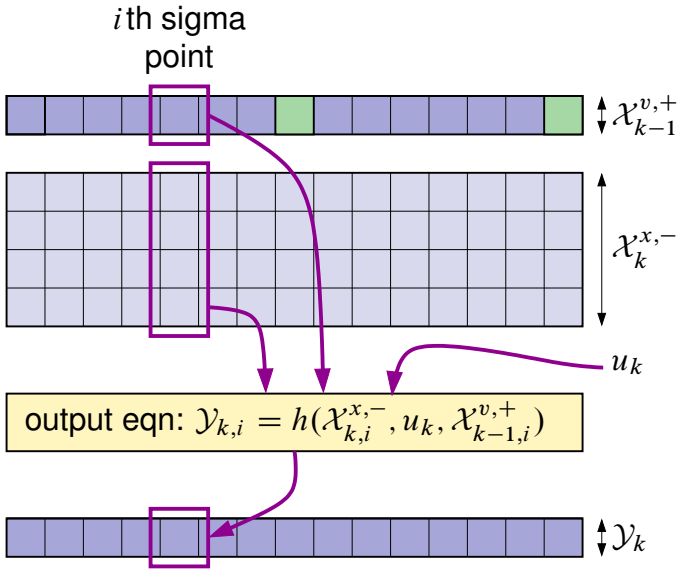
**SPKF step 1b:** Error covariance time update.

- Using the *a priori* sigma points from step 1a, the *a priori* covariance estimate is computed as

$$\Sigma_{\tilde{x},k}^{-} = \sum_{i=0}^{p} \alpha_i^{(c)} \left(\mathcal{X}_{k,i}^{x,-} - \hat{x}_k^{-}\right)\left(\mathcal{X}_{k,i}^{x,-} - \hat{x}_k^{-}\right)^T.$$

**SPKF step 1c:** Estimate system output $y_k$.

- The output $y_k$ is estimated by evaluating the model output equation using the sigma points describing the state and sensor noise.
- First, we compute the points $\mathcal{Y}_{k,i} = h(\mathcal{X}_{k,i}^{x,-}, u_k, \mathcal{X}_{k-1,i}^{v,+})$.




- The output estimate is then

$$\hat{y}_k = \mathbb{E}\left[h(x_k, u_k, v_k) \mid \mathbb{Y}_{k-1}\right] \approx \sum_{i=0}^{p} \alpha_i^{(m)} h(\mathcal{X}_{k,i}^{x,-}, u_k, \mathcal{X}_{k-1,i}^{v,+}) = \sum_{i=0}^{p} \alpha_i^{(m)} \mathcal{Y}_{k,i}.$$

- This can be computed with a simple matrix multiplication, as we did when calculating $\hat{x}_k^{-}$ at the end of step 1a.

**SPKF step 2a:** Estimator gain matrix $L_k$.

- To compute the estimator gain matrix, we must first compute the required covariance matrices.

$$\Sigma_{\tilde{y},k} = \sum_{i=0}^{p} \alpha_i^{(c)} \left(\mathcal{Y}_{k,i} - \hat{y}_k\right)\left(\mathcal{Y}_{k,i} - \hat{y}_k\right)^T$$

$$\Sigma^{-}_{\tilde{x}\tilde{y},k} = \sum_{i=0}^{p} \alpha_i^{(c)} \left(\mathcal{X}^{x,-}_{k,i} - \hat{x}^{-}_k\right)\left(\mathcal{Y}_{k,i} - \hat{y}_k\right)^T.$$

- These depend on the sigma-point matrices $\mathcal{X}^{x,-}_k$ and $\mathcal{Y}_k$, already computed in steps 1b and 1c, as well as $\hat{x}^{-}_k$ and $\hat{y}_k$, already computed in steps 1a and 1c.
- The summations can be performed using matrix multiplies, as we did in step 1b.
- Then, we simply compute $L_k = \Sigma^{-}_{\tilde{x}\tilde{y},k}\Sigma^{-1}_{\tilde{y},k}$.

**SPKF step 2b:** State estimate measurement update.

- The state estimate is computed as

$$\hat{x}^{+}_k = \hat{x}^{-}_k + L_k(y_k - \hat{y}_k).$$

**SPKF step 2c:** Error covariance measurement update.

- The final step is calculated directly from the optimal formulation: $\Sigma^{+}_{\tilde{x},k} = \Sigma^{-}_{\tilde{x},k} - L_k\Sigma_{\tilde{y},k}L_k^T$.

# 3.17: An SPKF example, with code

- Consider the same example used to illustrate EKF:

$$x_{k+1} = \sqrt{5 + x_k} + w_k$$

$$y_k = x_k^3 + v_k$$

with $\Sigma_{\widetilde{w}} = 1$ and $\Sigma_{\widetilde{v}} = 2$.

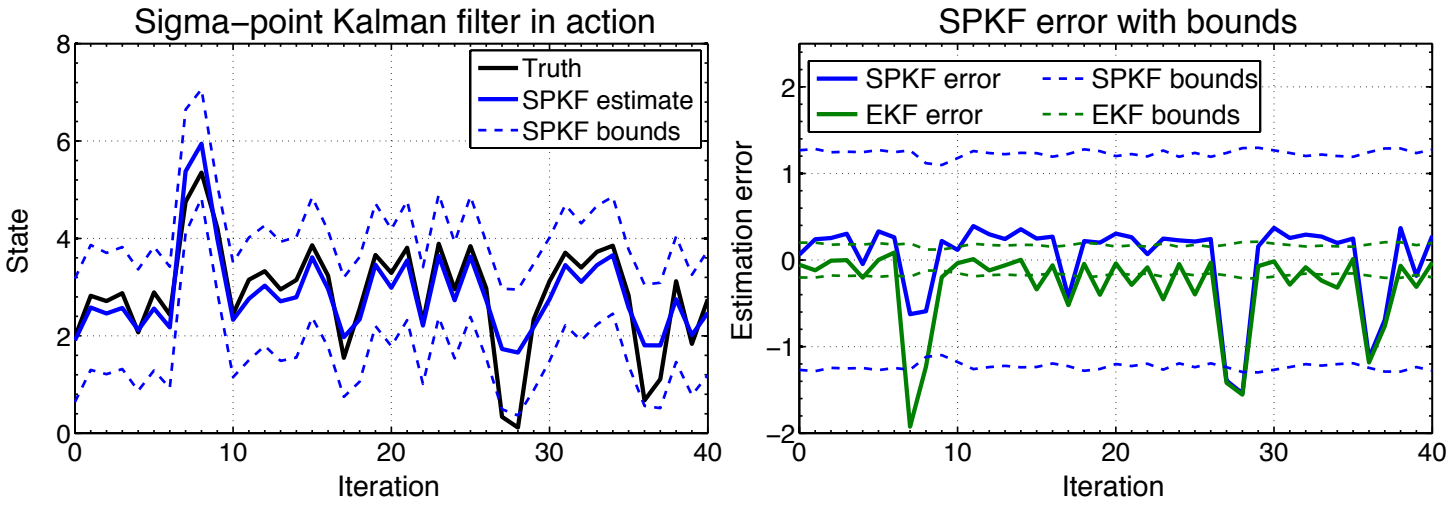
- The following is some sample code to implement SPKF

```
% Define size of variables in model
Nx = 1;      % state  = 1x1 scalar
Nxa = 3;     % augmented state has also w(k) and v(k) contributions
Ny = 1;      % output = 1x1 scalar

% Some constants for the SPKF algorithm. Use standard values for
% cases with Gaussian noises.  (These are the weighting matrices
% comprising the values of alpha(c) and alpha(m) organized in a
% way to make later computation efficient).
h = sqrt(3);
Wmx(1) = (h*h-Nxa)/(h*h); Wmx(2) = 1/(2*h*h); Wcx=Wmx;
Wmxy = [Wmx(1) repmat(Wmx(2),[1 2*Nxa])]';

% Initialize simulation variables
SigmaW = 1; % Process noise covariance
SigmaV = 2; % Sensor noise covariance
maxIter = 40;
xtrue = 2 + randn(1);  % Initialize true system initial state
xhat = 2;              % Initialize Kalman filter initial estimate
SigmaX = 1;          % Initialize Kalman filter covariance

% 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)^2);

for k = 1:maxIter,
  % SPKF Step 1a: State estimate time update
  % 1a-i: Calculate augmented state estimate, including ...
  xhata = [xhat; 0; 0]; % process and sensor noise mean
```

```
% 1a-ii: Get desired Cholesky factor
Sigmaxa = blkdiag(SigmaX,SigmaW,SigmaV);
sSigmaxa = chol(Sigmaxa,'lower');
% 1a-iii: Calculate sigma points (strange indexing of xhat to
 avoid
% "repmat" call, which is very inefficient in Matlab)
X = xhata(:,ones([1 2*Nxa+1])) + h*[zeros([Nxa 1]), ...
                                    sSigmaxa, -sSigmaxa];
% 1a-iv: Calculate state equation for every element
% Hard-code equation here for efficiency
Xx = sqrt(5+X(1,:)) + X(2,:);
xhat = Xx*Wmxy;

% SPKF Step 1b: Covariance of prediction
Xs = (Xx(:,2:end) - xhat(:,ones([1 2*Nxa])))*sqrt(Wcx(2));
Xs1 = Xx(:,1) - xhat;
SigmaX = Xs*Xs' + Wcx(1)*Xs1*Xs1';

% [Implied operation of system in background, with
% input signal u, and output signal y]
w = chol(SigmaW)'*randn(1);
v = chol(SigmaV)'*randn(1);
ytrue = xtrue^3 + v;  % y is based on present x and u
xtrue = sqrt(5+xtrue) + w;  % future x is based on present u

% SPKF Step 1c: Create output estimate
% Hard-code equation here for efficiency
Y = Xx.^3 + X(3,:);
yhat = Y*Wmxy;

% SPKF Step 2a: Estimator gain matrix
Ys = (Y(:,2:end) - yhat*ones([1 2*Nxa])) * sqrt(Wcx(2));
Ys1 = Y(:,1) - yhat;
SigmaXY = Xs*Ys' + Wcx(1)*Xs1*Ys1';
SigmaY  = Ys*Ys' + Wcx(1)*Ys1*Ys1';
Lx= SigmaXY/SigmaY;

% SPKF Step 2b: Measurement state update
xhat = xhat + Lx*(ytrue-yhat); % update prediction to estimate

% SPKF Step 2c: Measurement covariance update
SigmaX = SigmaX - Lx*SigmaY*Lx';
```

```
  % [Store information for evaluation/plotting purposes]
  xstore(k+1,:) = xtrue;
  xhatstore(k,:) = xhat;
  SigmaXstore(k,:) = (SigmaX(:))';
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;
legend('true','estimate','bounds');
xlabel('Iteration'); ylabel('State');
title('Sigma-point Kalman filter in action');

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




- Note the improved estimation accuracy, and greatly improved error bounds estimates.

# 3.18: Implementing SPKF on ESC model

- We refactor the SPKF code much like we did for EKF. The “wrapper” code is:

```
load CellModel           % loads "model" of cell

% Load cell-test data . Contains variable "DYNData" of which the field
% "script1" is of interest. It has sub-fields time, current, voltage, soc.
load('Cell_DYN_P25'); % loads data
T = 25; % Test temperature

time    = DYNData.script1.time(:);   deltat = time(2)-time(1);
time    = time-time(1); % start time at 0
current = DYNData.script1.current(:); % discharge > 0; charge < 0.
voltage = DYNData.script1.voltage(:);
soc     = DYNData.script1.soc(:);

% Reserve storage for computed results, for plotting
sochat = zeros(size(soc));
socbound = zeros(size(soc));

% Covariance values
SigmaX0 = diag([1e-3 1e-3 1e-2]); % uncertainty of initial state
SigmaV = 2e-1; % Uncertainty of voltage sensor, output equation
SigmaW = 1e1; % Uncertainty of current sensor, state equation

% Create spkfData structure and initialize variables using first
% voltage measurement and first temperature measurement
spkfData = initSPKF(voltage(1),T,SigmaX0,SigmaV,SigmaW,model);

% Now, enter loop for remainder of time, where we update the SPKF
% once per sample interval
hwait = waitbar(0,'Computing...');
for k = 1:length(voltage),
  vk = voltage(k); % "measure" voltage
  ik = current(k); % "measure" current
  Tk = T;          % "measure" temperature

  % Update SOC (and other model states)
  [sochat(k),socbound(k),spkfData] = iterSPKF(vk,ik,Tk,deltat,spkfData);
  % update waitbar periodically, but not too often (slow procedure)
```

```
  if mod(k,1000)==0, waitbar(k/length(current),hwait); end;
end
close(hwait);

figure(1); clf;  plot(time/60,100*sochat,time/60,100*soc); hold on
h = plot([time/60; NaN; time/60],...
         [100*(sochat+socbound); NaN; 100*(sochat-socbound)]);
title('SOC estimation using SPKF');
xlabel('Time (min)'); ylabel('SOC (%)');
legend('Estimate','Truth','Bounds'); grid on

fprintf('RMS SOC estimation error = %g%%\n',...
        sqrt(mean((100*(soc-sochat)).^2)));

figure(2); clf; plot(time/60,100*(soc-sochat)); hold on
h = plot([time/60; NaN; time/60],[100*socbound; NaN; -100*socbound]);
title('SOC estimation errors using SPKF');
xlabel('Time (min)'); ylabel('SOC error (%)'); ylim([-4 4]);
legend('Error','Bounds'); grid on

ind = find(abs(soc-sochat)>socbound);
fprintf('Percent of time error outside bounds = %g%%\n',...
        length(ind)/length(soc)*100);
```

- The SPKF initialization code is

```
function spkfData = initSPKF(v0,T0,SigmaX0,SigmaV,SigmaW,model)

  % Initial state description
  ir0   = 0;                          spkfData.irInd = 1;
  hk0   = 0;                          spkfData.hkInd = 2;
  SOC0  = SOCfromOCVtemp(v0,T0,model); spkfData.zkInd = 3;
  spkfData.xhat  = [ir0 hk0 SOC0]';    % initial state

  % Covariance values
  spkfData.SigmaX = SigmaX0;
  spkfData.SigmaV = SigmaV;
  spkfData.SigmaW = SigmaW;
  spkfData.Snoise = real(chol(diag([SigmaW; SigmaV]),'lower'));
  spkfData.Qbump = 5;

  % SPKF specific parameters
```

```
  Nx = length(spkfData.xhat); spkfData.Nx = Nx; % state-vector length
  Ny = 1; spkfData.Ny = Ny; % measurement-vector length
  Nu = 1; spkfData.Nu = Nu; % input-vector length
  Nw = size(SigmaW,1); spkfData.Nw = Nw; % process-noise-vector length
  Nv = size(SigmaV,1); spkfData.Nv = Nv; % sensor-noise-vector length
  Na = Nx+Nw+Nv; spkfData.Na = Na;     % augmented-state-vector length

  h = sqrt(3); spkfData.h = h; % SPKF/CDKF tuning factor
  Weight1 = (h*h-Na)/(h*h); % weighting factors when computing mean
  Weight2 = 1/(2*h*h);      % and covariance
  spkfData.Wm = [Weight1; Weight2*ones(2*Na,1)]; % mean
  spkfData.Wc = spkfData.Wm;                     % covar

  % previous value of current
  spkfData.priorI = 0;
  spkfData.signIk = 0;

  % store model data structure too
  spkfData.model = model;
end
```

- The SPKF iteration code is:

```
function [zk,zkbnd,spkfData] = iterSPKF(vk,ik,Tk,deltat,spkfData)
  model = spkfData.model;

  % Load the cell model parameters
  Q  = getParamESC('QParam',Tk,model);
  G  = getParamESC('GParam',Tk,model);
  M  = getParamESC('MParam',Tk,model);
  M0 = getParamESC('M0Param',Tk,model);
  RC = exp(-deltat./abs(getParamESC('RCParam',Tk,model)))';
  R  = getParamESC('RParam',Tk,model)';
  R0 = getParamESC('R0Param',Tk,model);
  eta = getParamESC('etaParam',Tk,model);
  if ik<0, ik=ik*eta; end;

  % Get data stored in spkfData structure
  I = spkfData.priorI;
  SigmaX = spkfData.SigmaX;
  xhat = spkfData.xhat;
  Nx = spkfData.Nx;
```

```
Nw = spkfData.Nw;
Nv = spkfData.Nv;
Na = spkfData.Na;
Snoise = spkfData.Snoise;
Wc = spkfData.Wc;
irInd = spkfData.irInd;
hkInd = spkfData.hkInd;
zkInd = spkfData.zkInd;
if abs(ik)>Q/100, spkfData.signIk = sign(ik); end;
signIk = spkfData.signIk;

% Step 1a: State estimate time update
%          - Create xhatminus augmented SigmaX points
%          - Extract xhatminus state SigmaX points
%          - Compute weighted average xhatminus(k)

% Step 1a-1: Create augmented SigmaX and xhat
[sigmaXa,p] = chol(SigmaX,'lower');
if p>0,
  fprintf('Cholesky error.  Recovering...\n');
  theAbsDiag = abs(diag(SigmaX));
  sigmaXa = diag(max(SQRT(theAbsDiag),SQRT(spkfData.SigmaW)));
end
sigmaXa=[real(sigmaXa) zeros([Nx Nw+Nv]); zeros([Nw+Nv Nx]) Snoise];
xhata = [xhat; zeros([Nw+Nv 1])];
% NOTE: sigmaXa is lower-triangular

% Step 1a-2: Calculate SigmaX points (strange indexing of xhata to
% avoid "repmat" call, which is very inefficient in MATLAB)
Xa = xhata(:,ones([1 2*Na+1])) + ...
     spkfData.h*[zeros([Na 1]), sigmaXa, -sigmaXa];

% Step 1a-3: Time update from last iteration until now
%     stateEqn(xold,current,xnoise)
Xx = stateEqn(Xa(1:Nx,:),I,Xa(Nx+1:Nx+Nw,:));
xhat = Xx*spkfData.Wm;

% Step 1b: Error covariance time update
%          - Compute weighted covariance sigmaminus(k)
%            (strange indexing of xhat to avoid "repmat" call)
Xs = Xx - xhat(:,ones([1 2*Na+1]));
SigmaX = Xs*diag(Wc)*Xs';
```

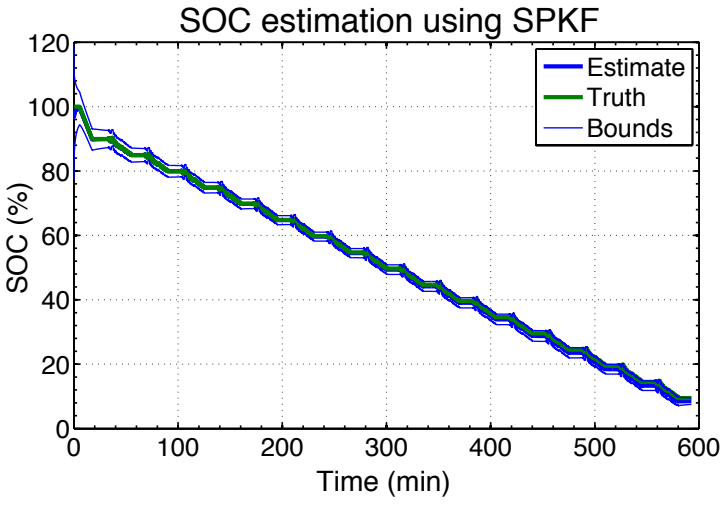
 

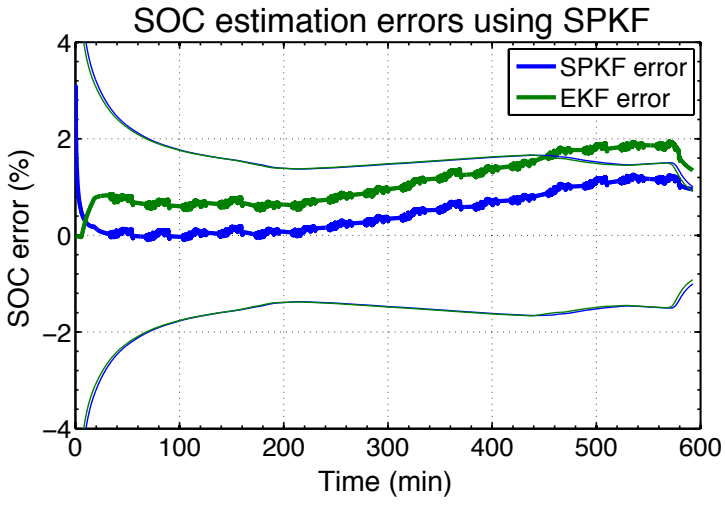
```
% Step 1c: Output estimate
%           - Compute weighted output estimate yhat(k)
I = ik; yk = vk;
Y = outputEqn(Xx,I,Xa(Nx+Nw+1:end,:),Tk,model);
yhat = Y*spkfData.Wm;

% Step 2a: Estimator gain matrix
Ys = Y - yhat(:,ones([1 2*Na+1]));
SigmaXY = Xs*diag(Wc)*Ys';
SigmaY = Ys*diag(Wc)*Ys';
L = SigmaXY/SigmaY;

% Step 2b: State estimate measurement update
r = yk - yhat; % residual.  Use to check for sensor errors...
if r^2 > 100*SigmaY, L(:,1)=0.0; end
xhat = xhat + L*r;
xhat(zkInd)=min(1.05,max(-0.05,xhat(zkInd)));

% Step 2c: Error covariance measurement update
SigmaX = SigmaX - L*SigmaY*L';
[~,S,V] = svd(SigmaX);
HH = V*S*V';
SigmaX = (SigmaX + SigmaX' + HH + HH')/4; % Help maintain robustness

% Q-bump code
if r^2>4*SigmaY, % bad voltage estimate by 2-SigmaX, bump Q
  fprintf('Bumping sigmax\n');
  SigmaX(zkInd,zkInd) = SigmaX(zkInd,zkInd)*spkfData.Qbump;
end

% Save data in spkfData structure for next time...
spkfData.priorI = ik;
spkfData.SigmaX = SigmaX;
spkfData.xhat = xhat;
zk = xhat(zkInd);
zkbnd = 3*sqrt(SigmaX(zkInd,zkInd));

% Calculate new states for all of the old state vectors in xold.
function xnew = stateEqn(xold,current,xnoise)
  current = current + xnoise; % noise adds to current
  xnew = 0*xold;
```

```
    xnew(irInd,:) = RC*xold(irInd,:) + (1-RC)*current;
    Ah = exp(-abs(current*G*deltat/(3600*Q)));  % hysteresis factor
    xnew(hkInd,:) = Ah.*xold(hkInd,:) - (1-Ah).*sign(current);
    xnew(zkInd,:) = xold(zkInd,:) - current/3600/Q;
    xnew(hkInd,:) = min(1,max(-1,xnew(hkInd,:)));
    xnew(zkInd,:) = min(1.05,max(-0.05,xnew(zkInd,:)));
  end

  % Calculate cell output voltage for all of state vectors in xhat
  function yhat = outputEqn(xhat,current,ynoise,T,model)
    yhat = OCVfromSOCtemp(xhat(zkInd,:),T,model);
    yhat = yhat + M*xhat(hkInd,:) + M0*signIk;
    yhat = yhat - R*xhat(irInd,:) - R0*current + ynoise(1,:);
  end

  % "Safe" square root
  function X = SQRT(x)
    X = sqrt(max(0,x));
  end
end
```

- For the following example, the SPKF was executed for the same test profiles as before.
  - RMS SOC estimation error = 0.65 %.
  - Percent of time error outside bounds = 0 %.



# 3.19: Real-world issues pertaining to sensors, initialization

## Current-sensor bias

- KF theory assumes that all noises are zero mean.
- An unknown current-sensor bias can introduce permanent SOC error.
  - Accumulated ampere-hours of bias tend to move SOC estimate faster than measurement updates can correct.
- Best solution would be to design sensing hardware to eliminate current-sensor bias, but this can be done only approximately.
- So, we can also attempt to correct for the (unknown, time-varying) bias algorithmically by estimating the bias.
- Using the ESC model as an example, we augment the pack state

$$\begin{aligned} z_k &= z_{k-1} - (i_{k-1} - i^b_{k-1} + w_{k-1})\Delta t/Q \\ i_{R_j,k} &= A_{RC} i_{R_j,k-1} + B_{RC}(i_{k-1} - i^b_{k-1} + w_{k-1}) \\ A_{h,k} &= \exp\left(-\left|(i_{k-1} - i^b_{k-1} + w_{k-1})\gamma\Delta t/Q\right|\right) \\ h_k &= A_{h,k} h_{k-1} + (1 - A_{h,k})\,\mathrm{sgn}(i_{k-1} - i^b_{k-1} + w_{k-1}) \\ i^b_k &= i^b_{k-1} + n^b_{k-1}, \end{aligned}$$

  where $n^b_k$ is a fictitious noise source included in the model only that allows the SPKF to adapt the bias state.
- The output equation is also modified:

$$y_k = \mathrm{OCV}(z_k) + M h_k - \sum_j R_j i_{R_j,k} - R_0(i_k - i^b_k) + v_k,$$

  where $v_k$ models sensor noise.

## Real-world issue: Voltage-sensor faults

- Consider first a one-output situation (common).
- Part of the SPKF calculates $\Sigma_{\tilde{y},k}$ , and we know $\sigma_{\tilde{y},k} = \sqrt{\Sigma_{\tilde{y},k}}$.
- If the absolute value of $\tilde{y}_k$ is "significantly" greater than $\sigma_{\tilde{y},k}$, then either our state estimate is way off, or we have a voltage sensor fault.
  1. We can skip measurement update of SPKF, and/or
  2. We can "bump up" $\Sigma_{\tilde{x},k}$ by multiplying it by a value greater than 1, especially if a number of "measurement errors" happen in a row.
- Both done in practice to aid robustness of a real implementation.
- For a multi-output model, define $z_k = M_k \tilde{y}_k$.
  - The mean of $z_k$ is $\mathbb{E}[z_k] = \mathbb{E}[M_k \tilde{y}_k] = 0$.
  - The covariance of $z_k$ is $\Sigma_{\tilde{z},k} = \mathbb{E}[M_k \tilde{y}_k \tilde{y}_k^T M_k^T] = M_k \Sigma_{\tilde{y},k} M_k^T$.
  - $z_k$ is Gaussian (since it is a linear combination of Gaussians).
- If we define $M_k$ such that $M_k^T M_k = \Sigma_{\tilde{y},k}^{-1}$, then
  - $M_k$ is the lower-triangular Cholesky factor of $\Sigma_{\tilde{y},k}^{-1}$.
  - We also have $z_k \sim \mathcal{N}(0, I)$ since

$$\Sigma_{\tilde{z},k} = M_k \left(M_k^T M_k\right)^{-1} M_k^T = M_k M_k^{-1} M_k^{-T} M_k^T = I.$$

- If we further compute normalized estimation error squared (NEES)

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

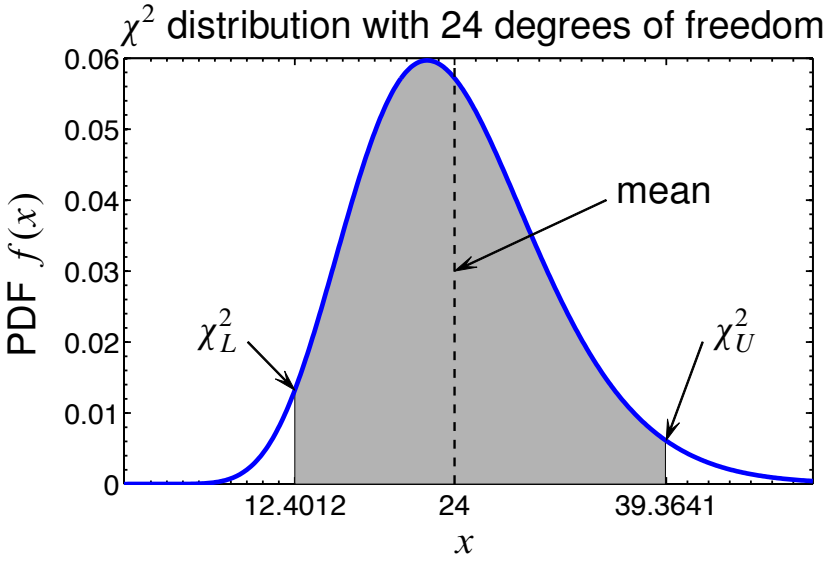
  then $e_k^2$ is the sum of squares of independent $\mathcal{N}(0,1)$ RVs.
- Then, $e_k^2$ is a chi-square RV with $m$ degrees of freedom, where $m$ is the dimension of $\tilde{y}_k$.

- Since it is a sum of squares, it is never negative; it is also asymmetric about its mean value.
- The PDF of a chi-square RV $X$ having $m$ degrees of freedom is

$$f_X(x) = \frac{1}{2^{m/2}\Gamma(m/2)} x^{(m/2-1)} e^{-m/2},$$

  which is tricky, but we never need to evaluate it in real time.
- Instead, we rely on values precomputed from the distribution.
- For confidence interval estimation we need to find two critical values.
- For $1-\alpha$ confidence of a valid measurement, want $\alpha/2$ area between 0 and $\chi^2_L$ and $\alpha/2$ area above $\chi^2_U$.
- Figure drawn for $\alpha = 0.05$.




- We find $\chi^2_L$ from where the inverse CDF of the distribution is equal to $\alpha/2$. In MATLAB:

```
X2L = chi2inv(0.025,24) % Lower critical value X2L = 12.4012
```

- We find $\chi^2_U$ from where the inverse CDF is equal to $1-\alpha/2$. In MATLAB:

```
X2U = chi2inv(1-0.025,24) % Upper critical value X2U = 39.3641
```

- Note that $\chi^2_L$ and $\chi^2_U$ need to be computed once only, offline.
  - They are based on the number of measurements in the output vector and the desired confidence level $1-\alpha$ only.
  - They do not need to be recalculated as the EKF or SPKF runs.

- For hand calculations a $\chi^2$-table is available on page 3–96.
- If a value of $\tilde{y}_k < \chi^2_L$ or if $\tilde{y}_k > \chi^2_U$ , then the measurement is discarded. Otherwise, the measurement is kept.

## Real-world issue: Other sensor faults

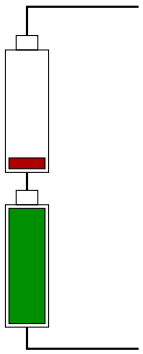
- Not obvious how to catch temperature- and current-sensor faults.
  - Current can change "instantly" from $i_{\min}$ to $i_{\max}$.
  - Might consider thermal model of cell/module to
    1. Reduce number of required temperature sensors, and
    2. Catch sensor faults as done above for voltage sensor.

## Real-world issue: Initialization

- If vehicle is off for a "long" time, just assume that cell voltage is equivalent to OCV:
  - Reset SOC estimate based on OCV.
  - Set diffusion voltages to zero.
  - Keep prior value of hysteresis state.
- If vehicle has been off for a "short" period of time
  - Set up and execute simple time/measurement update (simple KF) equations for SOC and diffusion voltages.
  - Hysteresis voltages do not change.
  - Run a single-step Kalman filter to update state estimate based on total time off.

## 3.20: Real-world issue: Speed, solved by "bar-delta" filtering

- We consider again a philosophical question with very important practical implications:
- Consider the picture to the right. What is the pack SOC?



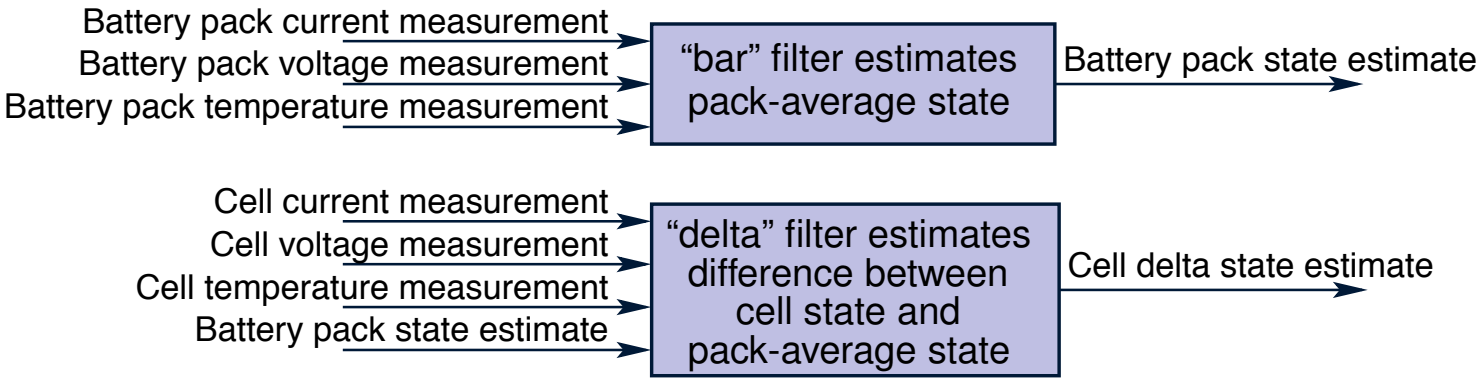
  - SOC cannot be 0 % because we cannot charge.
  - SOC cannot be 100 % because we cannot discharge.
  - SOC cannot be the average of the two, 50 %, because we can neither charge nor discharge.
- So, battery "pack SOC" is not a helpful concept, by itself.
- The example is an extreme case, but it is important to estimate the SOC of all cells even in the typical case.
- The problem is that the SPKF is computationally complex.
  - Running SPKF for one cell is okay, but
  - Running 100 SPKFs for 100 cells is probably not okay.
- In this section we talk about efficient SOC estimation for all individual cells in a large battery pack.

**OBSERVATION:** While "pack SOC" does not make sense, the concept of "pack-average SOC" is a useful one.

- Since all cells in a series string experience the same current, we expect their SOC values to
  1. Move in the same direction for any given applied current, by
  2. A similar amount (but different because of unequal cell capacities).
- We take advantage of this similarity by creating:

- One algorithm to determine the composite average behavior of all cells in the battery pack, and
- Another algorithm to determine the individual differences between specific cells and that composite average behavior.

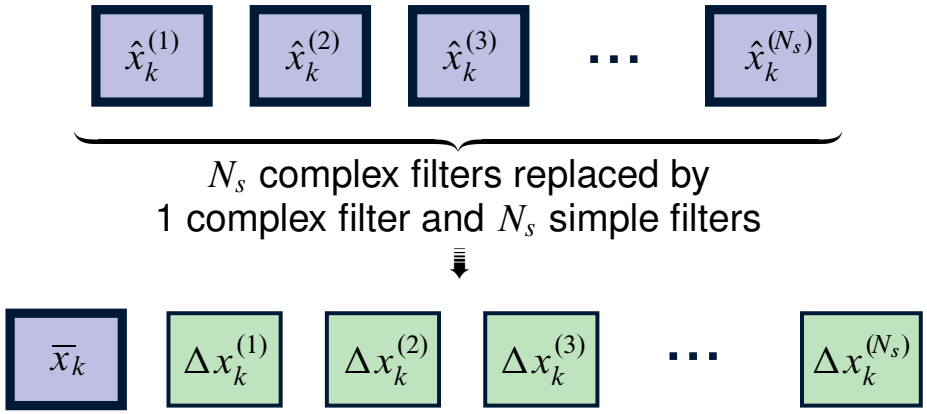
- We define pack-average state "$x$-bar" as $\bar{x}_k = \dfrac{1}{N_s}\sum_{i=1}^{N_s} x_k^{(i)}$.
  - Note that $0 \leq \min_i(z_k^{(i)}) \leq \bar{z}_k \leq \max_i(z_k^{(i)}) \leq 1$; therefore, its range is within the standard SOC range.
- We can then write an individual cell's state vector as $x_k^{(i)} = \bar{x}_k + \Delta x_k^{(i)}$ where $\Delta x_k^{(i)}$ (called "delta-$x$") is the difference between the state vector of cell $i$ and the pack-average state vector.
  - The method is called "bar-delta filtering," as inspired by the "$x$-bar" and "delta-$x$" naming convention.
- We use one SPKF to estimate the pack-average state, and $N_s$ SPKFs (or similar) to estimate the delta states:




- It may seem that we have taken a problem of complexity $N_s$ and replaced it with a problem of complexity $N_s + 1$.
- However, this is not the case—the three different types of estimator involved are not of identical computational complexity.

- The bar filter *is* of the same computational complexity as the individual state estimators that it uses as a basis (*e.g.,* SPKF).
- However, the delta filters can be made very simple.
- Also, the delta states change much more slowly than the average state, so the delta filters can be run less frequently, down to $1/N_s$ times the rate of the bar filter.
- Overall complexity can be reduced from order $N_s$ to order $1^+$.



## 3.21: Bar-delta filtering using the ESC cell model

*The pack bar filter*

- In the implementation that we describe here, a pack-average SPKF estimated the following quantities:
  - The pack-average state-of-charge, pack-average diffusion current(s), and the pack-average hysteresis voltage.
- We model current-sensor bias as

$$i_k^b = i_{k-1}^b + n_{k-1}^b,$$

  where $n_k^b$ is a fictitious noise source that is included in the model to allow SPKF to adapt the bias estimate.
- We now need to find the pack-average-quantity state equations.
- For example, starting with a single-cell SOC equation

$$z_k^{(i)} = z_{k-1}^{(i)} - i_{k-1}\Delta t / Q^{(i)}$$

$$\frac{1}{N_s}\sum_{i=1}^{N_s} z_k^{(i)} = \frac{1}{N_s}\sum_{i=1}^{N_s} z_{k-1}^{(i)} - \frac{i_{k-1}\Delta t}{N_s}\sum_{i=1}^{N_s}\frac{1}{Q^{(i)}}$$

$$= \frac{1}{N_s}\sum_{i=1}^{N_s} z_{k-1}^{(i)} - \frac{i_{k-1}\Delta t}{N_s}\sum_{i=1}^{N_s} Q_{\text{inv}}^{(i)}$$

$$\bar{z}_k = \bar{z}_{k-1} - i_{k-1}\Delta t\,\bar{Q}_{\text{inv}}.$$

- Note the new concept of "inverse capacity" to make the equations simpler. If we are estimating all cells' capacities, we then have a time-varying quantity $\bar{Q}_{\text{inv},k-1}$.

- And, if we also consider the current-bias state,

$$\bar{z}_k = \bar{z}_{k-1} - (i_{k-1} - i^b_{k-1})\Delta t\, \bar{Q}_{\text{inv},k-1}.$$

- Similarly, the dynamics of all pack-average states and parameters of interest may be summarized as:

$$\begin{aligned}
\bar{z}_k &= \bar{z}_{k-1} - (i_{k-1} - i^b_{k-1})\Delta t\, \bar{Q}_{\text{inv},k-1} \\
\bar{i}_{R_j,k} &= A_{RC}\bar{i}_{R_j,k} + B_{RC}(i_{k-1} - i^b_{k-1}) \\
A_{h,k} &= \exp\left(-\left|(i_{k-1} - i^b_{k-1})\gamma\Delta t\, \bar{Q}_{\text{inv},k-1}\right|\right) \\
\bar{h}_k &= A_{h,k}\bar{h}_{k-1} + (1 - A_{h,k})\,\text{sgn}(i_{k-1} - i^b_{k-1}) \\
\bar{R}_{0,k} &= \bar{R}_{0,k-1} + n^{\bar{R}_0}_{k-1} \\
\bar{Q}_{\text{inv},k} &= \bar{Q}_{\text{inv},k-1} + n^{\bar{Q}_{\text{inv}}}_{k-1} \\
i^b_k &= i^b_{k-1} + n^b_{k-1},
\end{aligned}$$

  where $n^{\bar{R}_0}_k$ and $n^{\bar{Q}_{\text{inv}}}_k$ are fictitious noise sources that allow the SPKF to adapt the corresponding pack-average parameters.

- The bar-filter for the pack employs an SPKF that uses this model of pack-average states and the measurement equation

$$\bar{y}_k = \text{OCV}(\bar{z}_k) + M\bar{h}_k - \sum_j R_j\bar{i}_{R_j,k} - \bar{R}_{0,k}(i_k - i^b_k) + v_k,$$

  where $v_k$ models sensor noise.

*The cell delta filters*

- The quantities that we are most interested in estimating at the individual cell level are: SOC, resistance, and capacity.

- These all factor into determining pack available power and lifetime (state-of-health) estimates.
- We will first consider the delta filter approach to determining cell SOC.
- Note, from before, $\Delta z_k^{(i)} = z_k^{(i)} - \bar{z}_k$. Then, using prior equations for the dynamics of $z_k^{(i)}$ and $\bar{z}_k$, we find:

$$\begin{aligned}\Delta z_k^{(i)} &= z_k^{(i)} - \bar{z}_k \\ &= \left(z_{k-1}^{(i)} - (i_{k-1} - i_{k-1}^b)\Delta t\, Q_{\text{inv},k-1}^{(i)}\right) - \left(\bar{z}_{k-1} - (i_{k-1} - i_{k-1}^b)\Delta t\, \bar{Q}_{\text{inv},k-1}\right) \\ &= \Delta z_{k-1}^{(i)} - (i_{k-1} - i_{k-1}^b)\Delta t\, \Delta Q_{\text{inv},k-1}^{(i)}\end{aligned}$$

  where $\Delta Q_{\text{inv},k}^{(i)} = Q_{\text{inv},k}^{(i)} - \bar{Q}_{\text{inv},k}$.
- Because $\Delta Q_{\text{inv},k}^{(i)}$ tends to be small, the state $\Delta z_k^{(i)}$ does not change quickly, and can be updated at a slower rate than the pack-average SOC by accumulating $(i_{k-1} - i_{k-1}^b)\Delta t$ in-between updates.
- An output equation suitable for combining with this state equation is

$$y_k^{(i)} = \text{OCV}(\bar{z}_k + \Delta z_k^{(i)}) + M\bar{h}_k - \sum_j R_j \bar{i}_{R_j,k} - (\bar{R}_{0,k} + \Delta R_{0,k}^{(i)})(i_k - i_k^b) + v_k$$

- To estimate $\Delta z_k^{(i)}$, an SPKF is used with these two equations. Since it is a single-state SPKF, it is very fast.
- As a preview of parameter estimation (talked about more in the next chapter...) we can similarly make state-space models of the delta-resistance and delta capacity states.
- A simple state-space model of the delta-resistance state is:

$$\begin{aligned}\Delta R_{0,k}^{(i)} &= \Delta R_{0,k-1}^{(i)} + n_{k-1}^{\Delta R_0} \\ y_k &= \text{OCV}(\bar{z}_k + \Delta z_k^{(i)}) - (\bar{R}_{0,k} + \Delta R_{0,k}^{(i)})(i_k - i_k^b) + v_k^{\Delta R_0},\end{aligned}$$

where $\Delta R_{0,k}^{(i)} = R_{0,k}^{(i)} - \bar{R}_{0,k}$ and is modeled as a constant value with a fictitious noise process $n_k^{\Delta R_0}$ allowing adaptation, $y_k$ is a crude estimate of the cell's voltage, and $v_k^{\Delta R_0}$ models estimation error.

- The dynamics of the delta-resistance state are simple and linear enough to use a single-state EKF rather than an SPKF.
- To estimate cell capacity using an EKF, we model

$$\Delta Q_{\text{inv},k}^{(i)} = \Delta Q_{\text{inv},k-1}^{(i)} + n_{k-1}^{\Delta Q_{\text{inv}}}$$

$$d_k = (z_k^{(i)} - z_{k-1}^{(i)}) + (i_{k-1} - i_{k-1}^b)\Delta t \times \left(\bar{Q}_{\text{inv},k-1} + \Delta Q_{\text{inv},k-1}^{(i)}\right) + e_k$$

  The second equation is a reformulation of the SOC state equation such that the expected value of $d_k$ is equal to zero by construction.
- As the EKF runs, the computation for $d_k$ in the second equation is compared to the known value (zero, by construction), and the difference is used to update the inverse-capacity estimate.
- Note that good estimates of present and previous SOCs are required.
  - Here, they come from the pack SPKF combined with the cell SPKF.
- The output of the delta filters is computed by combining the average battery pack state with the battery cell module delta states produced by the individual Kalman filters:

$$z_k^{(i)} = \bar{z}_k + \Delta z_k^{(i)}$$

$$R_{0,k}^{(i)} = \bar{R}_{0,k} + \Delta R_{0,k}^{(i)}$$

$$Q_k^{(i)} = \frac{1}{\bar{Q}_{\text{inv},k} + \Delta Q_{\text{inv},k}^{(i)}}$$

## 3.22: Example of bar-delta, using desktop validation

- Two basic approaches to algorithm validation

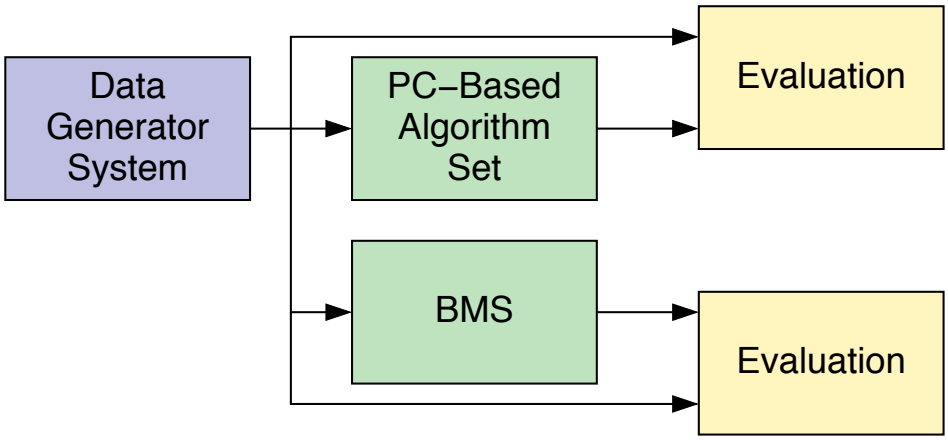
**HARDWARE IN THE LOOP (HIL):**

- HIL reveals baseline truth, but it is sometimes hard/impossible to determine "truth" of all cell states based on recorded data.

**DESKTOP VALIDATION:**

- Use model of cell to create synthetic test data.
- Allows access to "truth" of all cell and algorithm states.
- Very useful for tuning algorithms.
- Validity of results limited by the accuracy of cell model.

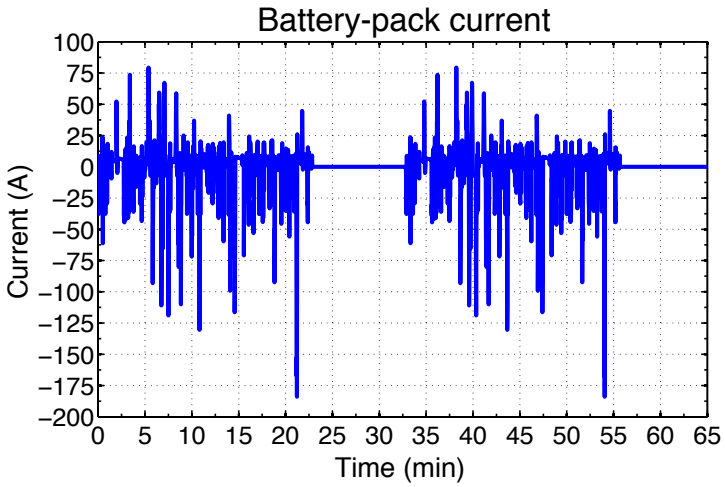
- With desktop validation, we need a "data generation" component:
  - Creates synthetic BMS data based on drive cycles and other initialization parameters.
- And, we need a "BMS algorithm simulation" component:
  - Simulates the SPKF algorithms using the synthetic data as input, based on various initialization parameters.

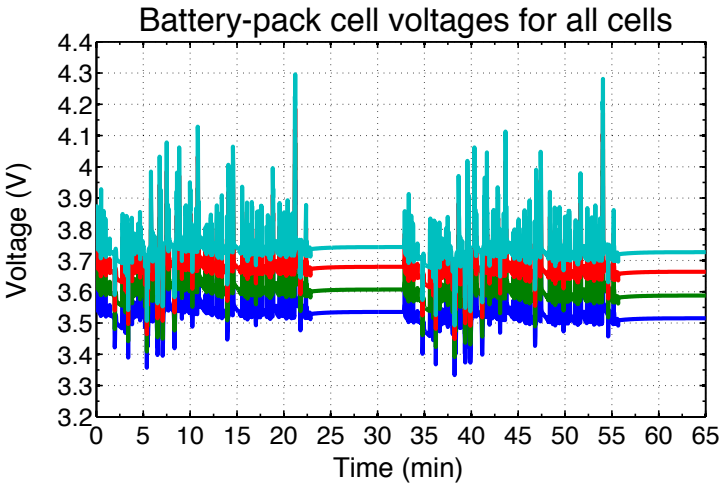


- Either way, validation scenarios include: Normal operation, improper SOC initialization, sensor failures (fault + noise), temperature drift, new and old cells mixed, different drive cycles, current-sensor bias.
- Insight and validation via analysis and display of outputs.

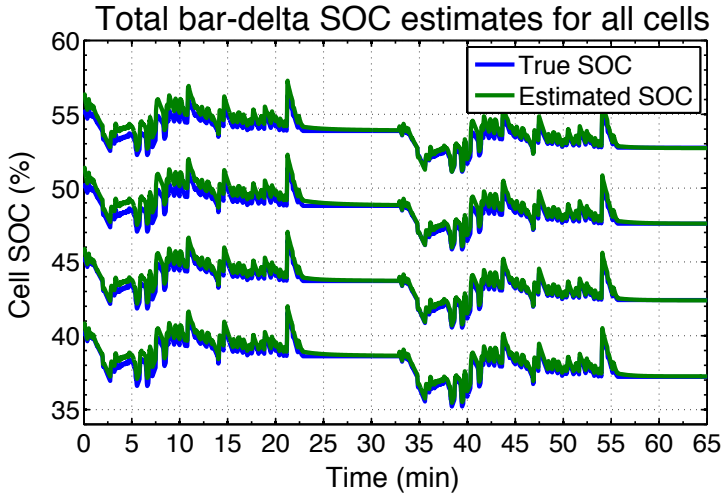
## Examples of bar-delta accuracy and speed

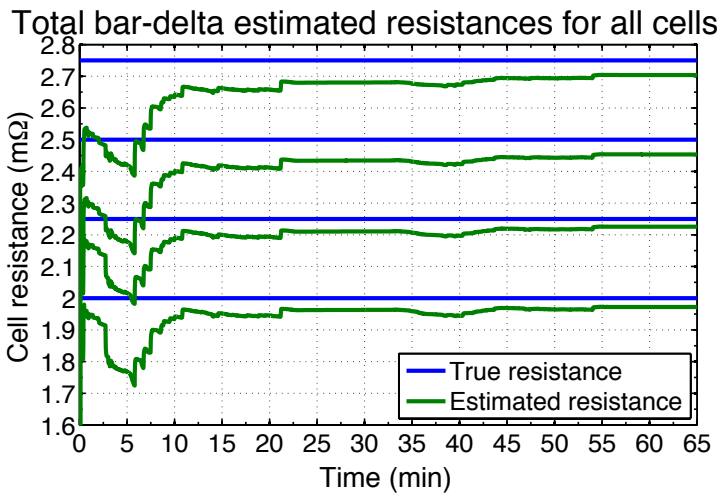
- Simulated cycling of a four-cell pack with a UDDS cycle, a rest period, the same UDDS cycle, and a rest period.
- The pack cells had true capacities of 6.5, 7.0, 7.5, and 8.0 Ah, resistances of 2.0, 2.25, 2.5, and 2.75 mΩ, and initial SOC values of 40, 45, 50, and 55 %. The current-sensor bias was 0.5 A.
- The algorithms were initialized with all cells having estimated capacity of 6.2 Ah, estimated resistances of 2.25 mΩ, estimated current-sensor bias of 0 A, and initial SOC estimates based on initial voltages.
- SPKF was used for the bar filter and the SOC delta filters, and EKF was used for the resistance and capacity-inverse delta filters.
- Pack current and individual cell voltage profiles for algorithm testing:



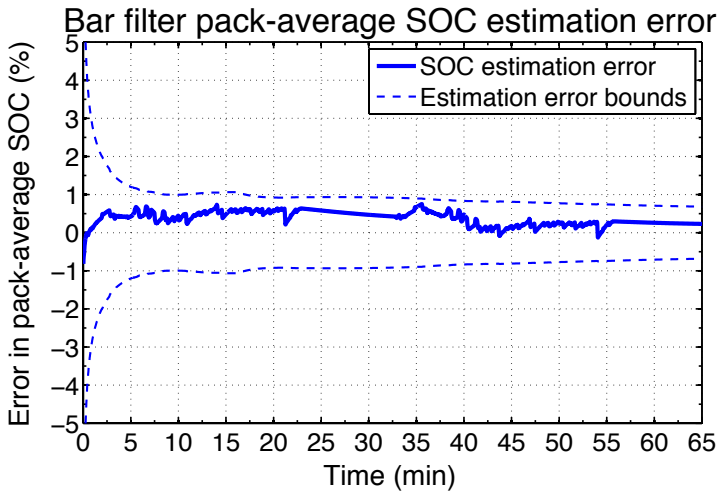



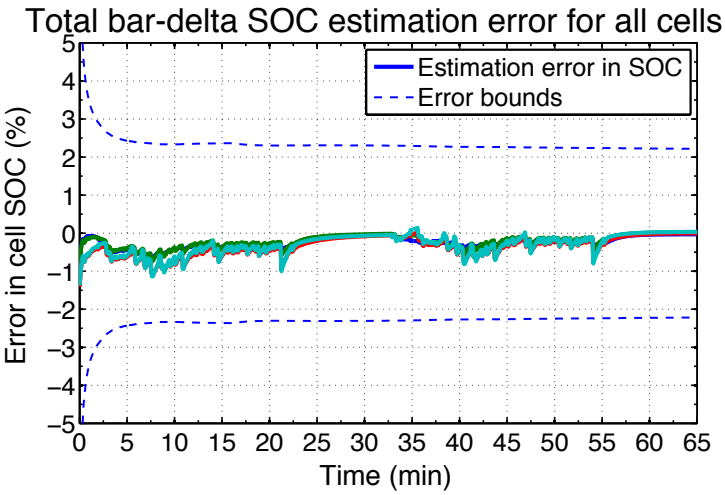
- Some results showing accuracy of the method:



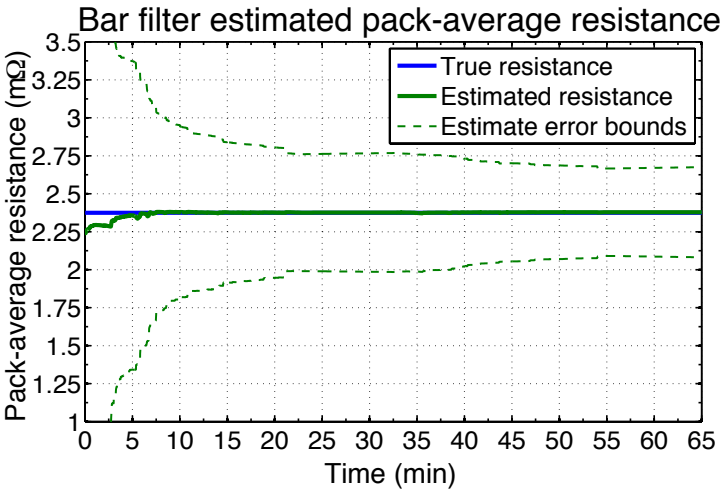



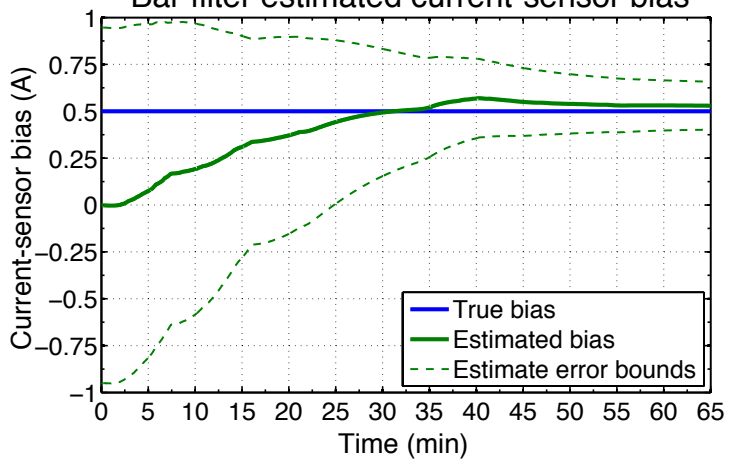

- A different point of view, with SOC estimation errors:




- Some other estimation accuracies:



- Capacity estimates evolve in a similar way to resistance estimates.
  - However, the time scale of adaptation is much longer, since capacity is very weakly linked to the output measurement.
  - Abrupt changes in capacity will not be tracked very quickly; but, capacity fade due to normal aging will be tracked very well.
- Speedup of the method (hand-coded C code, run on G4 processor)

| Description of test (for pack comprising 100 cells) | CPU time per iteration | Speedup |
|---|---|---|
| One SPKF per cell | 5.272 ms | 1.0 |
| One pack bar filter only, no delta filters | 0.067 ms | 78.7 |
| One pack bar filter, 100 delta filters updated per iteration | 0.190 ms | 27.7 |
| One pack bar filter, 50 delta filters updated per iteration | 0.123 ms | 42.9 |

## Where from here?

- We have seen good and bad ways to estimate SOC for all cells.
- Model-based methods are preferred; KF-based methods are “optimal” in some sense.
- Additionally, KF estimates entire state—not only SOC—therefore can also be used for degradation predictions. . .
- Lots of nuances unexplored in this short section of notes. ECE5550 goes into much more depth and breadth of implementation of KF
- Our next step is to look at state-of-health estimation, which is a form of parameter estimation.

# Appendix: General sequential probabilistic inference solution

**General state-space model:**

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1})$$

$$y_k = h(x_k, u_k, v_k),$$

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

**Definitions:** Let

$$\tilde{x}_k^- = x_k - \hat{x}_k^-, \qquad \tilde{y}_k = y_k - \hat{y}_k.$$

**Initialization:** For $k = 0$, set

$$\hat{x}_0^+ = \mathbb{E}\big[x_0\big]$$

$$\Sigma_{\tilde{x},0}^+ = \mathbb{E}\big[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T\big].$$

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

*State estimate time update:* $\hat{x}_k^- = \mathbb{E}\big[f(x_{k-1}, u_{k-1}, w_{k-1}) \mid \mathbb{Y}_{k-1}\big].$

*Error covariance time update:* $\Sigma_{\tilde{x},k}^- = \mathbb{E}\big[(\tilde{x}_k^-)(\tilde{x}_k^-)^T\big].$

*Output estimate:* $\hat{y}_k = \mathbb{E}\big[h(x_k, u_k, v_k) \mid \mathbb{Y}_{k-1}\big].$

*Estimator gain matrix:* $L_k = \mathbb{E}\big[(\tilde{x}_k^-)(\tilde{y}_k)^T\big]\Big(\mathbb{E}\big[(\tilde{y}_k)(\tilde{y}_k)^T\big]\Big)^{-1}.$

*State estimate measurement update:* $\hat{x}_k^+ = \hat{x}_k^- + L_k\big(y_k - \hat{y}_k\big).$

*Error covariance measurement update:* $\Sigma_{\tilde{x},k}^+ = \Sigma_{\tilde{x},k}^- - L_k \Sigma_{\tilde{y},k} L_k^T.$

# Appendix: Summary of the linear Kalman filter

**Linear state-space model:**

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

$$y_k = C_k x_k + D_k u_k + v_k,$$

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

$$\hat{x}_0^+ = \mathbb{E}[x_0]$$

$$\Sigma_{\tilde{x},0}^+ = \mathbb{E}\big[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T\big].$$

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

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

*Estimator gain matrix:** $L_k = \Sigma_{\tilde{x},k}^- C_k^T [C_k \Sigma_{\tilde{x},k}^- C_k^T + \Sigma_{\tilde{v}}]^{-1}.$

*State estimate measurement update:* $\hat{x}_k^+ = \hat{x}_k^- + L_k\big(y_k - \hat{y}_k\big).$

*Error covariance measurement update:* $\Sigma_{\tilde{x},k}^+ = (I - L_k C_k)\Sigma_{\tilde{x},k}^-.$

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

# Appendix: Summary of the nonlinear extended Kalman filter

**Nonlinear state-space model:**

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1})$$

$$y_k = h(x_k, u_k, v_k),$$

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

**Definitions:**

$$\hat{A}_k = \left.\frac{\mathrm{d}f(x_k, u_k, w_k)}{\mathrm{d}x_k}\right|_{x_k=\hat{x}_k^+} \qquad \hat{B}_k = \left.\frac{\mathrm{d}f(x_k, u_k, w_k)}{\mathrm{d}w_k}\right|_{w_k=\bar{w}_k}$$

$$\hat{C}_k = \left.\frac{\mathrm{d}h(x_k, u_k, v_k)}{\mathrm{d}x_k}\right|_{x_k=\hat{x}_k^-} \qquad \hat{D}_k = \left.\frac{\mathrm{d}h(x_k, u_k, v_k)}{\mathrm{d}v_k}\right|_{v_k=\bar{v}_k}.$$

**Initialization:** For $k = 0$, set

$$\hat{x}_0^+ = \mathbb{E}[x_0]$$

$$\Sigma_{\tilde{x},0}^+ = \mathbb{E}\big[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T\big].$$

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

*State estimate time update:* $\hat{x}_k^- = f(\hat{x}_{k-1}^+, u_{k-1}, \bar{w}_{k-1}).$

*Error covariance time update:* $\Sigma_{\tilde{x},k}^- = \hat{A}_{k-1}\Sigma_{\tilde{x},k-1}^+\hat{A}_{k-1}^T + \hat{B}_{k-1}\Sigma_{\widetilde{w}}\hat{B}_{k-1}^T.$

*Output estimate:* $\hat{y}_k = h(\hat{x}_k^-, u_k, \bar{v}_k).$

*Estimator gain matrix:* $L_k = \Sigma_{\tilde{x},k}^-\hat{C}_k^T[\hat{C}_k\Sigma_{\tilde{x},k}^-\hat{C}_k^T + \hat{D}_k\Sigma_{\tilde{v}}\hat{D}_k^T]^{-1}.$

*State estimate measurement update:* $\hat{x}_k^+ = \hat{x}_k^- + L_k\big(y_k - \hat{y}_k\big).$

*Error covariance measurement update:* $\Sigma_{\tilde{x},k}^+ = (I - L_k\hat{C}_k)\Sigma_{\tilde{x},k}^-.$

# Appendix: Summary of the nonlinear sigma-point Kalman filter

**Nonlinear state-space model:**

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1})$$

$$y_k = h(x_k, u_k, v_k),$$

where $w_k$ and $v_k$ are independent, Gaussian noise processes with means $\bar{w}$ and $\bar{v}$ and covariance matrices $\Sigma_{\tilde{w}}$ and $\Sigma_{\tilde{v}}$, respectively.

**Definitions:** Let

$$x_k^a = \left[x_k^T,\ w_k^T,\ v_k^T\right]^T, \quad \mathcal{X}_k^a = \left[(\mathcal{X}_k^x)^T,\ (\mathcal{X}_k^w)^T,\ (\mathcal{X}_k^v)^T\right]^T, \quad p = 2 \times \dim(x_k^a).$$

**Initialization:** For $k = 0$, set

$$\hat{x}_0^+ = \mathbb{E}\left[x_0\right] \qquad \Sigma_{\tilde{x},0}^+ = \mathbb{E}\left[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T\right]$$

$$\hat{x}_0^{a,+} = \mathbb{E}\left[x_0^a\right] = \left[(\hat{x}_0^+)^T,\ \bar{w},\ \bar{v}\right]^T \qquad \Sigma_{\tilde{x},0}^{a,+} = \mathbb{E}\left[(x_0^a - \hat{x}_0^{a,+})(x_0^a - \hat{x}_0^{a,+})^T\right]$$

$$= \mathrm{diag}\left(\Sigma_{\tilde{x},0}^+, \Sigma_{\tilde{w}}, \Sigma_{\tilde{v}}\right).$$

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

*State estimate time update:*

$$\mathcal{X}_{k-1}^{a,+} = \left\{\hat{x}_{k-1}^{a,+},\ \hat{x}_{k-1}^{a,+} + \gamma\sqrt{\Sigma_{\tilde{x},k-1}^{a,+}},\ \hat{x}_{k-1}^{a,+} - \gamma\sqrt{\Sigma_{\tilde{x},k-1}^{a,+}}\right\}.$$

$$\mathcal{X}_{k,i}^{x,-} = f(\mathcal{X}_{k-1,i}^{x,+}, u_{k-1}, \mathcal{X}_{k-1,i}^{w,+}).$$

$$\hat{x}_k^- = \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \mathcal{X}_{k,i}^{x,-}.$$

*Error covariance time update:*

$$\Sigma_{\tilde{x},k}^- = \sum_{i=0}^{p} \alpha_i^{(\mathrm{c})} \left(\mathcal{X}_{k,i}^{x,-} - \hat{x}_k^-\right)\left(\mathcal{X}_{k,i}^{x,-} - \hat{x}_k^-\right)^T.$$

*Output estimate:*

$$\mathcal{Y}_{k,i} = h(\mathcal{X}_{k,i}^{x,-}, u_k, \mathcal{X}_{k-1,i}^{v,+}).$$

$$\hat{y}_k = \sum_{i=0}^{p} \alpha_i^{(\mathrm{m})} \mathcal{Y}_{k,i}.$$

**Computation:** (cont):

*Estimator gain matrix:*

$$\Sigma_{\tilde{y},k} = \sum_{i=0}^{p} \alpha_i^{(c)} \big(\mathcal{Y}_{k,i} - \hat{y}_k\big)\big(\mathcal{Y}_{k,i} - \hat{y}_k\big)^T.$$

$$\Sigma_{\tilde{x}\tilde{y},k}^{-} = \sum_{i=0}^{p} \alpha_i^{(c)} \big(\mathcal{X}_{k,i}^{x,-} - \hat{x}_k^{-}\big)\big(\mathcal{Y}_{k,i} - \hat{y}_k\big)^T.$$

$$L_k = \Sigma_{\tilde{x}\tilde{y},k}^{-} \Sigma_{\tilde{y},k}^{-1}.$$

*State estimate meas. update:*

$$\hat{x}_k^{+} = \hat{x}_k^{-} + L_k\big(y_k - \hat{y}_k\big).$$

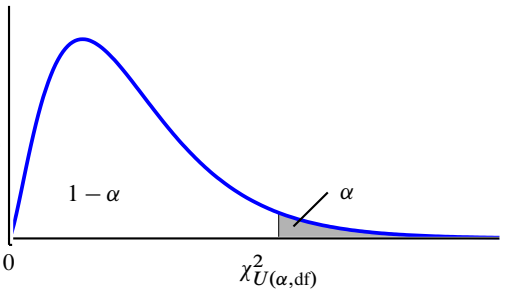
*Error covariance meas. update:*

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

# Appendix: Critical Values of $\chi^2$

- For some deg. of freedom, each entry represents the critical value of $\chi^2$ for a specified upper tail area $\alpha$.




| 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 |