

Estimation of Dynamic Systems

AERO 626



1	Basics of Probability	1
1.1	Random Variables	1
1.2	Expectations of Random Variables	10
1.3	A Few Important Distributions	14
2	Least Squares Estimation	27
2.1	Parameter Estimation	28
2.2	Least Squares: Parameter Estimation	52
2.3	Least Squares: State Estimation	70
2.4	Weighted Least Squares	98
2.5	The Minimum Variance Estimate	106
2.6	Sequential Least Squares	133
2.7	The Batch Processor Revisited	158
2.8	Some Remarks on Least Squares Methods	186
3	Minimum Mean Square Error Estimation	189
3.1	The Kalman Filter	191
3.2	The Extended Kalman Filter	242
3.3	The Discrete Kalman Filter	275

3.4	The Discrete Extended Kalman Filter	282
4	Alterations to the Kalman Filter	298
4.1	Care and Feeding of Your Kalman Filter	298
4.2	The Information Filter	354
4.3	Square Root Filters	366
5	Derivative-Free Kalman Filtering	405
5.1	The Unscented Kalman Filter	412
5.2	The Gauss-Hermite Kalman Filter	452
5.3	The Cubature Kalman Filter	475
5.4	Comments on Derivative-Free Filtering	492
6	Bayesian Filtering	494
6.1	Probabilistic State Space Models	498
6.2	Bayesian Filtering Equations	502
6.3	The Bayesian Kalman Filter	505
6.4	The Gaussian Mixture Filter	524
6.5	Non-Gaussian Extension to the GMF	556

6.6 Nonlinear Extensions to the GMF	579
---	-----

Basics of Probability

1.1 Random Variables

Random variables lie at the heart of many methods in estimation, and we will deal with several key aspects of random variables and the ways in which they are described throughout this course. If we think about estimating the height of a falling body, for instance, we think of the height of the object as being random. We cannot exactly determine the object's height, and so we have to develop ways to characterize where we think the object is and how certain we are in that belief.

A random variable x is, in the simplest terms, a variable that takes on values at random

and realizations of the random variable may be thought of as the outcomes of some *experiment*. There are two types of random variables that are commonly utilized: discrete random variables and continuous random variable. Discrete random variables have a finite set of outcomes, i.e., the roll of a die, whereas continuous random variables do not, i.e., a point in the interval $(0, 1]$. It is also possible to form hybrid random variables by combining discrete and continuous random variables.

Continuous random variables are commonly encountered in estimation problems, but discrete random variables are easier to visualize. As such, we will use discrete random variables to describe a few important concepts in probability.

Consider the previously mentioned experiment that is the roll of a die. The *outcome* or the *event*, E_i , is the value of the die after the roll. For a standard, six-sided die, the possible values of the outcome is a finite (or discrete) list: $E_i \in \{1, 2, 3, 4, 5, 6\}$. The probability of each event must be such that

$$0 \leq \Pr\{E_i\} \leq 1 \quad \text{and} \quad \sum_{i=1}^n \Pr\{E_i\} = 1$$

Each event must have non-negative probability of occurring, and the total probability of any of the events occurring must be equal to one. For the case of the die, if it is a fair die, all outcomes have the same chances of occurring; hence, the probability of each outcome is $\Pr\{E_i\} = 1/6$.

We can also consider multiple events taking place, says events A , B , and C . The joint event of “ A and B and C ” is denoted by ABC , and the probability of this event is $\Pr\{ABC\}$. Similarly, the joint event of “either A or B or C ” is denoted by $A + B + C$, and the probability of this event is $\Pr\{A + B + C\}$.

Events are said to be *mutually independent* if the occurrence of one of the events has no bearing on the occurrence of any of the other of them. If this is the case, then $\Pr\{ABC\} = \Pr\{A\}\Pr\{B\}\Pr\{C\}$. If we roll two dice, the joint events are the pairs of the outcomes of each die, i.e. $E_i \in \{(1, 1), (1, 2), \dots, (6, 6)\}$. Assuming that the dice do not collide in any way or otherwise influence one another, the outcome on each die is independent of the other die; hence, $\Pr\{(1, 1)\} = \dots = \Pr\{(6, 6)\} = 1/36$.

Events are said to be *mutually exclusive* if the occurrence of one event precludes the occurrence of any other event. If this is the case, then $\Pr\{A+B+C\} = \Pr\{A\}+\Pr\{B\}+\Pr\{C\}$.

Considering the roll of a single die, the outcomes are all mutually exclusive; that is, if you roll a 1, you cannot have rolled a 2 or any other number. As such, $\Pr\{1 + 2\} = \Pr\{1\} + \Pr\{2\} = 1/3$.

If two events are *not mutually exclusive*, $\Pr\{A + B\} = \Pr\{A\} + \Pr\{B\} - \Pr\{AB\}$. As an example, if we roll two dice (call them A and B) and ask ourselves if the value of either die is 1, we would find that $\Pr\{A = 1 \text{ or } B = 1\} = \Pr\{A = 1\} + \Pr\{B = 1\} - \Pr\{A = 1 \text{ and } B = 1\} = 1/6 + 1/6 - 1/36 = 11/36$.

Quite clearly, if two events are mutually exclusive, their joint probability is zero.

Another concept that we will find quite useful is that of *conditional probability*. Given events A and B , the probability of A occurring given that B has occurred is defined as

$$\Pr\{A|B\} = \frac{\Pr\{AB\}}{\Pr\{B\}}$$

If A and B are independent, $\Pr\{AB\} = \Pr\{A\}\Pr\{B\}$, and $\Pr\{A|B\} = \Pr\{A\}$. There is no special significance assigned to A and B , meaning that they can be considered as

placeholders such that they can be interchanged to yield

$$\Pr\{B|A\} = \frac{\Pr\{AB\}}{\Pr\{A\}}$$

Solving each of the conditional probabilities for the joint probability of A and B leads to the equality

$$\Pr\{A|B\} \Pr\{B\} = \Pr\{B|A\} \Pr\{A\}$$

which can be rearranged to yield a basic formulation of Bayes' rule that is given by

$$\Pr\{A|B\} = \frac{\Pr\{B|A\} \Pr\{A\}}{\Pr\{B\}}$$

When we move from discrete random variables to continuous random variables, we have to deal with the fact that the space of outcomes moves from a finite list to an infinite one. The consequence is that we cannot talk about the probability of a single point in the space of outcomes. Instead, we use the probability distribution function, $F(x)$, to specify the probability with which different outcomes occur.

The probability distribution function is also called the cumulative distribution function (cdf) and is given by

$$F(x) = \Pr(x \leq x)$$

Note that we sometimes write the cdf as $F_x(x)$, but oftentimes the subscript is dropped for notational simplicity.

An alternative is to make use of the probability density function (pdf), $p(x)$, which is given by

$$p(x) = \frac{dF(x)}{dx}$$

Similar to the cdf, we sometimes write the pdf as $p_x(x)$; again, the subscript is often dropped for simplicity.

The inverse relationship between the distribution and density functions is

$$F(x) = \int_{-\infty}^x p(u)du$$

It is then clear that a characteristic of any probability distribution is

$$F(\infty) = \int_{-\infty}^{\infty} p(u)du = 1$$

since the total probability of the random variable occurring within all possible values must be 1. This is the same as saying that the probability density function must integrate to unity when the limits of integration are taken to be the support of the density.

By applying the definition of the derivative, it follows that

$$p(x) = \frac{dF(x)}{dx} = \lim_{dx \rightarrow 0} \frac{F(x + dx) - F(x)}{dx} = \lim_{dx \rightarrow 0} \frac{\Pr(x \leq x \leq x + dx)}{dx}$$

That is, the interpretation of $p(x)$ is that it is the *density* of probability of the event that x takes on in the vicinity of x . This function is finite if the probability that x takes a value in the infinitesimal interval between x and $x + dx$ is an infinitesimal of order dx . This is usually true of random variables that take values over a continuous range.

So far, we have considered only a single random variable. The simultaneous consideration of more than one random variable is often necessary. In the case of two random

variables, for instance, the probability of the occurrence of pairs of values in a given range is given by the joint cdf

$$F(x, y) = \Pr(x \leq x \text{ and } y \leq y)$$

where x and y are the random variables of interest. The corresponding joint pdf is

$$p(x, y) = \frac{\partial^2 F(x, y)}{\partial x \partial y}$$

The individual probability distribution and density functions for x and y can be found from the joint distribution and density functions. For example

$$F_x(x) = F(x, \infty)$$

$$p_x(x) = \int_{-\infty}^{\infty} p(x, y) dy$$

and similar relationships hold for $F_y(y)$ and $p_y(y)$.

We can revisit our concept of *mutually independent* events for continuous random variables. If x and y are independent, the event $x \leq x$ is independent of the event $y \leq y$; thus,

the probability of the joint occurrence of these events is the product of the probabilities of the individual events, or

$$\begin{aligned}F(x, y) &= \Pr(x \leq x \text{ and } y \leq y) \\&= \Pr(x \leq x) \Pr(y \leq y) \\&= F_x(x)F_y(y)\end{aligned}$$

Similarly, for the joint pdf, we have that

$$\begin{aligned}p(x, y) &= \frac{\partial^2 F(x, y)}{\partial x \partial y} \\&= \frac{\partial^2 F_x(x)F_y(y)}{\partial x \partial y} \\&= \frac{\partial F_x(x)}{\partial x} \frac{\partial F_y(y)}{\partial y} \\&= p_x(x)p_y(y)\end{aligned}$$

1.2 Expectations of Random Variables

The *expectation* of a random variable is defined as the sum of all values that the random variable may take, each weighted by the probability with which the value is taken. For a random variable that takes values over a continuous range, the summation is done by integration. The probability, in the limit as $dx \rightarrow 0$, that x takes a value in the infinitesimal interval of width dx near x is $p(x)dx$. Therefore, the expectation of x , which we denote by $E\{x\}$ is

$$E\{x\} = \int_{-\infty}^{\infty} xp(x)dx$$

This is also called the mean value of x , the mean of the distribution of x , or the first moment of x .

Frequently, it is necessary to find the expectation of a function of a random variable. If y is a function of the random variable x via

$$y = f(x)$$

then y is itself a random variable with a distribution derivable from the distribution of x . The expectation of *any* function of x can be calculated directly from the distribution of x by the integral

$$E\{y\} = E\{f(x)\} = \int_{-\infty}^{\infty} f(x)p(x)dx$$

An important statistical parameter descriptive of the distribution of x is its *mean squared value*. From the definition of the expected value, the expectation of the square of x is

$$E\{x^2\} = \int_{-\infty}^{\infty} x^2 p(x)dx$$

The quantity $E\{x^2\}$ is also called the second (raw) moment of x . The root-mean-squared (rms) value of x is the square root of $E\{x^2\}$.

The variance of a random variable is the mean squared deviation of the random variable from its mean; it is denoted by σ^2 , where

$$\sigma^2 = \int_{-\infty}^{\infty} (x - E\{x\})^2 p(x)dx$$

$$= E\{(x - E\{x\})^2\}$$

Alternatively, by expanding out the square, we see that

$$\begin{aligned}\sigma^2 &= \int_{-\infty}^{\infty} (x - E\{x\})^2 p(x) dx \\&= \int_{-\infty}^{\infty} (x^2 - 2xE\{x\} + E\{x\}^2) p(x) dx \\&= \int_{-\infty}^{\infty} x^2 p(x) dx - 2E\{x\} \int_{-\infty}^{\infty} xp(x) dx + E\{x\}^2 \int_{-\infty}^{\infty} p(x) dx \\&= E\{x^2\} - 2E\{x\}E\{x\} + E\{x\}^2 \\&= E\{x^2\} - E\{x\}^2\end{aligned}$$

This is also called as the second (central) moment of x . The square root of the variance, or σ , is the *standard deviation* of the random variable.

The rms value and the standard deviation are equal only for a zero-mean random variable.

The *covariance* of two random variables is given by the expectation of the product of the deviations of the random variables from their respective means, such that

$$E\{(x - E\{x\})(y - E\{y\})\} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - E\{x\})(y - E\{y\}) p(x, y) dx dy$$

As before, we can expand out the product within the integral to arrive at a different relationship for the covariance

$$E\{(x - E\{x\})(y - E\{y\})\} = E\{xy\} - E\{x\}E\{y\}$$

The term $E\{xy\}$ is the second (raw) moment of x and y .

The covariance, normalized by the standard deviations of x and y , is called the *correlation coefficient*

$$\rho = \frac{E\{xy\} - E\{x\}E\{y\}}{\sigma_x \sigma_y}$$

The correlation coefficient is a measure of the degree of linear dependence between x and y :

- if x and y are independent, $\rho = 0$
- if y is a linear function of x , $\rho = \pm 1$

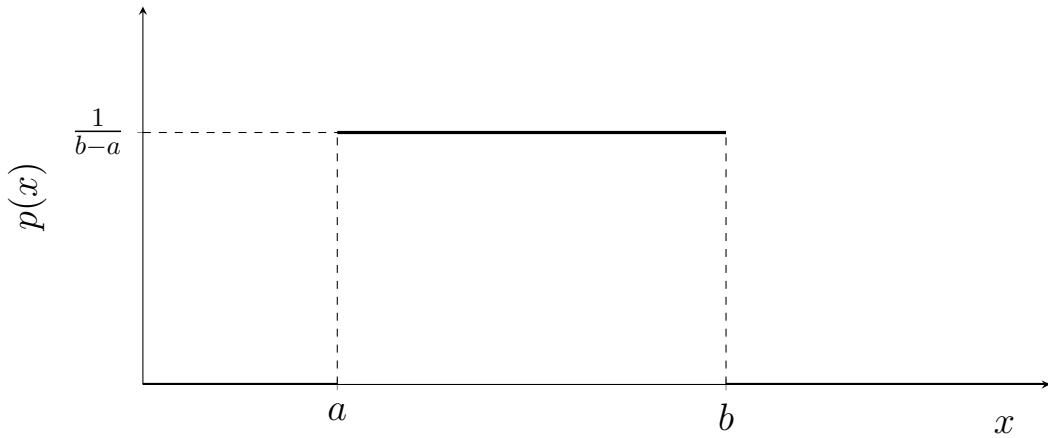
Note: if $\rho = 0$, it is not necessarily true that x and y are independent. It can only be said that x and y are uncorrelated. Independence of the random variables implies that they are uncorrelated, but the converse is not true.

1.3 A Few Important Distributions

There are a plethora of distributions used throughout estimation; herein, we outline a few of the more common ones, especially ones that will show up throughout this course.

The uniform distribution is characterized by a uniform (constant) probability density over some finite interval. The magnitude of the density function in this internal is the reciprocal of the interval width, as required to make the integral of the pdf unity.

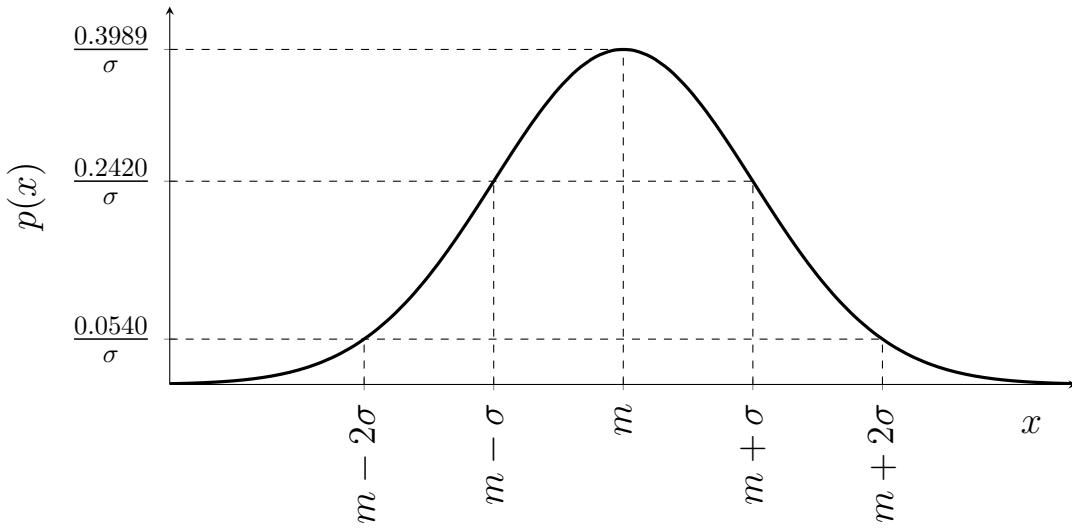
$$p(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$



The normal (Gaussian) probability density function is given by

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-m)^2}{2\sigma^2}\right\}$$

where the two parameters that define the distribution are the mean, m , and the standard deviation, σ . The integral of the function is unity, which is required for this to be a valid probability density function.



- The area within the $\pm 1 \sigma$ interval (centered about the mean) is approximately 0.68.
- The area within the $\pm 2 \sigma$ interval (centered about the mean) is approximately 0.95.

As an interpretation, the probability that a normally distributed random variable resides outside of the $\pm 2 \sigma$ interval is approximately 0.05. It is important to note that these specific values hold *only* for the univariate case.

Oftentimes, we are interested in two or more jointly Gaussian random variables. For the case of n jointly Gaussian random variables, the pdf takes the form

$$p(\mathbf{x}) = |2\pi \mathbf{P}_{xx}|^{-1/2} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \mathbf{m}_x)^T \mathbf{P}_{xx}^{-1} (\mathbf{x} - \mathbf{m}_x) \right\}$$

where

$$\mathbf{x}^T = [x_1 \ x_2 \ \cdots \ x_n]$$

The quantities \mathbf{m}_x and \mathbf{P}_{xx} are, respectively, the mean and covariance of the vector \mathbf{x} :

$$\mathbf{m}_x = \text{E}\{\mathbf{x}\} \quad \text{and} \quad \mathbf{P}_{xx} = \text{E}\{(\mathbf{x} - \mathbf{m}_x)(\mathbf{x} - \mathbf{m}_x)^T\}$$

That is, we adhere to the definitions of the mean and covariance on an element-wise basis.

For the mean, this implies that

$$\mathbf{m}_x = \text{E}\{\mathbf{x}\} = \begin{bmatrix} \text{E}\{x_1\} \\ \text{E}\{x_2\} \\ \vdots \\ \text{E}\{x_n\} \end{bmatrix} = \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_n \end{bmatrix}$$

Similarly, for the covariance, we have

$$\mathbf{P}_{xx} = \begin{bmatrix} E\{(x_1 - m_1)(x_1 - m_1)\} & E\{(x_1 - m_1)(x_2 - m_2)\} & \cdots & E\{(x_1 - m_1)(x_n - m_n)\} \\ E\{(x_2 - m_2)(x_1 - m_1)\} & E\{(x_2 - m_2)(x_2 - m_2)\} & \cdots & E\{(x_2 - m_2)(x_n - m_n)\} \\ \vdots & \vdots & \ddots & \vdots \\ E\{(x_n - m_n)(x_1 - m_1)\} & E\{(x_n - m_n)(x_2 - m_2)\} & \cdots & E\{(x_n - m_n)(x_n - m_n)\} \end{bmatrix}$$

We therefore see that $\mathbf{P}_{xx} = \mathbf{P}_{xx}^T$; the covariance matrix is symmetric.

If we specialize this n -dimensional form of the Gaussian pdf to the case where $n = 1$, it follows that the vector-valued \mathbf{x} becomes the scalar-valued x , with mean and covariance

$$\mathbf{m}_x = m \quad \text{and} \quad \mathbf{P}_{xx} = \sigma^2$$

and that

$$p(x) = |2\pi\sigma^2|^{-1/2} \exp \left\{ -\frac{1}{2}(x - m)\sigma^{-2}(x - m) \right\} = \frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{(x - m)^2}{2\sigma^2} \right\}$$

This is precisely the definition with which we started for the univariate Gaussian pdf.

Another useful reduction is for the case of independent Gaussian random variables. In this case, the i^{th} element of the mean vector remains the same

$$[\boldsymbol{m}_x]_i = m_i = \text{E}\{x_i\}$$

The covariance, however, takes on a greatly simplified form, with the i^{th} row, j^{th} column being given by

$$[\boldsymbol{P}_{xx}]_{ij} = P_{ij} = \text{E}\{(x_i - m_i)(x_j - m_j)\} = \sigma_i^2 \delta_{ij}$$

where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$, otherwise. That is, the covariance matrix is diagonal. Applying the preceding relationships for the mean and covariance, it follows that

$$p(\boldsymbol{x}) = \left[\prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma_i} \right] \exp \left\{ -\frac{1}{2} \sum_{i=1}^n \frac{(x_i - m_i)^2}{\sigma_i^2} \right\} = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma_i} \exp \left\{ -\frac{1}{2} \frac{(x_i - m_i)^2}{\sigma_i^2} \right\}$$

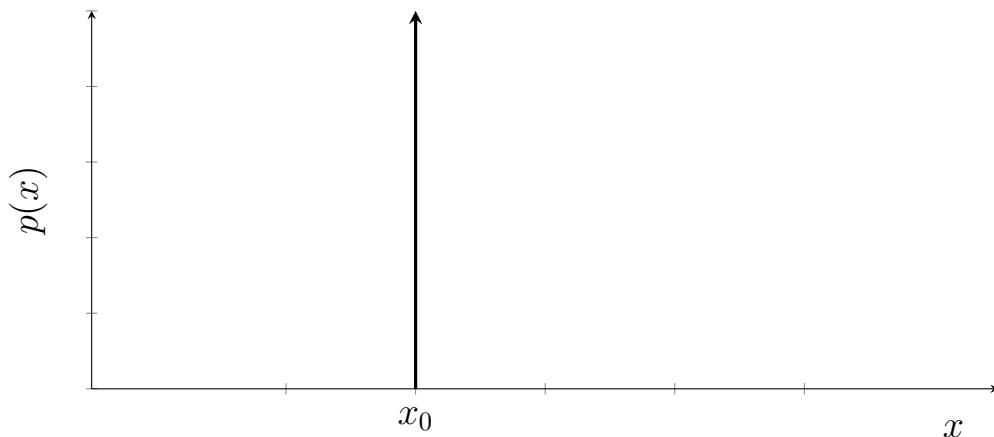
As expected, since the random variables are all independent, the joint pdf reduces down to the product of the individual pdfs.

We might consider what happens to a univariate Gaussian pdf as $\sigma \rightarrow 0^+$. As the

standard deviation decreases, the uncertainty intervals collapse towards the mean, and the peak of the pdf becomes larger. In the limit, the Gaussian collapses to an impulse, where all of the probability mass is centered on one point in the state space. In the language of pdfs, this is the Dirac pdf that is given by

$$p(x) = \delta(x - x_0)$$

where x_0 is the shift parameter of the Dirac pdf.



The *Dirac delta*, $\delta(x - x_0)$, is, in relatively loose terms, a function that is zero everywhere, except at $x = x_0$, where it is infinite in such a way that the integral of the function across the singularity is unity. This interpretation follows from our limiting procedure of the Gaussian, but it is also a fundamental property of the Dirac delta.

Another important property of the Dirac delta is known as the *sifting property*. For a finite-valued function $g(x)$ that is continuous at $x = x_0$,

$$\int_{-\infty}^{\infty} g(x)\delta(x - x_0)dx = g(x_0)$$

In the strict sense, the Dirac pdf does not represent a uncertainty outcome over a continuous space; instead, it best represents discrete outcomes represented in the continuous space. Nevertheless, we will find uses for the Dirac pdf for representing uncertainty of continuous random variables.

As a recap, we have introduced the uniform, Gaussian, and Dirac pdfs. Each one of these pdfs is parameterized by a finite collection of information. As such, we often write these pdfs in the form $p(x; \theta)$, where θ denotes the parameters used to define the pdf. For the uniform pdf, the parameters are the interval on which the pdf is non-zero, such

that

$$p(x) = p_u(x; a_x, b_x)$$

For the Gaussian pdf, we use the mean and covariance, leading to

$$p(\mathbf{x}) = p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx})$$

which is easily reduced to the univariate case. The Dirac pdf is characterized by the shift parameter, such that

$$p(\mathbf{x}) = p_\delta(\mathbf{x}; \mathbf{x}_0)$$

which is also easily reduced to the univariate case.

An interesting extension of the pdfs that we have covered (and of pdfs that we have not covered) is that of finite mixtures. A finite mixture is formed by taking a weighted sum of pdfs of the form

$$p(\mathbf{x}) = \sum_{\ell=1}^L w^{(\ell)} p^{(\ell)}(\mathbf{x})$$

where L is the number of mixture elements, or components, $w^{(\ell)}$ is the weight of the ℓ^{th} component, and $p^{(\ell)}(\mathbf{x})$ is the ℓ^{th} pdf that forms the mixture. The requirement that $p(\mathbf{x})$ is a valid pdf coupled with the assumption that each $p^{(\ell)}(\mathbf{x})$ is a valid pdf leads to the requirement that

$$\sum_{\ell=1}^L w^{(\ell)} = 1$$

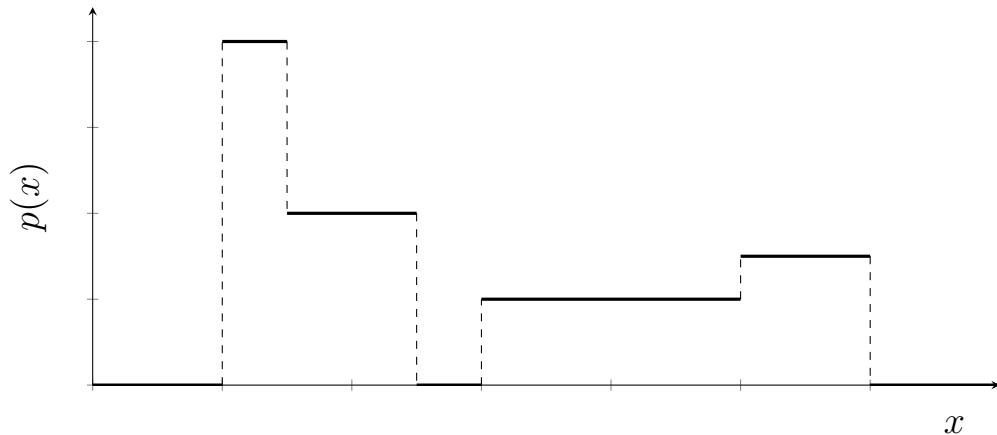
Additionally, to ensure that the overall pdf remains non-negative, each of the weights must be non-negative. Clearly, these requirements also imply that each weight is bounded above by one.

Most commonly, the components are all of the same type of pdf with different parameters, i.e.,

$$p(\mathbf{x}) = \sum_{\ell=1}^L w^{(\ell)} p(\mathbf{x}; \boldsymbol{\theta}^{(\ell)})$$

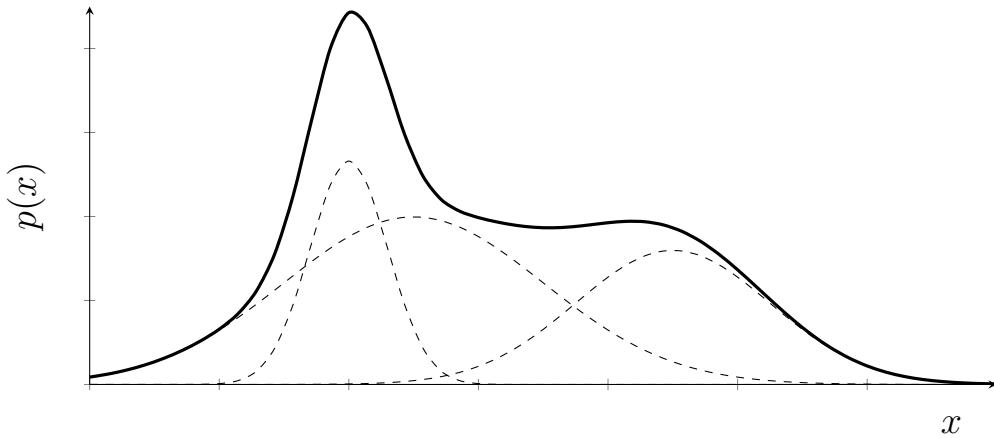
For instance, one can formulate a uniform mixture (UM) as

$$p(x) = \sum_{\ell=1}^L w^{(\ell)} p_u(x; a_x^{(\ell)}, b_x^{(\ell)})$$



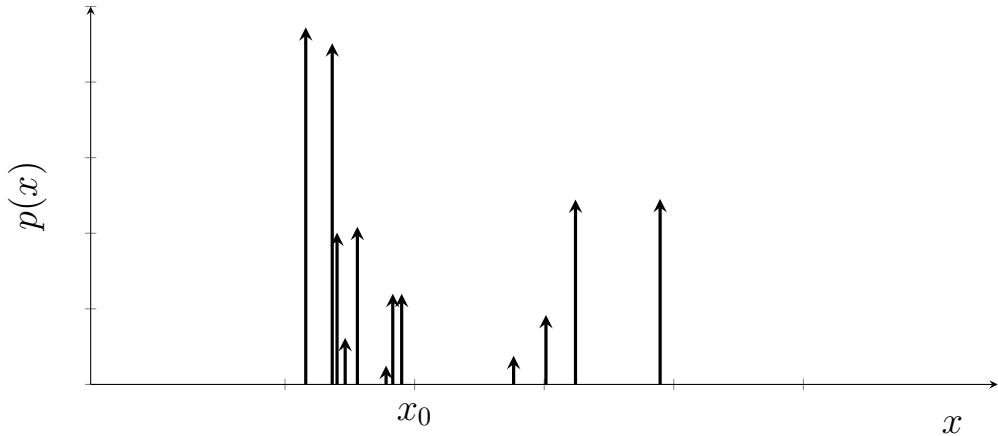
a Gaussian mixture (GM) as

$$p(\mathbf{x}) = \sum_{\ell=1}^L w^{(\ell)} p_g(\mathbf{x}; \mathbf{m}_x^{(\ell)}, \mathbf{P}_{xx}^{(\ell)})$$



and a Dirac mixture (DM) as

$$p(\mathbf{x}) = \sum_{\ell=1}^L w^{(\ell)} p_\delta(\mathbf{x}; \mathbf{x}_0^{(\ell)})$$



2

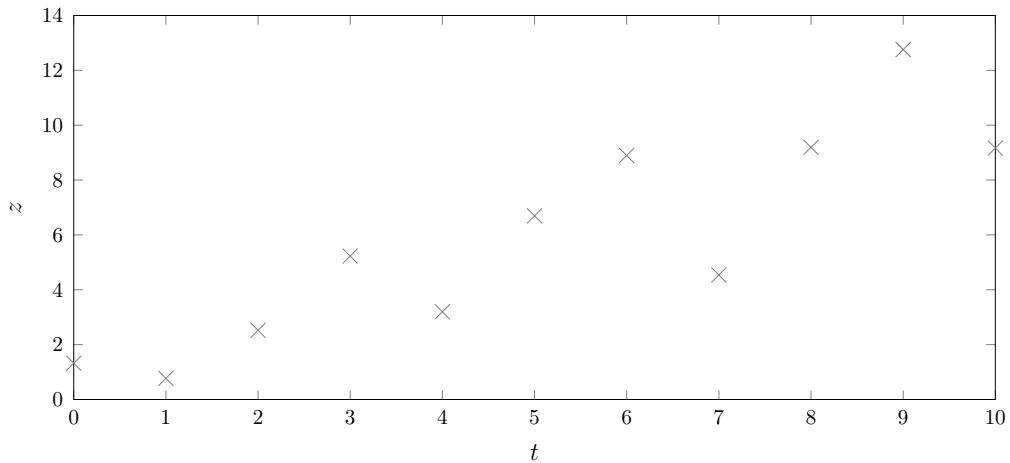
Least Squares Estimation

Suppose you are interested in the time it takes to travel from one point to another, say from home to campus or vice versa. Suppose that each day, you have knowledge (by some means) of the number of cars on the road. This is your independent variable. Each day, you measure the duration of your travel. These are your observations, which are dependent upon the number of cars on the road.

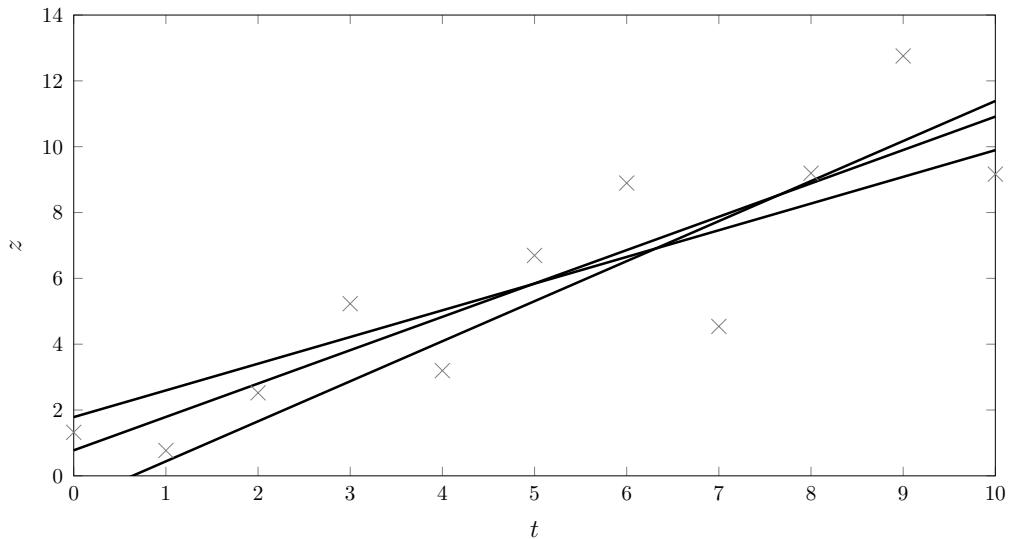
The question is then, if you had prior knowledge of the number of cars on the road on some day, could you estimate the time it would take you to get to campus (or, more importantly, to get home)? This is a parameter estimation problem that we can answer using the method of least squares.

2.1 Parameter Estimation

Given observations z_i of “something” at “times” t_i for $i \in \{1, 2, \dots, m\}$, it is desired to fit a model, h , to the data.



For instance, if it is desired to fit a line to the data, then the model would be $h = a + bt$, such that the model-predicted observation is $h_i = a + bt_i$.

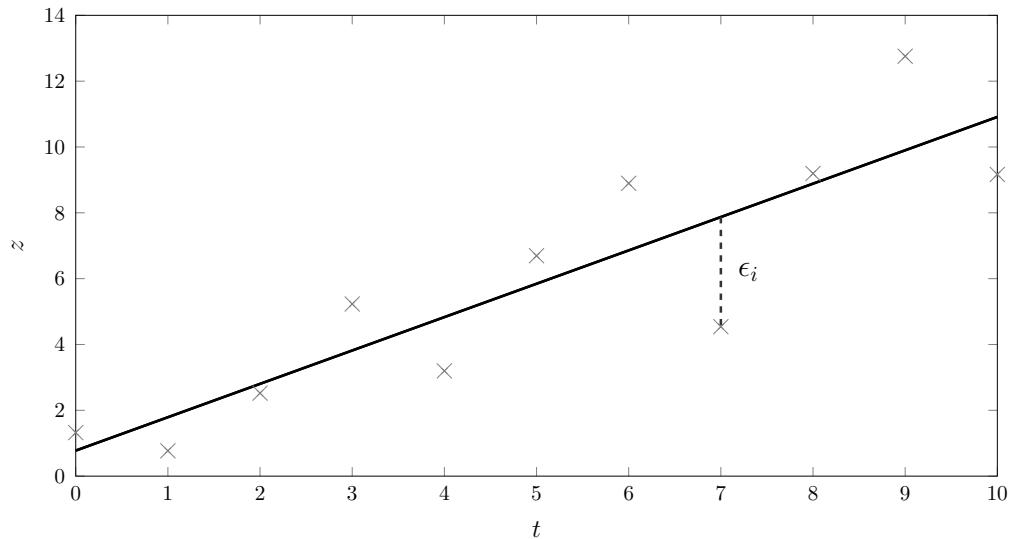


The objective then becomes to determine a method by which the parameters of the model, which are, in the case of fitting a line, the numbers a and b , can be selected.

The general approach is to choose the model parameters such that some performance index is minimized. What should the performance index be?

The performance index should measure the cumulative error between the observations that were taken (e.g. z_i taken at time t_i) and the values predicted by the model (e.g., $h_i = a + bt_i$).

Define this difference between the actual and predicted observations to be the residual, $\epsilon_i = z_i - h_i$.



If the performance index is taken to be the sum of the errors, there could be a model such that the i^{th} residual is opposite in sign and equal in magnitude to the j^{th} residual causing no change in the performance index.

This does not yield a desirable performance index since observations can effectively nullify one another.

Instead, let's consider the performance index to be the sum of the squares of the residuals

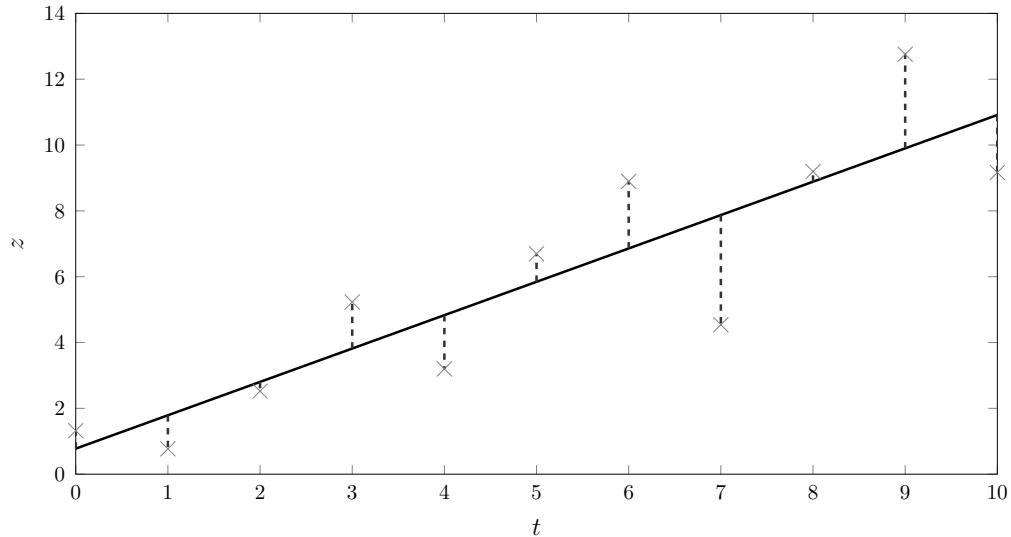
$$J = \sum_{i=1}^m \epsilon_i^2$$

Note that sometimes a factor of $1/2$ is used to scale the performance index. This is only a matter of convenience, and it will not influence the final result.

For m measurements and the line-fitting problem, choosing the performance index in

this manner yields

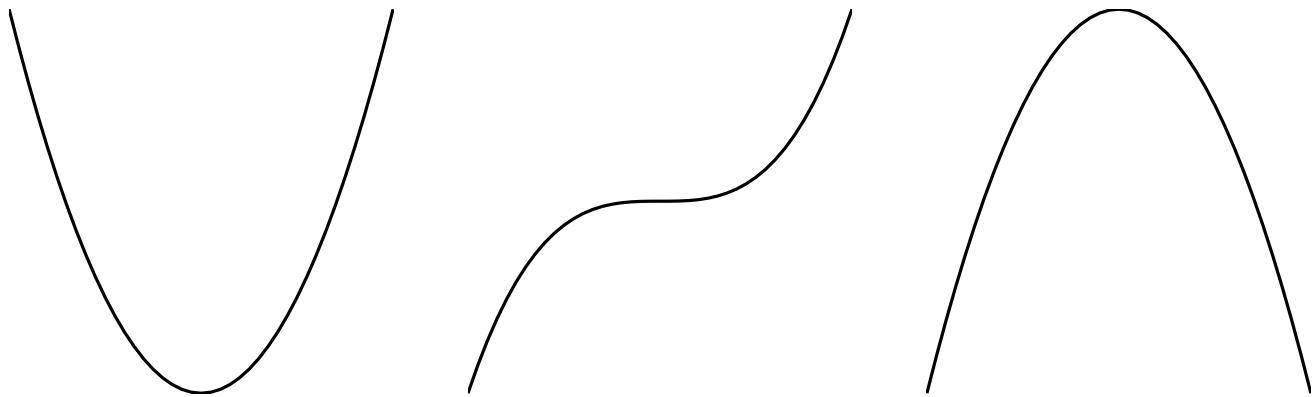
$$J(a, b) = \sum_{i=1}^m \epsilon_i^2 = \sum_{i=1}^m [z_i - (a + bt_i)]^2$$



For each choice of (a, b) , i.e., for each choice of the model, J can have a different value.

We want to choose the model that minimizes the performance index, which yields a parameter optimization problem. This leads to the classical process/conditions:

- Set the first derivatives equal to zero, and solve for the parameters.
- If the matrix of second derivatives is positive definite at the solution from the previous step, this is a minimum.



We call the parameters of the model that minimize the performance index the least squares estimate. For the line-fitting problem, these are (\hat{a}, \hat{b}) .

Let's take a look at where the conditions for a minimum come from.

To do so, consider a generic one-dimensional problem where we want to find the point(s) where some scalar function $\phi(x)$ takes on minimum values. That is, we want to find a point x at which the performance index

$$J = \phi(x)$$

takes on a minimal value.

By definition, a relative minimal point is a point x that satisfies the condition

$$\Delta J = \phi(x_*) - \phi(x) > 0$$

for all admissible comparison points x_* in a small, but finite, neighborhood of x ; that is

$$0 < |x_* - x| < \epsilon$$

This can be rewritten as

$$x_* = x + dx, \quad 0 < |dx| < \epsilon$$

where dx is a small, but finite, displacement that can be positive *or negative*. That is, the admissible comparison point can exist to the left and to the right of x .

The condition for a minimum becomes

$$\Delta J = \phi(x + dx) - \phi(x) > 0$$

regardless of the choice of the admissible comparison point (regardless of the choice of dx).

By expanding the equation for ΔJ in a Taylor series, it follows that

$$\Delta J = dJ + \frac{1}{2!}d^2J + \dots > 0$$

where

$$dJ = \phi_x(x)dx$$

$$d^2J = \phi_{xx}(x)dx^2$$

The first and second differentials are obtained by taking differentials and noting that the differential of an independent differential (in this case, dx) is zero.

If $\phi(x)$ and all of its derivatives, $\phi^{(k)}(x)$, are continuous, the Taylor series is a descending series. This means that as long as the function and its derivatives are finite, a dx can be found such that each term in the series is smaller, in fact negligibly smaller, than the previous term.

This is demonstrated by forming the ratio of two consecutive terms in the series:

$$R = \frac{\frac{1}{(k+1)!} \phi^{(k+1)}(x) dx^{k+1}}{\frac{1}{k!} \phi^{(k)}(x) dx^k} = \frac{1}{k+1} \frac{\phi^{(k+1)}(x)}{\phi^{(k)}(x)} dx$$

Therefore, if all of the derivatives of ϕ are continuous, they are finite, and R is on the order of dx . Thus, R tends to zero as dx tends to zero, which indicates that the Taylor series is a descending series.

We will now show that ϕ_x must vanish at a minimal point.

Assume that x is a minimal point but that $\phi_x(x) \neq 0$. If $\phi_x(x) \neq 0$, then the first

differential dJ is the dominating term in the Taylor series expansion, so that

$$\Delta J \approx dJ = \phi_x(x)dx$$

This must be positive regardless of the choice of $dx \neq 0$.

At x , $\phi_x(x)$ has a specific non-zero value, i.e., it is either positive or negative.

Assume that $\phi_x(x) > 0$.

If $dx > 0$, then $dJ > 0$; if $dx < 0$, then $dJ < 0$.

Since ΔJ must be positive at a minimal point for all $dx \neq 0$, x cannot be a minimal point.

This is a contradiction, and it must therefore be that

$$\phi_x(x) = 0$$

at a minimal point.

This first differential condition is a *necessary* condition for a minimum. The points that satisfy the condition $\phi_x(x) = 0$ are called optimal points. They can be minima, maxima, or saddle points.

To classify the type of point, we must look at the second differential condition.

Assume that x is a solution of the first differential condition.

Since the first differential vanishes, the value of ΔJ can be approximated as

$$\Delta J \approx \frac{1}{2!} d^2 J = \frac{1}{2!} \phi_{xx}(x) dx^2$$

It is apparent that if

$$\phi_{xx}(x) > 0$$

then ΔJ is positive for all $dx \neq 0$ and that x is a minimal point. This is a *sufficient* condition.

It is still possible, however, for $\phi_{xx}(x)$ to vanish and for a minimal point to still exist (consider $\phi(x) = x^4$). Therefore, a *necessary* condition is given by

$$\phi_{xx}(x) \geq 0$$

If the second derivative vanishes, we must examine higher-order differentials to determine the nature of the optimal point.

This is beyond what we need, but for completeness

$$\phi^{(3)}(x) \neq 0 \quad \Rightarrow \quad \text{saddle point}$$

$$\phi^{(3)}(x) = 0 \quad \text{and} \quad \phi^{(4)}(x) > 0 \quad \Rightarrow \quad \text{minimum}$$

Examples are, respectively, $\phi(x) = x^3$ and $\phi(x) = x^4$.

This analysis can be carried out for arbitrarily high order differentials, but to summarize

1. optimal points satisfy the necessary condition $\phi_x(x) = 0$
2. optimal points that satisfy the sufficient condition $\phi_{xx}(x) > 0$ are minimal points

There are no conditions that are both necessary and sufficient.

Now, let's go back to our line-fitting example.

Recall that we have m measurements and that we chose the performance index to be a least-squares cost function of the form

$$J(a, b) = \sum_{i=1}^m [z_i - (a + bt_i)]^2$$

where a and b are parameters of the model that we want to find so that the model best fits the data.

For each choice of (a, b) , i.e. for each choice of the model, J can have a different value.

We want to choose the model that minimizes the performance index.

Suppose that the performance index is evaluated at (\hat{a}, \hat{b}) , which is assumed to be a minimum.

Then any other value of J at $(\hat{a} + da, \hat{b} + db)$ should be larger than $J(\hat{a}, \hat{b})$; otherwise, the values of \hat{a} and \hat{b} do not lead to a minimum of the performance index.

Let's consider a Taylor series expansion of the performance index about both $a = \hat{a} + da$ and $b = \hat{b} + db$ simultaneously, which produces

$$\begin{aligned} J(a, b) &= J(\hat{a}, \hat{b}) + \left[\frac{\partial J}{\partial a} \Big|_{(\cdot)=\hat{\cdot}} \right] da + \left[\frac{\partial J}{\partial b} \Big|_{(\cdot)=\hat{\cdot}} \right] db + \frac{1}{2} \left[\frac{\partial^2 J}{\partial a \partial a} \Big|_{(\cdot)=\hat{\cdot}} \right] (da)^2 \\ &\quad + \frac{1}{2} \left[\frac{\partial^2 J}{\partial b \partial b} \Big|_{(\cdot)=\hat{\cdot}} \right] (db)^2 + \left[\frac{\partial^2 J}{\partial a \partial b} \Big|_{(\cdot)=\hat{\cdot}} \right] da db + \text{H.O.T.} \end{aligned}$$

We can also represent this using matrices (with a bit a shorthand notation) as

$$\begin{aligned} J(a, b) &= J(\hat{a}, \hat{b}) + \begin{bmatrix} \partial J / \partial a \\ \partial J / \partial b \end{bmatrix}^T \begin{bmatrix} da \\ db \end{bmatrix} \\ &\quad + \frac{1}{2} \begin{bmatrix} da \\ db \end{bmatrix}^T \begin{bmatrix} \partial^2 J / \partial a^2 & \partial^2 J / \partial a \partial b \\ \partial^2 J / \partial a \partial b & \partial^2 J / \partial b^2 \end{bmatrix} \begin{bmatrix} da \\ db \end{bmatrix} + \text{H.O.T.} \end{aligned}$$

$$= J(\hat{a}, \hat{b}) + \mathbf{J}' \begin{bmatrix} da \\ db \end{bmatrix} + \frac{1}{2} \begin{bmatrix} da \\ db \end{bmatrix}^T \mathbf{J}'' \begin{bmatrix} da \\ db \end{bmatrix} + \text{H.O.T.}$$

where \mathbf{J}' is the (row) vector of first derivatives and \mathbf{J}'' is the (symmetric) matrix of second derivatives.

Recall that the least-squares performance index for the line-fitting problem is given by

$$J = \sum_{i=1}^m [z_i - (a + bt_i)]^2$$

We can use this to compute all of the required first and second derivatives.

We begin by considering only the first derivatives. We will use the first derivatives to find a candidate minimum, and then we will return to the second derivatives to determine if the candidate minimum satisfies the condition for a minimum.

The first derivative of the performance index with respect to the parameter a is

$$\begin{aligned}\frac{\partial J}{\partial a} &= \frac{\partial}{\partial a} \sum_{i=1}^m [z_i - (a + bt_i)]^2 = \sum_{i=1}^m 2[z_i - (a + bt_i)](-1) \\ &= -2 \left[\underbrace{\sum_{i=1}^m z_i}_{e_1} - a \underbrace{\sum_{i=1}^m 1}_m - b \underbrace{\sum_{i=1}^m t_i}_{\beta} \right] = -2[e_1 - ma - \beta b]\end{aligned}$$

Likewise, for the parameter b , it follows that

$$\begin{aligned}\frac{\partial J}{\partial b} &= \frac{\partial}{\partial b} \sum_{i=1}^m [z_i - (a + bt_i)]^2 = \sum_{i=1}^m 2[z_i - (a + bt_i)](-t_i) \\ &= -2 \left[\underbrace{\sum_{i=1}^m z_i t_i}_{e_2} - a \underbrace{\sum_{i=1}^m t_i}_\beta - b \underbrace{\sum_{i=1}^m t_i^2}_\alpha \right] = -2[e_2 - \beta a - \alpha b]\end{aligned}$$

Now, to apply the first differential condition, we simultaneously set the preceding derivatives equal to zero, which gives

$$0 = -2[e_1 - ma - \beta b]$$

$$0 = -2[e_2 - \beta a - \alpha b]$$

This set of two equations can be expressed as a set of linear equations via

$$\begin{bmatrix} m & \beta \\ \beta & \alpha \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$$

We can (try to) solve this linear system to obtain the least-squares estimate of the model parameters, \hat{a} and \hat{b} . Note that a solution exists only if $ma - \beta^2 \neq 0$.

Assuming that we have solved for the parameters, we need to determine if they represent a solution that minimizes the performance index.

Therefore, we need to compute the matrix of second derivatives and check the second differential condition, i.e. that $\mathbf{J}'' > \mathbf{0}$.

Note: we must compute the second derivative from the first derivative *before* we apply the first differential condition.

The elements of the matrix of second derivatives are

$$\frac{\partial^2 J}{\partial a \partial a} = \frac{\partial}{\partial a} \{-2[e_1 - ma - \beta b]\} = 2m$$

$$\frac{\partial^2 J}{\partial b \partial b} = \frac{\partial}{\partial b} \{-2[e_2 - \beta a - \alpha b]\} = 2\alpha$$

$$\frac{\partial^2 J}{\partial a \partial b} = \frac{\partial}{\partial b} \{-2[e_1 - ma - \beta b]\} = 2\beta$$

$$\frac{\partial^2 J}{\partial b \partial a} = \frac{\partial}{\partial a} \{-2[e_2 - \beta a - \alpha b]\} = 2\beta$$

Assembling these gives the matrix of second derivatives as

$$\mathbf{J}'' = 2 \begin{bmatrix} m & \beta \\ \beta & \alpha \end{bmatrix}$$

Is \mathbf{J}'' positive definite? How can we check?

Necessary and sufficient conditions for the second derivative matrix to be positive definite are that the leading principle minors are positive.

These conditions yield

$$|2m| > 0 \quad \text{and} \quad \begin{vmatrix} 2m & 2\beta \\ 2\beta & 2\alpha \end{vmatrix} > 0$$

Therefore, we have a minimum when

$$2m > 0 \quad \text{and} \quad 4(m\alpha - \beta^2) > 0$$

It is worth noting that the second condition above requires that the original linear system be solvable. The first condition holds provided that we took at least one measurement, but this condition cannot be considered by itself.

The question now is when will these conditions hold?

Let's look at a few cases to gain a bit of insight.

1. Case 1: one measurement

From our definitions

$$m = 1, \quad \beta = t_1, \quad \text{and} \quad \alpha = t_1^2$$

Is the matrix of second derivatives positive definite?

$$2m = 2 > 0$$

$$4(m\alpha - \beta^2) = 4(t_1^2 - t_1^2) = 0 \not> 0$$

No! The matrix of second derivatives is not positive definite.

Can we solve the system of equations?

$$\begin{bmatrix} m & \beta \\ \beta & \alpha \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} 1 & t_1 \\ t_1 & t_1^2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} z_1 \\ z_1 t_1 \end{bmatrix}$$

No! This is a system of two linearly dependent equations.

What is happening physically? An infinite number of lines pass through a single

point. All of these lines have $J = 0$, so there is no best line to fit through one point.

This is an under-determined system. We do not have enough linearly independent observations to determine a least-squares estimate of the model.

2. Case 2: two measurements

From our definitions

$$m = 2, \quad \beta = t_1 + t_2, \quad \text{and} \quad \alpha = t_1^2 + t_2^2$$

Is the matrix of second derivatives positive definite?

$$2m = 4 > 0$$

$$4(m\alpha - \beta^2) = 4(t_1 - t_2)^2 > 0$$

Yes! The matrix of second derivatives is positive definite (so long as the measurements are at different times).

We should be able to solve the system of equations...

$$\begin{bmatrix} m & \beta \\ \beta & \alpha \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} 2 & t_1 + t_2 \\ t_1 + t_2 & t_1^2 + t_2^2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} z_1 + z_2 \\ z_1 t_1 + z_2 t_2 \end{bmatrix}$$

Solving the system yields

$$\hat{a} = \frac{z_1 t_2 - z_2 t_1}{t_2 - t_1} \quad \text{intercept}$$

$$\hat{b} = \frac{z_2 - z_1}{t_2 - t_1} \quad \text{slope}$$

Since the matrix of second derivatives is positive definite, this solution is a minimum. It is the least-squares solution.

Additionally, this solution passes exactly through both measurements, so the performance index is zero.

Since the solution passes exactly through both measurements, the residuals (ϵ_1 and ϵ_2) are both zero (you should try to show this).

3. Case 3: more than two measurements

We won't go into the details here, but provided that the measurements are linearly independent, we now have an over-determined system (more measurements than model parameters).

This is precisely what least-squares is meant for, but the details get a bit tedious in this notation.

In general, the least-squares solution is

$$\begin{bmatrix} \hat{a} \\ \hat{b} \end{bmatrix} = \begin{bmatrix} m & \beta \\ \beta & \alpha \end{bmatrix}^{-1} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$$

where

$$\beta = \sum_{i=1}^m t_i, \quad \alpha = \sum_{i=1}^m t_i^2, \quad e_1 = \sum_{i=1}^m z_i, \quad \text{and} \quad e_2 = \sum_{i=1}^m z_i t_i$$

You should also check the matrix of second derivatives to ensure that it is positive definite.

One line can't pass through all of the observations in general, so all of the residuals will not be zero as was the case in the two measurement scenario.

Consider, for example, the task of fitting a line to a set of times and observations (which is in fact the data from the example presented in Section 2.2) given by

$t_i :$	0	1	2	3	4	5	6	7	8	9	10
$z_i :$	1.3219	0.7698	2.5220	5.2300	3.1972	6.6962	8.8959	4.5403	9.1943	12.7572	9.1665

Applying the previously discussed techniques, it can be shown that

$$\beta = 55, \quad \alpha = 385, \quad e_1 = 64.2911, \quad \text{and} \quad e_2 = 432.9642$$

Solving the linear system for the least-squares estimate yields

$$\hat{a} = 0.7761 \quad \hat{b} = 1.0137$$

and it can be shown that this estimate minimizes the least-squares performance index.

2.2 Least Squares: Parameter Estimation

While the preceding method gives us a least-squares estimate, the notation is cumbersome (to say the least).

In order to make least squares a viable method, we need to make it more generally applicable. To accomplish this, we now set about re-deriving the least-squares method in matrix-vector notation.

Assume that q observations, $\mathbf{z}_i \in \mathbb{R}^p$, are acquired at times t_i . Note that p is the dimension of a single vector observation.

The first step is to identify the parameters we wish to estimate, which will denote by \mathbf{x} . The set of parameters is taken to be n -dimensional, or $\mathbf{x} \in \mathbb{R}^n$. As an example, the parameter set for the line-fitting problem is $\mathbf{x}^T = [a \ b]$, which has dimension $n = 2$.

The next step is to express the model as linear combinations of the parameters, such

that

$$\mathbf{h}_i = \mathbf{H}_i \mathbf{x}$$

We have extended our model to a vector instead of a scalar, such that $\mathbf{h}_i \in \mathbb{R}^p$. Given the dimensions of \mathbf{h}_i and \mathbf{x} , it follows that the model matrix, \mathbf{H}_i , is a $p \times n$ matrix.

As before, we define the residual to be the difference between the actual observations $\mathbf{z}_i \in \mathbb{R}^p$ and the model-predicted values \mathbf{h}_i , which gives

$$\boldsymbol{\epsilon}_i = \mathbf{z}_i - \mathbf{H}_i \mathbf{x}$$

The least-squares performance index is now the sum of the squares of the residuals:

$$J = \sum_{i=1}^q \boldsymbol{\epsilon}_i^T \boldsymbol{\epsilon}_i$$

Note that, since we are dealing with vector observations, we cannot simply square the individual residuals, but the inner product produces the equivalent formulation since each scalar element of the residual is squared.

Substituting for the residual into the performance index gives

$$J = \sum_{i=1}^q [\mathbf{z}_i - \mathbf{H}_i \mathbf{x}]^T [\mathbf{z}_i - \mathbf{H}_i \mathbf{x}]$$

In order to proceed, we want to re-express the performance index in a more convenient fashion. To that end, define a concatenated measurement, a concatenated model matrix, and a concatenated residual as

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_q \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \\ \vdots \\ \mathbf{H}_q \end{bmatrix}, \quad \text{and} \quad \boldsymbol{\epsilon} = \begin{bmatrix} \boldsymbol{\epsilon}_1 \\ \boldsymbol{\epsilon}_2 \\ \vdots \\ \boldsymbol{\epsilon}_q \end{bmatrix}$$

Note that we can use the concatenated residual to produce the same performance index that we started with, i.e.,

$$J = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} = \boldsymbol{\epsilon}_1^T \boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_2^T \boldsymbol{\epsilon}_2 + \cdots + \boldsymbol{\epsilon}_q^T \boldsymbol{\epsilon}_q = \sum_{i=1}^q \boldsymbol{\epsilon}_i^T \boldsymbol{\epsilon}_i$$

Define $m = pq$, such that $\boldsymbol{\epsilon} \in \mathbb{R}^m$, $\mathbf{z} \in \mathbb{R}^m$, and $\mathbf{H} \in \mathbb{R}^{m \times n}$. Now, substitute for the residual in terms of the observations and the model, to get

$$J = [\mathbf{z} - \mathbf{Hx}]^T [\mathbf{z} - \mathbf{Hx}]$$

At this point, we are ready to apply our conditions for minimizing the performance index; namely, setting the first derivative of the performance index with respect to the parameter set equal to zero and verifying that the second derivative of the performance index with respect to the parameter set is positive definite.

The first derivative is

$$\frac{\partial J}{\partial \mathbf{x}} = -2[\mathbf{z} - \mathbf{Hx}]^T \mathbf{H}$$

Setting this equal to zero yields

$$[\mathbf{z} - \mathbf{Hx}]^T \mathbf{H} = \mathbf{0}^T \quad \rightarrow \quad \mathbf{z}^T \mathbf{H} - \mathbf{x}^T \mathbf{H}^T \mathbf{H} = \mathbf{0}^T$$

Transposing the preceding result gives us

$$\mathbf{H}^T \mathbf{Hx} = \mathbf{H}^T \mathbf{z}$$

Solving for the parameters produces the (potential) least-squares estimate as

$$\hat{\mathbf{x}} = [\mathbf{H}^T \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{z}$$

This result is the solution to the so-called normal equation:

$$\mathbf{H}^T \mathbf{H} \hat{\mathbf{x}} = \mathbf{H}^T \mathbf{z}$$

Whenever we see an inverted matrix, we should always wonder if the inverse exists. We will show that we expect it to exist.

Let's move on to the second derivative. The first thing we do is transpose the result of the first derivative (before we made any manipulations to solve the equation). This gives us

$$\left[\frac{\partial J}{\partial \mathbf{x}} \right]^T = -2 \mathbf{H}^T [\mathbf{z} - \mathbf{Hx}]$$

Now, we can take the derivative of this expression with respect to \mathbf{x} :

$$\frac{\partial}{\partial \mathbf{x}} \left[\frac{\partial J}{\partial \mathbf{x}} \right]^T = 2 \mathbf{H}^T \mathbf{H}$$

It is important to note that this is precisely the matrix we need to invert in order to find the (potential) least-squares estimate, up to a scale factor.

If this matrix is positive definite, we know that it can be inverted and we can solve the system. Additionally, we know that the solution does indeed minimize the performance index, so the solution is also the least-squares estimate.

How do we know that the matrix is positive definite?

First, note that the matrix \mathbf{H} is an $m \times n$ matrix with $m > n$ (in order to have an over-determined system).

Therefore, \mathbf{H} is at most rank n . Provided that we have n linearly independent observations, \mathbf{H} will be rank n ; that is, it is full (column) rank.

For any $\mathbf{A} \in \mathbb{R}^{m \times n}$ that is rank n ($n < m$), then $\mathbf{A}^T \mathbf{A}$ is rank n (full rank) and is therefore positive definite.

Applying the preceding result to the matrix \mathbf{H} means that as long as we have n lin-

early independent observations (where n is the number of parameters we are estimating), a least-squares solution exists.

Let's take another look at the line-fitting problem. For this problem, we had m observations z_i taken at times t_i , and we wanted to fit the model $h_i = a + bt_i$ to this data in the least-squares sense.

Previously, we came up with a solution through a specialized treatment for the line-fitting problem; now, we want to show that our general solution procedure produces the same solution.

The set of parameters that we want to estimate is defined to be $\mathbf{x}^T = [a \ b]$.

Next, we write the model as a linear combination of the parameters, which gives

$$h_i = \begin{bmatrix} 1 & t_i \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \mathbf{H}_i \mathbf{x}$$

Note that this is exactly the same as

$$h_i = a + bt_i$$

The least-squares solution is

$$\hat{\mathbf{x}} = [\mathbf{H}^T \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{z}$$

where

$$\mathbf{H} = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_m \end{bmatrix} \quad \text{and} \quad \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix}$$

Now, form the products $\mathbf{H}^T \mathbf{H}$ and $\mathbf{H}^T \mathbf{z}$:

$$\mathbf{H}^T \mathbf{H} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ t_1 & t_2 & \dots & t_m \end{bmatrix} \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^m 1 & \sum_{i=1}^m t_i \\ \sum_{i=1}^m t_i & \sum_{i=1}^m t_i^2 \end{bmatrix}$$

and

$$\mathbf{H}^T \mathbf{z} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ t_1 & t_2 & \cdots & t_m \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^m z_i \\ \sum_{i=1}^m z_i t_i \end{bmatrix}$$

If we define

$$\beta = \sum_{i=1}^m t_i, \quad \alpha = \sum_{i=1}^m t_i^2, \quad e_1 = \sum_{i=1}^m z_i, \quad \text{and} \quad e_2 = \sum_{i=1}^m z_i t_i$$

then, it follows that the least-squares solution may be expressed as

$$\hat{\mathbf{x}} = \begin{bmatrix} m & \beta \\ \beta & \alpha \end{bmatrix}^{-1} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$$

which is exactly the solution we had found previously.

2.2.1 Example: Line Fitting

Recall the line fitting problem from before, where we fit a line to the table of 11 observations by computing α , β , e_1 , and e_2 . Using these, we were able to find the parameters \hat{a} and \hat{b} to fit a line of the form $\hat{a} + \hat{b}t$ to these data.

We will redo this example, but this time, we will use MATLAB to perform line fitting to the data in a least-squares sense.

Since we will be generating our observations randomly, we will start by setting the random number seed.

```
% Set random seed  
rng(100)
```

Create a vector of times at which you wish to have observations.

```
% Times for observations  
t = (0:1:10)';
```

Determine the number of observations that we will have. There will be one scalar measurement at each time step, so the number of measurements is equal to the number of time steps.

```
% Number of measurements  
m = length(t);
```

Set the true parameters of the line; in this case, we will use $a = 1$ and $b = 1$. It is important to note that this information is not known to the least-squares method that we will use soon.

```
% Set true parameters of the line  
a = 1;  
b = 1;
```

Create a set of nominal measurements from the true line.

```
% Generate data on a line  
y = a + b.*t;
```

Create a set of noisy observations from this line according to a Gaussian distribution with a mean of $m = 0$ and standard deviation of $\sigma = 2$.

```
% Generate noisy observations  
s = 2;  
z = y + s.*randn(m,1);
```

We are now ready to put the least-squares approach into action. Construct the model matrix \mathbf{H} by appending a column vector of ones with the times of the observations.

```
% Form the measurement mapping matrix  
H = [ones(m,1), t];
```

Now we build our least-squares estimate, and we're done.

```
% Compute least-squares line fit  
x = (H' * H) \ (H' * z);
```

Acknowledging that in MATLAB, $\mathbf{x} = [\hat{a} \ \hat{b}]^T$, we can obtain our estimates for the linear fit as

$$\hat{a} = 0.7761 \quad \hat{b} = 1.0137,$$

which is exactly what we found before!

2.2.2 Example: Polynomial Fitting

We can now very easily extend the least-squares solution to more than simple linear fits.

Consider the polynomial model

$$h_i = a_0 + a_1 t_i + \frac{1}{2} a_2 t_i^2 + \frac{1}{6} a_3 t_i^3 + \cdots + \frac{1}{k!} a_k t_i^k$$

where the a_i 's are the parameters to be determined. As such, define the parameter set to be

$$\mathbf{x}^T = [a_0 \ a_1 \ \frac{1}{2}a_2 \ \frac{1}{6}a_3 \ \cdots \ \frac{1}{k!}a_k]$$

The model mapping matrix is then given by

$$\mathbf{H}_i = [1 \ t_i \ t_i^2 \ t_i^3 \ \cdots \ t_i^k]$$

which gives the concatenated mapping matrix as the Vandermonde matrix

$$\mathbf{H} = \begin{bmatrix} 1 & t_1 & t_1^2 & t_1^3 & \cdots & t_1^k \\ 1 & t_2 & t_2^2 & t_2^3 & \cdots & t_2^k \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_m & t_m^2 & t_m^3 & \cdots & t_m^k \end{bmatrix}$$

From here, we can simply apply the least-squares solution to estimate the model parameters:

$$\hat{\mathbf{x}} = [\mathbf{H}^T \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{z}$$

Note that, to retain an over-determined system, we must ensure that the number of parameters in the model is smaller than the number of measurements.

This is precisely what the MATLAB routine `polyfit` does.

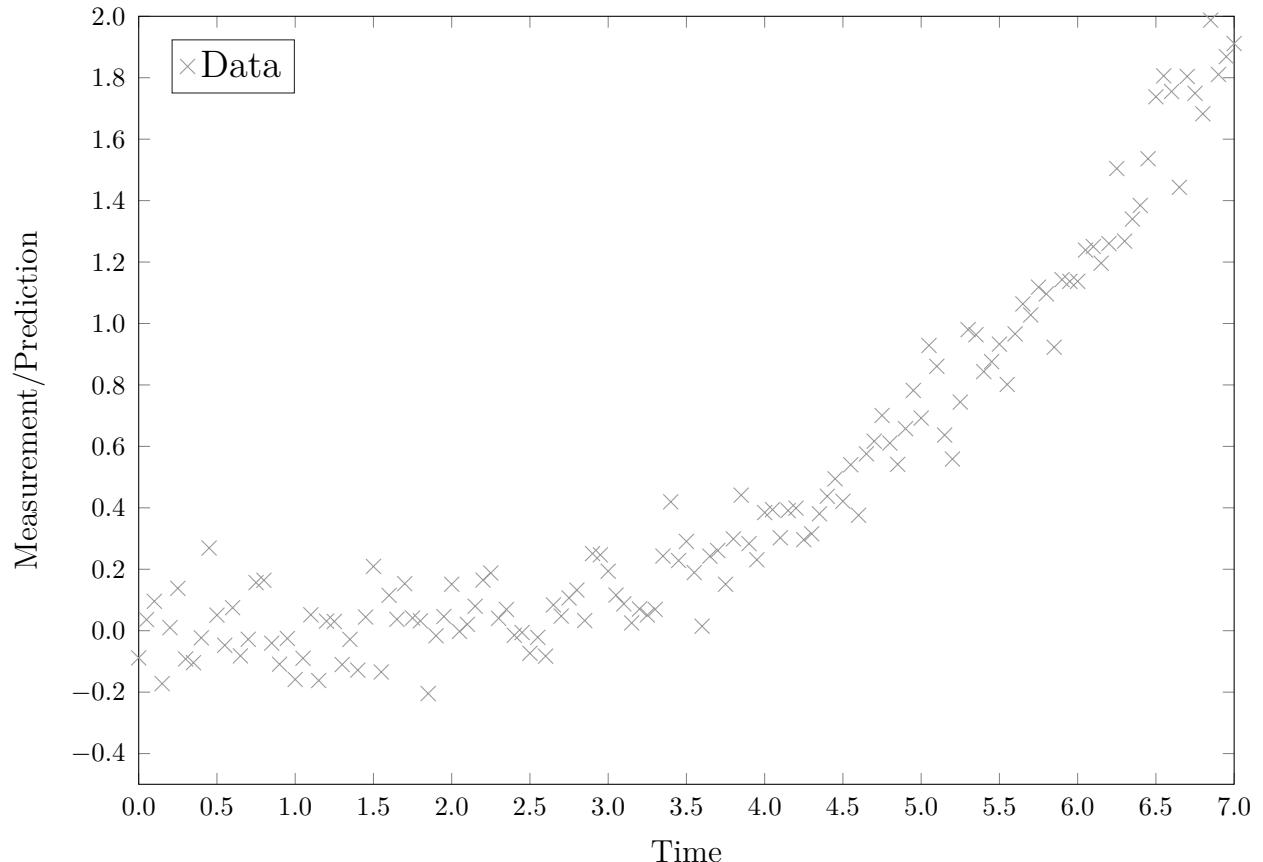
Let's look at an example:

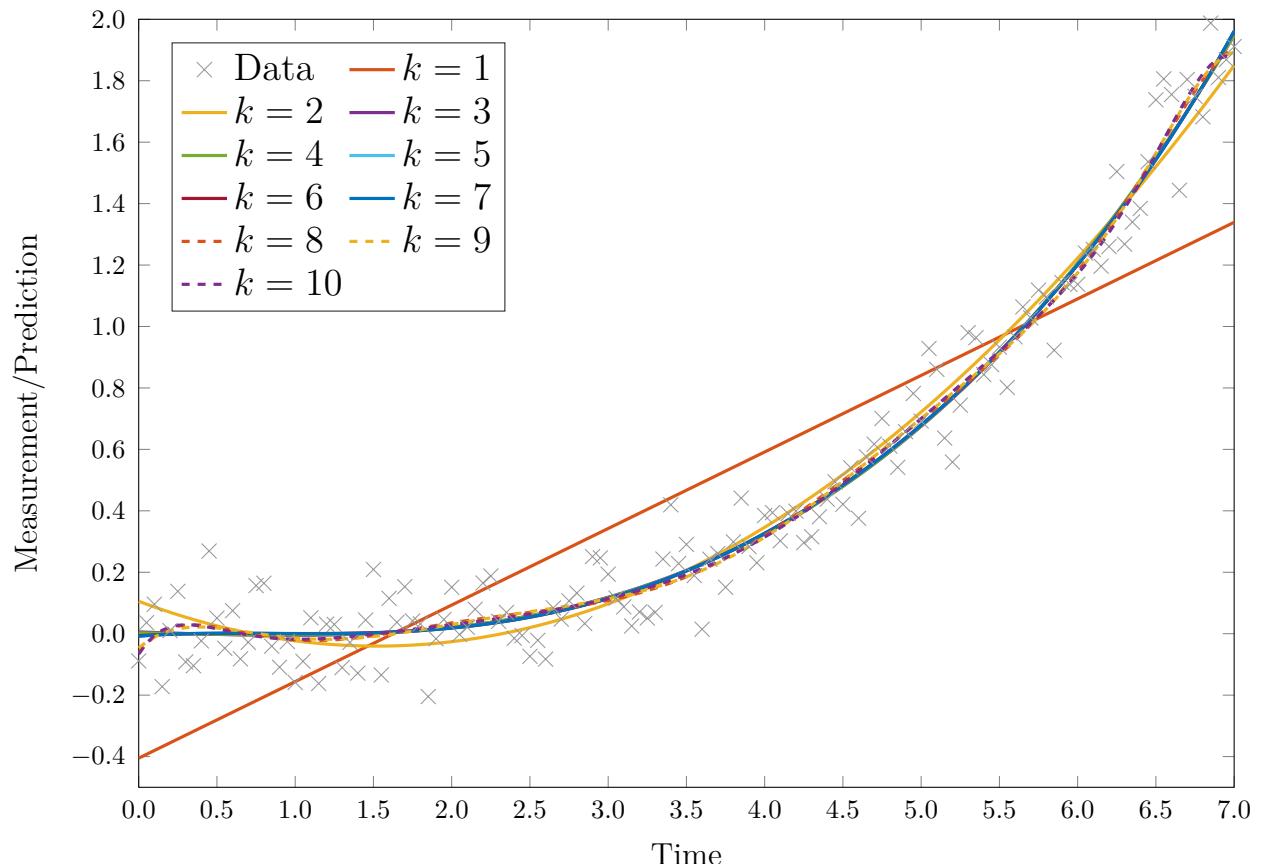
- The true data are generated from a fourth-degree polynomial.
- Measurement errors are added to the data.
- We use the least-squares polynomial fitting method to fit degree k polynomials.
- The quality of the fit is analyzed by computing the post-fit residuals as

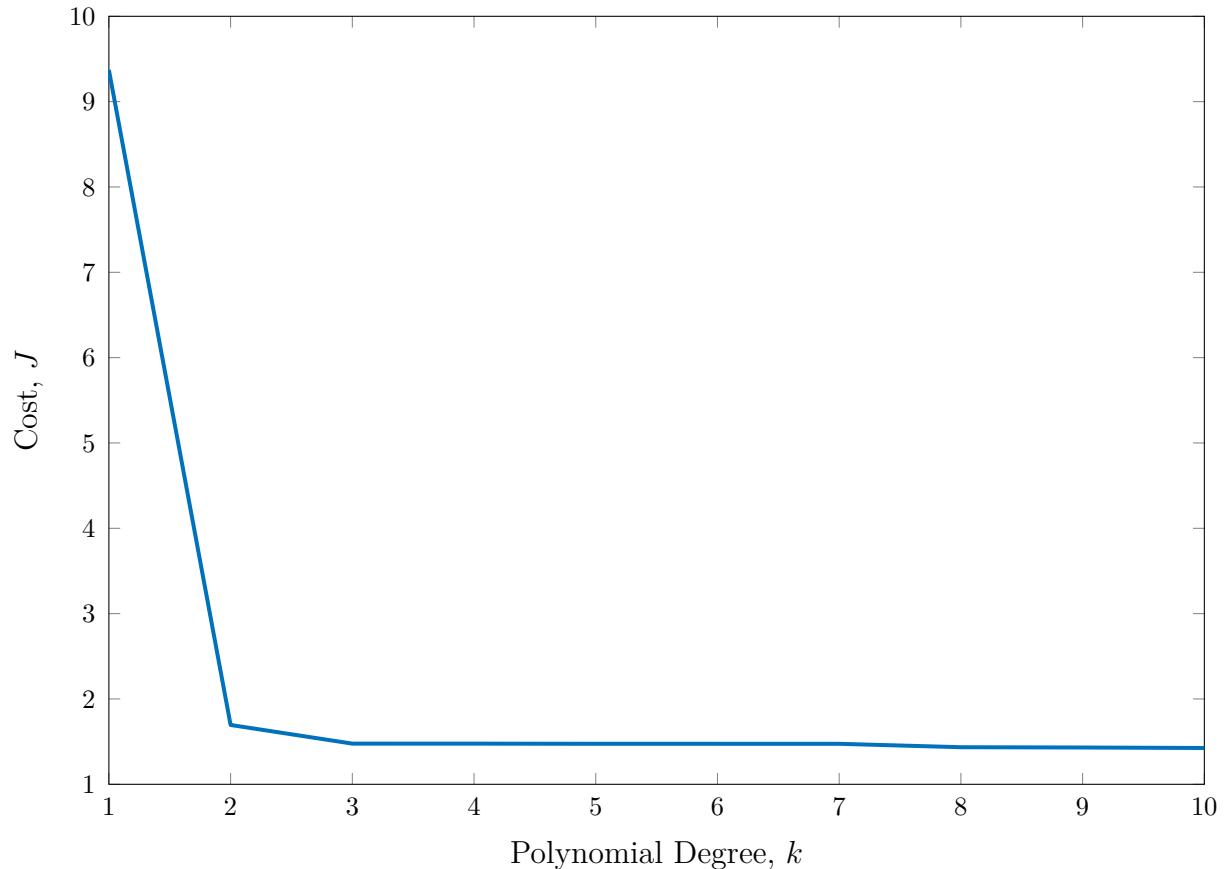
$$\hat{\epsilon}_i = z_i - \mathbf{H}_i \hat{\mathbf{x}}$$

and then finding the cost via

$$J = \sum_{i=1}^m \hat{\epsilon}_i^T \hat{\epsilon}_i$$







2.3 Least Squares: State Estimation

What if the set of parameters we want to estimate can change in time? We would need to include the dynamical behavior of the variables of interest into our process. When this is the case, we call the set of parameters that we want to estimate the “state” of our system.

Let’s assume that our state evolves in continuous time subject to initial conditions as

$$\dot{\mathbf{x}}(t) = \mathbf{F}(t)\mathbf{x}(t), \quad \mathbf{x}(t_0) = \mathbf{x}_0$$

As an example, consider a robot moving in the horizontal plane at constant velocity.

If the position is described by x and y and the velocity is described by u and v , then it follows that the dynamics of the robot are

$$\dot{x} = u$$

$$\dot{y} = v$$

$$\dot{u} = 0$$

$$\dot{v} = 0$$

By defining the state vector to be

$$\boldsymbol{x}(t) = \begin{bmatrix} x \\ y \\ u \\ v \end{bmatrix}$$

we can express the set of dynamics for the robot as

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x}(t)$$

This is equivalent to the model $\dot{\boldsymbol{x}}(t) = \boldsymbol{F}(t)\boldsymbol{x}(t)$ with

$$\boldsymbol{F}(t) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

In conjunction with the dynamical system, we have vector observations \mathbf{z}_k of the state at times t_k , and an appropriate model of the observations is given by

$$\mathbf{h}_k = \tilde{\mathbf{H}}_k \mathbf{x}_k \quad \text{where} \quad \mathbf{x}_k = \mathbf{x}(t_k)$$

For the robot case, let's assume that we can observe the position of the robot.

This gives us

$$\mathbf{h}_k = \begin{bmatrix} x_k \\ y_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \mathbf{x}_k = \tilde{\mathbf{H}}_k \mathbf{x}_k$$

The question is... can we handle this type of problem using least squares?

Least squares provides us with the machinery to estimate a static state, or a static set of parameters. We need to estimate a state that has dynamic behavior.

If we can relate the state at any arbitrary time back to what is called the epoch state, \mathbf{x}_0 , then we can apply the least squares method to estimate the state of the system at that time, and then we can use the dynamics to produce estimates of the state of the

system at other times of interest.

Moving forward, we will use $\mathbf{x}_0 = \mathbf{x}(t_0)$ as the epoch state, but it is important to note that the time selected for the epoch state can be generalized.

The solution of the continuous time dynamical system is given by

$$\mathbf{x}(t) = \Phi(t, t_0)\mathbf{x}_0$$

where $\Phi(t, t_0)$ is the state transition matrix, which satisfies the properties:

1. $\Phi(t_i, t_i) = \mathbf{I}_n$
2. $\Phi(t_i, t_\ell) = \Phi(t_i, t_j)\Phi(t_j, t_\ell)$
3. $\Phi(t_i, t_j) = \Phi^{-1}(t_j, t_i)$
4. $\dot{\Phi}(t, t_i) = \mathbf{F}(t)\Phi(t, t_i), \quad \Phi(t_i, t_i) = \mathbf{I}_n$

There is an easy representation for autonomous systems, in which $\mathbf{F}(t) = \mathbf{F}$.

We can expand the state of the system in a Taylor series as

$$\mathbf{x}(t) = \mathbf{x}(t_0) + \dot{\mathbf{x}}(t_0)(t - t_0) + \frac{1}{2}\ddot{\mathbf{x}}(t_0)(t - t_0)^2 + \dots$$

From our system dynamics model, it follows that

$$\dot{\mathbf{x}}(t_0) = \mathbf{F}\mathbf{x}(t_0)$$

Similarly,

$$\ddot{\mathbf{x}}(t_0) = \mathbf{F}\dot{\mathbf{x}}(t_0) = \mathbf{F}\mathbf{F}\mathbf{x}(t_0) = \mathbf{F}^2\mathbf{x}(t_0)$$

Then, from the Taylor series, we see that

$$\begin{aligned}\mathbf{x}(t) &= \mathbf{x}(t_0) + \dot{\mathbf{x}}(t_0)(t - t_0) + \frac{1}{2}\ddot{\mathbf{x}}(t_0)(t - t_0)^2 + \dots \\ &= \mathbf{x}(t_0) + \mathbf{F}\mathbf{x}(t_0)(t - t_0) + \frac{1}{2}\mathbf{F}^2\mathbf{x}(t_0)(t - t_0)^2 + \dots \\ &= [\mathbf{I}_n + \mathbf{F} \cdot (t - t_0) + \frac{1}{2}\mathbf{F}^2 \cdot (t - t_0)^2 + \dots] \mathbf{x}(t_0)\end{aligned}$$

The matrix in brackets is the series expansion of the matrix exponential, such that

$$\mathbf{x}(t) = e^{\mathbf{F} \cdot (t - t_0)} \mathbf{x}(t_0)$$

where

$$e^{\mathbf{F} \cdot (t - t_0)} = [\mathbf{I}_n + \mathbf{F} \cdot (t - t_0) + \frac{1}{2}\mathbf{F}^2 \cdot (t - t_0)^2 + \dots] = \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{F}^k \cdot (t - t_0)^k$$

This is only true for stationary dynamics, and it is implemented as `expm` in MATLAB.

What happens when we apply the matrix exponential to the robot's dynamics?

Recall that

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

Computing $\text{expm}(\mathbf{F} \cdot (t - t_0))$ yields

$$\Phi(t, t_0) = \begin{bmatrix} 1 & 0 & t - t_0 & 0 \\ 0 & 1 & 0 & t - t_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We can also arrive at this solution by noting that $\mathbf{F}^2 = \mathbf{0}$; therefore, only the first two terms of the matrix exponential series are required. In this case, \mathbf{F} is called nilpotent with index 2. In fact, any triangular matrix with zeros along the main diagonal is nilpotent.

From the solution of the linear system, we can write the state at time t_k as

$$\mathbf{x}_k = \Phi(t_k, t_0)\mathbf{x}_0$$

Therefore, we can express the model in terms of the epoch state, which gives

$$\begin{aligned}\mathbf{h}_k &= \tilde{\mathbf{H}}_k \mathbf{x}_k \\ &= \tilde{\mathbf{H}}_k \Phi(t_k, t_0) \mathbf{x}_0 \\ &= \mathbf{H}_k \mathbf{x}_0 \quad \text{where} \quad \mathbf{H}_k = \tilde{\mathbf{H}}_k \Phi(t_k, t_0)\end{aligned}$$

We're done! We have expressed the individual models as functions of a single epoch state, so we can directly apply the work we've done to develop the least-squares solution.

To summarize:

1. We have a dynamical system with some known $\mathbf{F}(t)$

$$\dot{\mathbf{x}}(t) = \mathbf{F}(t) \mathbf{x}(t)$$

2. We have observations \mathbf{z}_k at times t_k , which are modeled as

$$\mathbf{h}_k = \tilde{\mathbf{H}}_k \mathbf{x}_k$$

- For each observation, we integrate the state transition matrix to the time t_k

$$\dot{\Phi}(t, t_0) = \mathbf{F}(t)\Phi(t, t_0), \quad \Phi(t_0, t_0) = \mathbf{I}_n$$

Note: In special cases, we have a closed-form solution of the state transition matrix. In other cases, however, we have to numerically integrate the differential equation for the state transition matrix to get $\Phi(t, t_0)$.

- Next, we determine the mapped observation model matrix, such that

$$\mathbf{H}_k = \tilde{\mathbf{H}}_k \Phi(t_k, t_0)$$

- We assemble the concatenated mapped observation model matrices and the concatenated observations

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{H}}_1 \Phi(t_1, t_0) \\ \tilde{\mathbf{H}}_2 \Phi(t_2, t_0) \\ \vdots \end{bmatrix} \quad \text{and} \quad \mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \end{bmatrix}$$

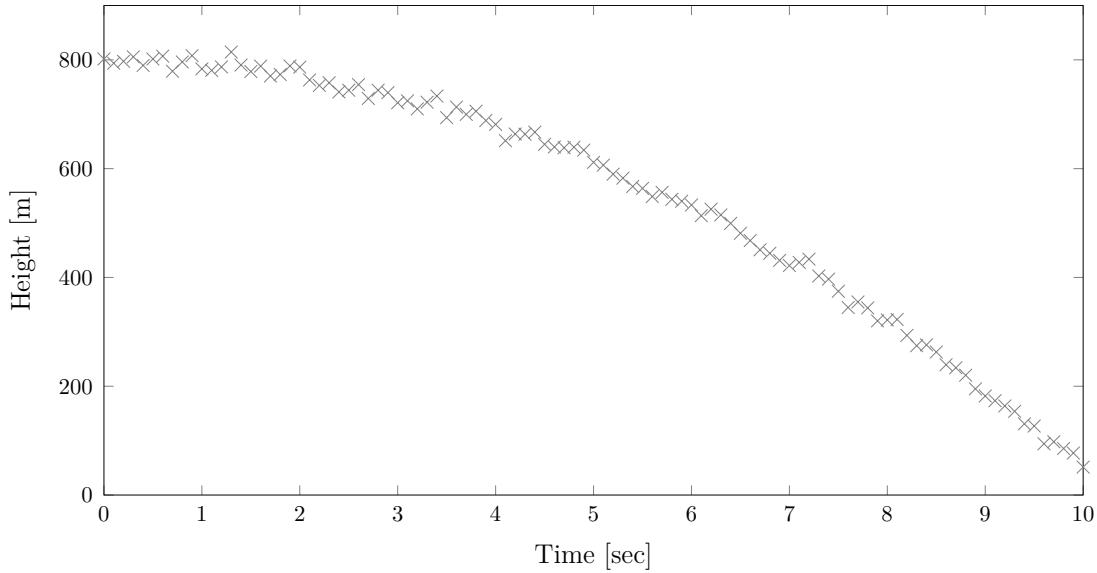
6. Finally, we compute the least-squares estimate of the epoch state

$$\hat{\mathbf{x}}_0 = [\mathbf{H}^T \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{z}$$

2.3.1 Example: Falling Body Problem

Imagine that we are conducting an experiment on a planet with an unknown constant gravitational acceleration, g , and that we want to determine the value of g on this planet.

To do so, we will drop a ball from top of a building of unknown height and record measurements of the height of the ball through time.



The height of the building and the speed at which the ball is “dropped” is also unknown. Therefore, we have three quantities that we want to estimate.

To produce these estimates, we will employ the least-squares approach making use of observations of the ball’s height at a rate of 10Hz. Using these measurements, we seek estimates of the local gravitational acceleration \hat{g} , the height of the building \hat{h}_0 , and the

velocity with which we released the ball $\dot{\hat{h}}_0$.

From Newtonian mechanics, we know that the height of the ball obeys the second-order differential equation

$$\ddot{h}(t) = g$$

Define a state corresponding to height and its rates of change

$$\mathbf{x}_k = [h_k \quad \dot{h}_k \quad \ddot{h}_k]^T$$

Taking the gravitational acceleration to be constant, the second-order dynamical system can be expressed in first-order form as

$$\frac{dh_k}{dt} = \dot{h}_k \quad \frac{d\dot{h}_k}{dt} = \ddot{h}_k \quad \frac{d\ddot{h}_k}{dt} = 0$$

In matrix-vector form, this gives us the model for the dynamical system as

$$\frac{d}{dt} \begin{bmatrix} h_k \\ \dot{h}_k \\ \ddot{h}_k \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_k \\ \dot{h}_k \\ \ddot{h}_k \end{bmatrix}$$

This means that we can write the system matrix, \mathbf{F} , as

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

We also need to establish our measurement model. Since we are taking measurements of the height of the ball, we have

$$z_k = h_k$$

This can then be written as a linear combination of our states as

$$z_k = [1 \ 0 \ 0] \begin{bmatrix} h_k \\ \dot{h}_k \\ \ddot{h}_k \end{bmatrix}$$

Therefore, our model matrix is

$$\tilde{\mathbf{H}}_k = [1 \ 0 \ 0]$$

We then use the fact that by definition

$$\Phi(t_k, t_0) = e^{\mathbf{F} \cdot (t_k - t_0)}$$

to construct the concatenated observation model and observations to perform the least-squares fit.

Let's look at how we would implement this in MATLAB.

Since we will be generating measurements based on a random generator, let's set the random number generation seed so we can exactly replicate these results.

```
% Set random seed  
rng(100)
```

Now, we have to generate our observations. First, define a sequence of times at which observations will occur and the *true* gravitational acceleration.

```
t = 0:0.1:10; % Generate data every 0.1 [sec] for 10 [sec]  
g = -15;       % True gravitational acceleration [m/s^2]
```

Define the true motion of the ball so we can generate noisy observations of its motion. We know that its motion (under constant acceleration) obeys $x(t) = x_0 + v_0t + \frac{1}{2}gt^2$.

```
% True motion of the ball
h0 = 800;          % True height of the building [m]
v0 = 0;            % True drop speed [m/s]
h = h0 + v0.*t + 0.5.*g.*t.^2;
```

Generate measurements of the altitude of the ball by adding noise to the true altitude of the ball.

```
s = 10;           % Standard deviation of meas. [m]
z = h + s.*randn(size(x)); % Add noise to height meas. [m]
z = z';
```

In order to construct a least-squares estimate, we need the concatenated observational model matrix. We have to assemble this in a loop.

```

F = [0, 1, 0; 0, 0, 1; 0, 0, 0];
Ht = [1, 0, 0];
H = [];
for i = 1:length(z)
    Phi = expm(F*(t(i) - t(1)));
    Hk = Ht*Phi;
    H = [H; Hk];
end

```

Finally, we perform the least-squares fit.

```

% Least-squares estimate
xh0 = (H'*H)\(H'*z);

```

This procedure produces estimates for the gravitational acceleration and initial height as

$$\hat{g} = -14.9269 \text{ [m/s}^2\text{]} \quad \hat{h}_0 = 800.4682 \text{ [m]}$$

corresponding to true values of

$$g = -15 \text{ [m/s}^2\text{]} \quad h_0 = 800 \text{ [m]}$$

2.3.2 Example: Coding a Robot Estimation Problem

Let's code up a least-squares solver for estimating the position and velocity of a robot.

We've already discussed most of the elements that we need to solve this problem.

We'll walk through the main elements of each of the parts of a MATLAB code to generate the true path of a robot, the measurements of the position of the robot, and finally the construction of a least-squares estimate for the initial position and velocity of the robot.

In this case, we are assuming that we have a robot operating in a planar environment under a constant acceleration model. We are also assuming that we can acquire observations of the position of the robot, say by mounting a camera above the environment and taking images.

Since we would like to be able to replicate any experiments that we do, we start off our code by setting the seed on a random number generator.

```
% Set random seed  
rng(150)
```

For this problem, we will assume that the position is described by x and y and that the velocity is described by u and v . We will combine these four quantities into a vector to form our state; that is

$$\boldsymbol{x}(t) = [x(t) \ y(t) \ u(t) \ v(t)]^T$$

where we note that $x(t)$ is one element of the state and $\boldsymbol{x}(t)$ is the full state. To simulate the true path that the robot takes, we need to specify the true initial position and velocity of the robot.

```
% Specify the initial true state of the robot  
x0 = [0; 0; 0.1; 0.2];
```

We need to fix a range of times for which we will simulate the true path of the robot; this also gives us the ability to simulate our measurements of the position of the robot.

```

% Set up our timing variables
% - assume measurements every 0.1 [sec]
% - assume the measurements continue for 10 [sec]
t0 = 0.0;
dt = 0.1;
tf = 10.0;
tv = (t0:dt:tf)';

```

The last piece that we need to simulate the true path of the vehicle is the dynamical system description. Previously, we noted that the constant velocity model gives us the dynamics of the states as

$$\dot{x}(t) = u(t) \quad \dot{y}(t) = v(t) \quad \dot{u}(t) = 0 \quad \text{and} \quad \dot{v}(t) = 0$$

From the definition of our state vector, these dynamics give us the matrix-vector system

$$\dot{\mathbf{x}}(t) = \mathbf{F}(t)\mathbf{x}(t) \quad \text{where} \quad \mathbf{F}(t) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

We can now specify the dynamics in our code.

```
% Dynamics of the robot (continuous time)
F = [0, 0, 1, 0; 0, 0, 0, 1; 0, 0, 0, 0; 0, 0, 0, 0];
```

Now, we are ready to simulate the true path of our robot. We will do this using numerical integration to solve the initial value problem

$$\dot{\mathbf{x}}(t) = \mathbf{F}(t)\mathbf{x}(t) \quad \text{s.t. } \mathbf{x}(t_0) = \mathbf{x}_0$$

In MATLAB, the easiest (and most general) way is to use a numerical integrator, such as `ode45`, to propagate the state forward in time.

```
% Integrate the state for the true object
opts = odeset('AbsTol',1e-9,'RelTol',1e-9);
[~,X] = ode45(@eom_robot,tv,x0,opts,F);
```

Now that we have the true state as a function of time, we can generate measurements of the position. Since we are only considering measurements of the position, we will take the first two states at each time and we will add on some measurement noise to simulate the effects of the observation process. Here, we have assumed that the camera system can determine the position to within about 0.1 meters of the actual position.

```
% Create measurements of position
% - add noise with a standard deviation of 0.1 [m]
z = X(:,1:2)' + 0.1*randn(2,length(tv));
```

Up to this point, we've just been constructing the information that we need to synthesize measurements. We have not done anything related to determining a least-squares estimate. We will now move to that process.

We need two more pieces of information: the state transition matrix and the measurement mapping matrix.

For this problem, we've already determined these two elements:

$$\Phi(t_k, t_0) = \begin{bmatrix} 1 & 0 & t_k - t_0 & 0 \\ 0 & 1 & 0 & t_k - t_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \tilde{\mathbf{H}}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The state transition matrix can be found via the matrix exponential, and the measurement mapping matrix can be found by recalling that our measurements are of the position

of the robot.

Before computing the least-squares estimate, we need to accumulate all of the observations and all of the measurement mapping matrices (accounting for the state transition matrix). These are given by

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \end{bmatrix} \quad \text{and} \quad \mathbf{H} = \begin{bmatrix} \tilde{\mathbf{H}}_1 \Phi(t_1, t_0) \\ \tilde{\mathbf{H}}_2 \Phi(t_2, t_0) \\ \vdots \end{bmatrix}$$

We will do this inside of a loop.

1. Add in the code to accumulate the observations.
2. Add in the code to compute $\tilde{\mathbf{H}}_k$.
3. Add in the code to compute $\Phi(t_k, t_0)$.
4. Add in the code to accumulate the measurement mapping matrices.

The following is what the template code should look like where you'll be adding in the preceding elements.

```

% Assemble the concatenated measurement vector and ...
model matrix
z = [];
H = [];
for k = 1:length(tv)
    % time at kth observation
    tk = tv(k);

    % concatenated measurements
    z =

    % concatenated model matrix
    Htilde =
    Phik0 =
    H      =
end

```

Finally, we compute the least-squares estimate of the epoch state using

$$\hat{x}_0 = [\mathbf{H}^T \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{z}$$

Add in the code to compute the estimated initial position and velocity.

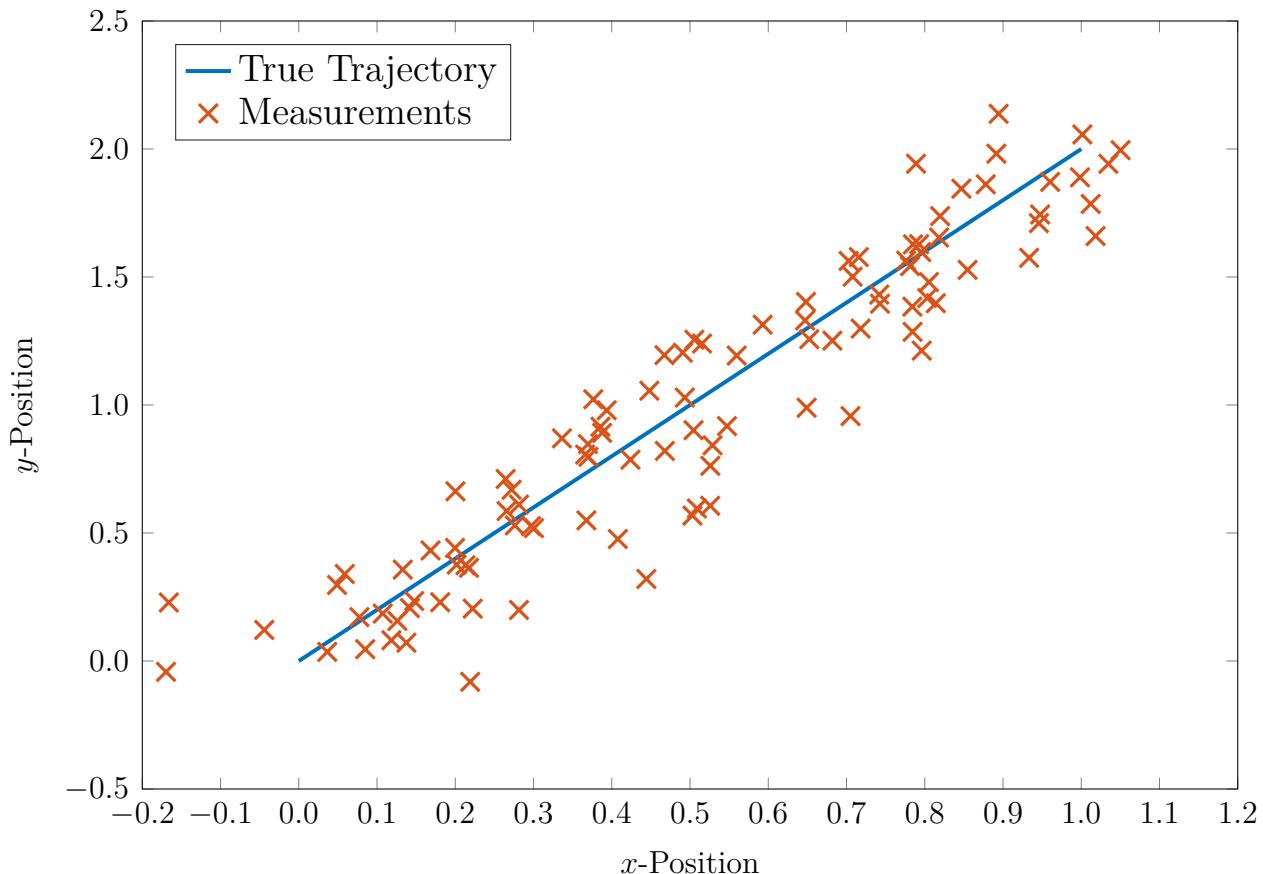
```
% Least-squares estimate  
xhat0 =
```

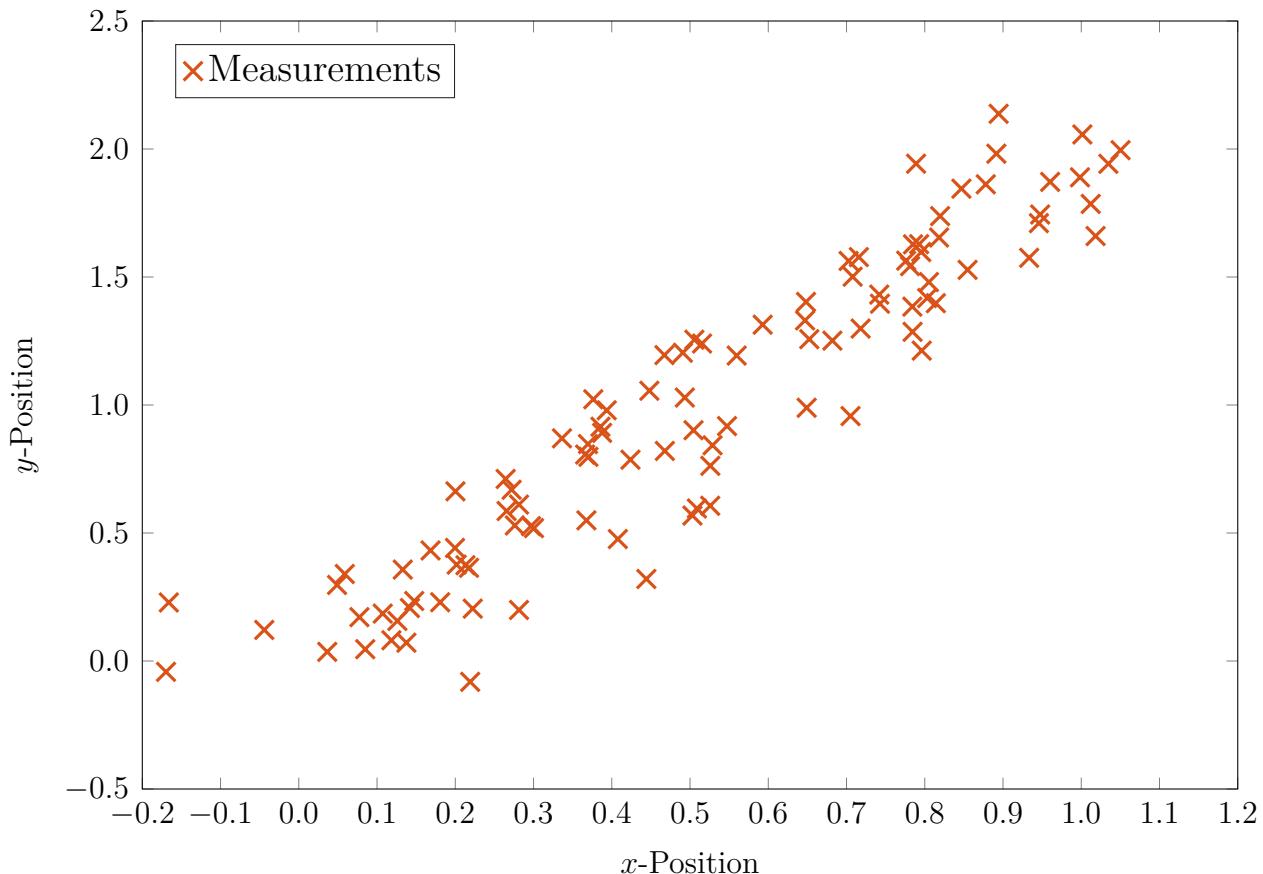
You should find that the estimated state is

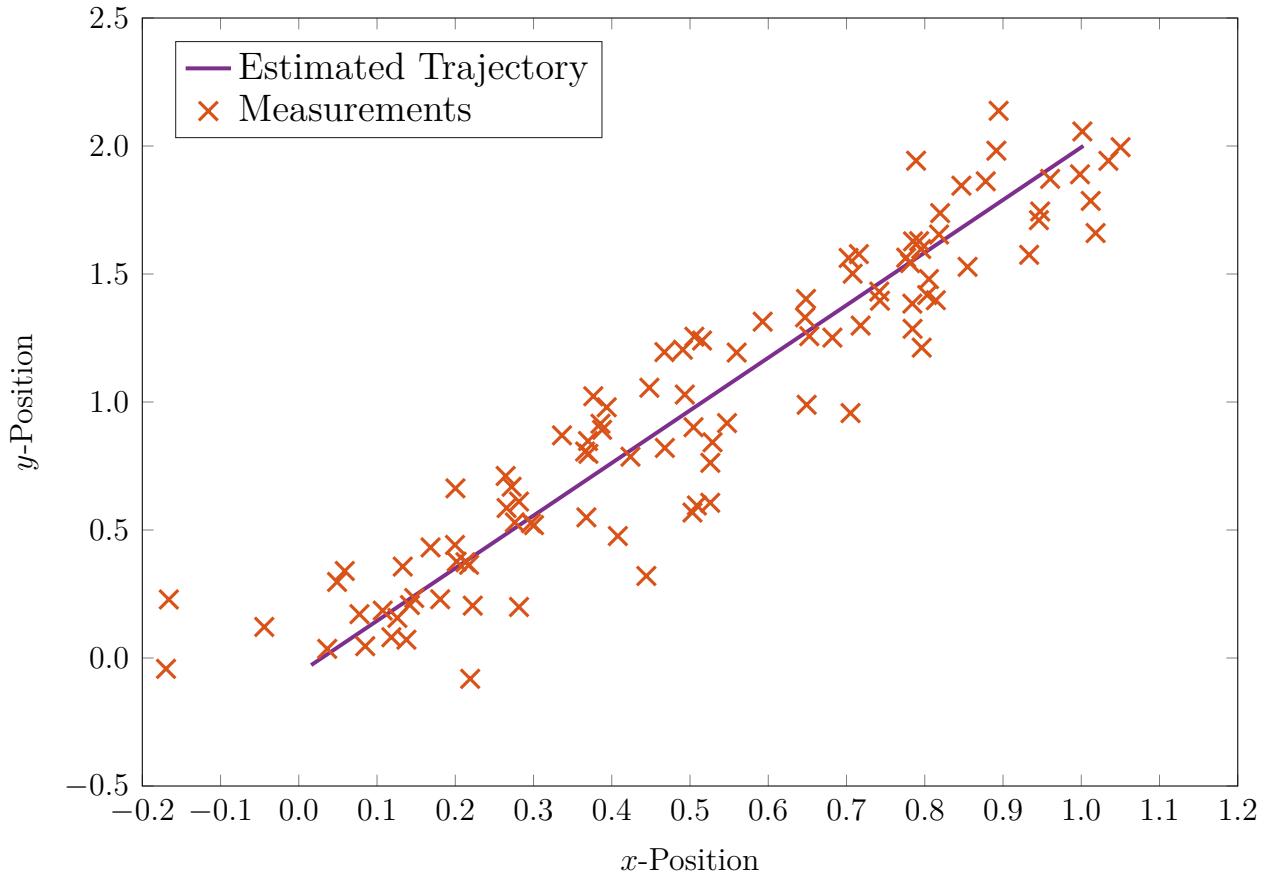
$$\hat{x}_0 = \begin{bmatrix} 0.0158 \\ -0.0277 \\ 0.0987 \\ 0.2028 \end{bmatrix}$$

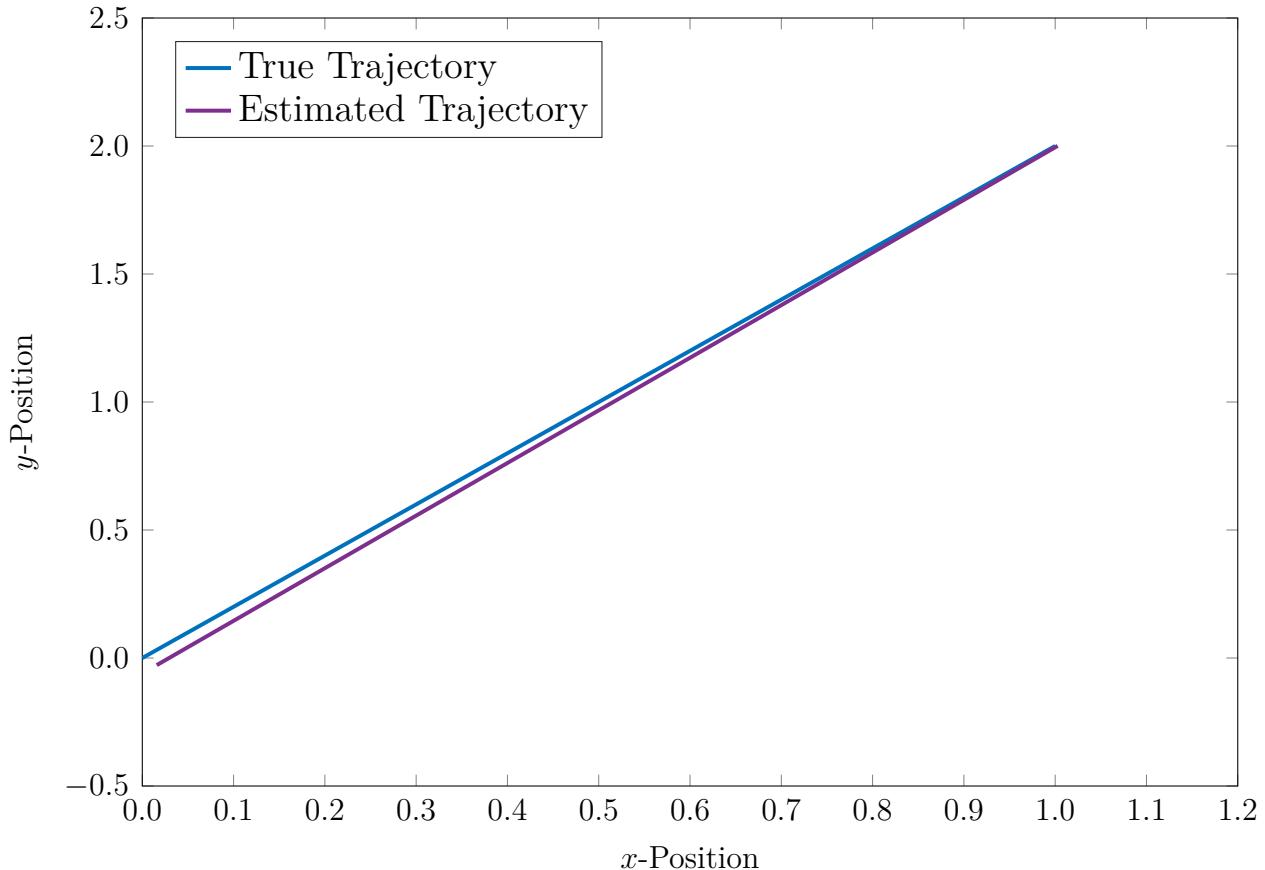
Comparing this to our initial true state, we see that we have a pretty good estimate of the initial position and velocity of our robot.

Let's look at a few plots to see what we got from the least squares process.









2.4 Weighted Least Squares

A shortcoming of the least-squares method is that it does not provide a mechanism for weighting certain observations more (or less) heavily than others. In cases where we may have more confidence in some measurements, due to their inherent accuracy, we would like to be able to place more emphasis on the information provided by these measurements.

Consider the case where we have measurements z_i at times t_i and where our predicted observations are modeled by

$$\mathbf{h}_i = \mathbf{H}_i \mathbf{x}$$

If we are estimating the state of a dynamic system, then we would write our predicted observations as

$$\mathbf{h}_i = \tilde{\mathbf{H}}_i \Phi(t_i, t_0) \mathbf{x}_0$$

We will use the first formulation of the predicted observations for ease of notation with the understanding that the second formulation is equivalent.

The i^{th} residual is given by

$$\boldsymbol{\epsilon}_i = \mathbf{z}_i - \mathbf{H}_i \mathbf{x}$$

In the standard least squares formulation, each of the residuals is given the same weight and the least-squares performance index for q measurements is taken to be

$$J = \sum_{i=1}^q \boldsymbol{\epsilon}_i^T \boldsymbol{\epsilon}_i$$

We wish to extend our formulation so that we can weight individual residuals (or individual measurements) differently. To accomplish this, let each residual be accompanied by a weight $w_i > 0$. In the vector-measurement case, this weight is a weight matrix of the form $\mathbf{W}_i = \mathbf{W}_i^T > \mathbf{0}$.

This gives us a set of q vector residuals with associated weights as

$$\boldsymbol{\epsilon}_i = \mathbf{z}_i - \mathbf{H}_i \mathbf{x} \quad \text{with weight } \mathbf{W}_i$$

The performance index is then modified to be

$$\begin{aligned} J &= \sum_{i=1}^q \boldsymbol{\epsilon}_i^T \mathbf{W}_i \boldsymbol{\epsilon}_i \\ &= \sum_{i=1}^q [\mathbf{z}_i - \mathbf{H}_i \mathbf{x}]^T \mathbf{W}_i [\mathbf{z}_i - \mathbf{H}_i \mathbf{x}] \end{aligned}$$

As before, we define concatenated terms, including a concatenated weight, such that

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \\ \vdots \\ \mathbf{H}_q \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_q \end{bmatrix}, \quad \text{and} \quad \mathbf{W} = \begin{bmatrix} \mathbf{W}_1 & & & \\ & \mathbf{W}_2 & & \\ & & \ddots & \\ & & & \mathbf{W}_q \end{bmatrix}$$

With these definitions, the weighted least-squares performance index becomes

$$J = [\mathbf{z} - \mathbf{Hx}]^T \mathbf{W} [\mathbf{z} - \mathbf{Hx}]$$

which is completely equivalent to the summation representation.

The first derivative condition for an optimal is

$$\frac{\partial J}{\partial \mathbf{x}} = \mathbf{0}^T \quad \text{where} \quad \frac{\partial J}{\partial \mathbf{x}} = -2[\mathbf{z} - \mathbf{H}\mathbf{x}]^T \mathbf{W}\mathbf{H}$$

Applying the first derivative condition to solve for $\hat{\mathbf{x}}$ yields

$$\mathbf{0}^T = -2[\mathbf{z} - \mathbf{H}\hat{\mathbf{x}}]^T \mathbf{W}\mathbf{H}$$

Manipulation of the preceding equation gives

$$\mathbf{H}^T \mathbf{W} \mathbf{H} \hat{\mathbf{x}} = \mathbf{H}^T \mathbf{W} \mathbf{z}$$

This is the normal equation for the weighted least-squares problem, which has the solution

$$\hat{\mathbf{x}} = [\mathbf{H}^T \mathbf{W} \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{W} \mathbf{z}$$

provided that the inverse exists.

Note that if the measurements (residuals) are all equally weighted, then $\mathbf{W} = w\mathbf{I}$, and

the solution becomes

$$\begin{aligned}\hat{\mathbf{x}} &= [\mathbf{H}^T w \mathbf{I} \mathbf{H}]^{-1} \mathbf{H}^T w \mathbf{I} \mathbf{z} \\ &= w [\mathbf{H}^T \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{z} \\ &= [\mathbf{H}^T \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{z}\end{aligned}$$

That is, we recover the original least-squares solution.

Is our solution a minimum? We need the second derivative to figure this out. Recall that the first derivative (before any manipulations) is

$$\frac{\partial J}{\partial \mathbf{x}} = -2[\mathbf{z} - \mathbf{Hx}]^T \mathbf{W} \mathbf{H}$$

such that

$$\begin{aligned}\frac{\partial}{\partial \mathbf{x}} \left[\frac{\partial J}{\partial \mathbf{x}} \right]^T &= \frac{\partial}{\partial \mathbf{x}} \left\{ -2 \mathbf{H}^T \mathbf{W} [\mathbf{z} - \mathbf{Hx}] \right\} \\ &= 2 \mathbf{H}^T \mathbf{W} \mathbf{H}\end{aligned}$$

Is the second derivative condition satisfied? Is this matrix positive definite? That is

$$2\mathbf{H}^T \mathbf{W} \mathbf{H} \stackrel{?}{> 0}$$

Note that we can factor $\mathbf{W} = \mathbf{V}^T \mathbf{V}$, such that

$$\frac{\partial}{\partial \mathbf{x}} \left[\frac{\partial J}{\partial \mathbf{x}} \right]^T = 2[\mathbf{V} \mathbf{H}]^T [\mathbf{V} \mathbf{H}]$$

Since $\mathbf{W} > 0$, $\mathbf{V} > 0$, and $\text{rank}\{\mathbf{V} \mathbf{H}\} = \text{rank}\{\mathbf{H}\}$. As such, we can use the previous arguments to conclude that if $\text{rank}\{\mathbf{H}\} = n$, then $\text{rank}\{\mathbf{H}^T \mathbf{W} \mathbf{H}\} = n$, i.e., the second derivative condition is satisfied so long as there are n linearly independent observations.

Since we can weight each observation differently, one might ask if we can also add some prior information regarding the state of the system. That is to say, if we have some information about the state, call it $\bar{\mathbf{x}}$, can we include this in our weighted least-squares approach? In addition, can we include it with some weight $\bar{\mathbf{W}}$?

We still have the sequence of observations \mathbf{z}_i at times t_i that are modeled as $\mathbf{h}_i = \mathbf{H}_i \mathbf{x}$.

We can also add a “zeroth” observation of the state, or

$$\mathbf{z}_0 = \bar{\mathbf{x}}$$

and model this observation as

$$\mathbf{h}_0 = \mathbf{x} \quad \text{i.e.} \quad \mathbf{H}_0 = \mathbf{I}_n$$

and assign it a weight of $\mathbf{W}_0 = \bar{\mathbf{W}}$.

Then, our weighted least-squares performance index is

$$J = \sum_{i=0}^q [\mathbf{z}_i - \mathbf{H}_i \mathbf{x}]^T \mathbf{W}_i [\mathbf{z}_i - \mathbf{H}_i \mathbf{x}]$$

Extracting out the zeroth observation yields

$$J = [\mathbf{z}_0 - \mathbf{H}_0 \mathbf{x}]^T \mathbf{W}_0 [\mathbf{z}_0 - \mathbf{H}_0 \mathbf{x}] + \sum_{i=1}^q [\mathbf{z}_i - \mathbf{H}_i \mathbf{x}]^T \mathbf{W}_i [\mathbf{z}_i - \mathbf{H}_i \mathbf{x}]$$

Substituting for the definitions of \mathbf{z}_0 , \mathbf{H}_0 , and \mathbf{W}_0 and writing the summation in the usual concatenated form yields

$$J = [\bar{\mathbf{x}} - \mathbf{x}]^T \bar{\mathbf{W}} [\bar{\mathbf{x}} - \mathbf{x}] + [\mathbf{z} - \mathbf{H}\mathbf{x}]^T \mathbf{W} [\mathbf{z} - \mathbf{H}\mathbf{x}]$$

Now, we are ready to apply the derivative conditions to solve for the least-squares estimate.

The first derivative condition for an optimal is

$$\frac{\partial J}{\partial \mathbf{x}} = \mathbf{0}^T$$

where

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{x}} &= \frac{\partial}{\partial \mathbf{x}} \left\{ [\bar{\mathbf{x}} - \mathbf{x}]^T \bar{\mathbf{W}} [\bar{\mathbf{x}} - \mathbf{x}] + [\mathbf{z} - \mathbf{H}\mathbf{x}]^T \mathbf{W} [\mathbf{z} - \mathbf{H}\mathbf{x}] \right\} \\ &= -2[\bar{\mathbf{x}} - \mathbf{x}]^T \bar{\mathbf{W}} - 2[\mathbf{z} - \mathbf{H}\mathbf{x}]^T \mathbf{W} \mathbf{H} \end{aligned}$$

Applying the first derivative condition to solve for $\hat{\mathbf{x}}$ yields

$$\mathbf{0} = -2\bar{\mathbf{W}}[\bar{\mathbf{x}} - \hat{\mathbf{x}}] - 2\mathbf{H}^T \mathbf{W}[\mathbf{z} - \mathbf{H}\hat{\mathbf{x}}]$$

Manipulation of the preceding equation gives

$$[\mathbf{H}^T \mathbf{W} \mathbf{H} + \bar{\mathbf{W}}] \hat{\mathbf{x}} = \mathbf{H}^T \mathbf{W} \mathbf{z} + \bar{\mathbf{W}} \bar{\mathbf{x}}$$

which has the solution

$$\hat{\mathbf{x}} = [\mathbf{H}^T \mathbf{W} \mathbf{H} + \bar{\mathbf{W}}]^{-1} [\mathbf{H}^T \mathbf{W} \mathbf{z} + \bar{\mathbf{W}} \bar{\mathbf{x}}]$$

This is the weighted least-squares solution with the inclusion of some prior knowledge of the state of the system.

If we have no prior knowledge of the state, then $\bar{\mathbf{W}} = \mathbf{0}_{n \times n}$, and we are left with the standard weighted least-squares solution.

2.5 The Minimum Variance Estimate

Least-squares provides a very powerful framework for estimating the state of a static or dynamic system; however,

- the original formulation of least squares gave us no ability to embed more or less confidence in individual observations of the system, so we developed the weighted least squares method; and
- while the weighted least squares method provided us with a natural mechanism for including information that we may have regarding the state of the system prior to acquiring measurements,

neither of these approaches includes any information on the statistical characteristics of the measurements or of the prior state.

The weights in weighted least-squares are correspond to how much we believe one quantity over another. The minimum variance approach, on the other hand, provides a mechanism for selecting weights in a statistically meaningful manner. This approach is statistical in nature, but it only requires the first and second moments (the mean and covariance) of the measurement errors and the prior state error.

Given the dynamical system

$$\mathbf{x}_i = \Phi(t_i, t_k)\mathbf{x}_k$$

and the observational system

$$\mathbf{z}_i = \tilde{\mathbf{H}}_i \mathbf{x}_i + \mathbf{v}_i$$

where

$$\text{E}\{\mathbf{v}_i\} = \mathbf{0} \quad \forall i \quad \text{and} \quad \text{E}\{\mathbf{v}_i \mathbf{v}_j^T\} = \mathbf{P}_{v_i v_j}$$

the objective of the minimum variance estimator is to find the linear, unbiased, minimum variance estimate (LUMVE), $\hat{\mathbf{x}}_k$ of the state \mathbf{x}_k .

Note that the problem statement now includes measurement errors; otherwise, this is the same problem statement that we encountered with least-squares state estimation.

First, using the methods employed previously, we reduce this problem to one of estimating \mathbf{x}_k through the use of the state transition matrix. That is, we express the collection of measurements as

$$\mathbf{z} = \mathbf{H} \mathbf{x}_k + \mathbf{v}$$

where

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \tilde{\mathbf{H}}_1 \Phi(t_1, t_k) \\ \tilde{\mathbf{H}}_2 \Phi(t_2, t_k) \\ \vdots \end{bmatrix} \quad \text{and} \quad \mathbf{v} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \end{bmatrix}$$

It is important to remember that we are now describing the measurements as being the model subjected to measurement noise. We assume that the concatenated measurement noise has first and second central moments of

$$E\{\mathbf{v}\} = \begin{bmatrix} E\{\mathbf{v}_1\} \\ E\{\mathbf{v}_2\} \\ \vdots \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \end{bmatrix} \quad \text{and} \quad E\{\mathbf{v}\mathbf{v}^T\} = \begin{bmatrix} \mathbf{P}_{vv,1} & \mathbf{P}_{v_1v_2} & \cdots \\ \mathbf{P}_{v_1v_2}^T & \mathbf{P}_{vv,2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} = \mathbf{P}_{vv}$$

Generally, the terms $\mathbf{P}_{v_iv_j}$ for $i \neq j$ are zero, implying that the measurement noise is a white sequence (uncorrelated in time). Also, it is common to find that the terms $\mathbf{P}_{vv,i}$ are all equal. Neither of these conditions, however, is required to be imposed in order to continue. In fact, the case of $\mathbf{P}_{v_iv_j} \neq \mathbf{0}$ corresponds to the case of time-correlated observation errors.

Now, we will begin to apply the conditions of LUMVE (linear, unbiased, and minimum variance) to determine an estimate, $\hat{\mathbf{x}}_k$.

Linear We require that our estimate is a linear combination of the measurement data that we received. That is, we form the estimate as

$$\hat{\mathbf{x}}_k = \mathbf{M}\mathbf{z}$$

where the matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$ is to be determined.

It is important to note that the estimated state is indeed random, but that we will assume that the true state of the system is not random.

Unbiased We require that our estimate is unbiased. That is, the expected value of our estimate should be the true state, which gives

$$\mathbb{E}\{\hat{\mathbf{x}}_k\} = \mathbf{x}_k$$

From the linear requirement, it follows that

$$\mathbb{E}\{\mathbf{M}\mathbf{z}\} = \mathbf{x}_k$$

Then, from our assumed model for the measurement data, we have

$$\mathrm{E}\{\boldsymbol{M}[\boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{v}]\} = \mathrm{E}\{\boldsymbol{M}\boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{M}\boldsymbol{v}\} = \boldsymbol{x}_k$$

The expected value is a linear operator, so we are able to break the summation apart. Moreover, since \boldsymbol{H} and \boldsymbol{M} are deterministic matrices, they can be brought outside of the expectation, such that

$$\boldsymbol{M}\boldsymbol{H}\mathrm{E}\{\boldsymbol{x}_k\} + \boldsymbol{M}\mathrm{E}\{\boldsymbol{v}\} = \boldsymbol{x}_k$$

Finally, recalling that the measurement noise is taken to be zero mean and that the true state is not random, we find that

$$\boldsymbol{M}\boldsymbol{H}\boldsymbol{x}_k = \boldsymbol{x}_k$$

or that $\boldsymbol{M}\boldsymbol{H} = \boldsymbol{I}_n$ in order to obtain an unbiased estimate.

This effectively imposes a constraint on the matrix \boldsymbol{M} . This condition requires that the rows of \boldsymbol{M} are orthogonal to the columns of \boldsymbol{H} .

Minimum Variance This is the heart of LUMVE, so it will take a bit of work to get through the minimum variance condition.

We want our estimate to be linear and unbiased, but we also want our estimate to be “good” in some quantifiable sense. In particular, we want to extract as much information as possible regarding the state of the system from our data.

Let’s define the error in our estimate to be the deviation of our estimate away from the truth:

$$\mathbf{e}_{x,k} = \mathbf{x}_k - \hat{\mathbf{x}}_k$$

Using the definition of the state estimation error, we can show that the unbiased condition gives us an expected error that is zero, i.e.,

$$\mathrm{E}\{\mathbf{e}_{x,k}\} = \mathrm{E}\{\mathbf{x}_k - \hat{\mathbf{x}}_k\} = \mathrm{E}\{\mathbf{x}_k\} - \mathrm{E}\{\hat{\mathbf{x}}_k\} = \mathbf{x}_k - \mathbf{x}_k = \mathbf{0}$$

which is desirable. From here, we can define the covariance of the error. We simply take the expected value of the product of the deviation of the error from its mean and the transpose of this quantity to give

$$\begin{aligned} \mathbf{P}_{xx,k} &= \mathrm{E}\{[\mathbf{e}_{x,k} - \mathrm{E}\{\mathbf{e}_{x,k}\}][\mathbf{e}_{x,k} - \mathrm{E}\{\mathbf{e}_{x,k}\}]^T\} \\ &= \mathrm{E}\{\mathbf{e}_{x,k}\mathbf{e}_{x,k}^T\} \end{aligned}$$

The unbiased condition centers our error on zero, but we also want the spread about the center to be small, and this is where our minimum variance condition comes in to play.

From the preceding covariance equation and the definition of the error, it follows that the error covariance is

$$\mathbf{P}_{xx,k} = \text{E}\{\left[\mathbf{x}_k - \hat{\mathbf{x}}_k\right]\left[\mathbf{x}_k - \hat{\mathbf{x}}_k\right]^T\}$$

We can manipulate this expression a bit by recalling the conditions of our estimator.

First of all, the estimate is linear, so we can replace the estimated state with its linear mapping to give

$$\mathbf{P}_{xx,k} = \text{E}\{\left[\mathbf{x}_k - \mathbf{Mz}\right]\left[\mathbf{x}_k - \mathbf{Mz}\right]^T\}$$

Next, we know that the data are related to the state by the observational equation,

so we can substitute this back into the covariance expression

$$\begin{aligned}\mathbf{P}_{xx,k} &= \text{E}\{\left[\mathbf{x}_k - \mathbf{M}(\mathbf{H}\mathbf{x}_k + \mathbf{v})\right]\left[\mathbf{x}_k - \mathbf{M}(\mathbf{H}\mathbf{x}_k + \mathbf{v})\right]^T\} \\ &= \text{E}\{\left[\mathbf{x}_k - \mathbf{M}\mathbf{H}\mathbf{x}_k - \mathbf{M}\mathbf{v}\right]\left[\mathbf{x}_k - \mathbf{M}\mathbf{H}\mathbf{x}_k - \mathbf{M}\mathbf{v}\right]^T\}\end{aligned}$$

Now, we recall the condition for an unbiased estimator is $\mathbf{M}\mathbf{H} = \mathbf{I}_n$, which gives

$$\mathbf{P}_{xx,k} = \text{E}\{\left[\mathbf{x}_k - \mathbf{x}_k - \mathbf{M}\mathbf{v}\right]\left[\mathbf{x}_k - \mathbf{x}_k - \mathbf{M}\mathbf{v}\right]^T\}$$

A simple elimination of the true state produces a covariance expression that is only dependent upon the linear mapping of the estimator and the measurement noise

$$\begin{aligned}\mathbf{P}_{xx,k} &= \text{E}\{\mathbf{M}\mathbf{v}\mathbf{v}^T\mathbf{M}^T\} \\ &= \mathbf{M}\text{E}\{\mathbf{v}\mathbf{v}^T\}\mathbf{M}^T\end{aligned}$$

where the second step follows from the fact that \mathbf{M} is deterministic. The expected value here is just the covariance of the concatenated measurement, such that

$$\mathbf{P}_{xx,k} = \mathbf{M}\mathbf{P}_{vv}\mathbf{M}^T$$

This is the covariance we wish to minimize, but we also have to keep in mind the active constraint on the matrix \mathbf{M} .

To account for the constraint on \mathbf{M} , we add “zero” to the covariance matrix, keeping in mind that the covariance matrix is symmetric, such that

$$\mathbf{P}_{xx,k} = \mathbf{M}\mathbf{P}_{vv}\mathbf{M}^T + \boldsymbol{\Lambda}^T[\mathbf{I}_n - \mathbf{M}\mathbf{H}]^T + [\mathbf{I}_n - \mathbf{M}\mathbf{H}]\boldsymbol{\Lambda}$$

The matrix $\boldsymbol{\Lambda}$ is a matrix of unspecified Lagrange multipliers that account for the constraint on the matrix \mathbf{M} while still retaining a symmetric covariance matrix. The advantage is that we can proceed with unconstrained minimization techniques.

To find a minimum, we need to set the first variation of the covariance matrix equal to zero

$$\delta\mathbf{P}_{xx,k} = \mathbf{0}$$

We have two parameters to solve for: \mathbf{M} and $\boldsymbol{\Lambda}$.

The variation of $\mathbf{P}_{xx,k}$ with respect to $\boldsymbol{\Lambda}$ simply returns the constraint $\mathbf{M}\mathbf{H} = \mathbf{I}_n$.

The variation of $\mathbf{P}_{xx,k}$ with respect to \mathbf{M} is what we're really after, and this is

$$\delta \mathbf{P}_{xx,k} = \delta \mathbf{M} \cdot \mathbf{P}_{vv} \mathbf{M}^T + \mathbf{M} \mathbf{P}_{vv} \cdot \delta \mathbf{M}^T + \boldsymbol{\Lambda}^T [-\delta \mathbf{M} \cdot \mathbf{H}]^T + [-\delta \mathbf{M} \cdot \mathbf{H}] \boldsymbol{\Lambda}$$

After a bit of rearranging, we find that the first variation is

$$\delta \mathbf{P}_{xx,k} = \delta \mathbf{M} [\mathbf{P}_{vv} \mathbf{M}^T - \mathbf{H} \boldsymbol{\Lambda}] + [\mathbf{M} \mathbf{P}_{vv} - \boldsymbol{\Lambda}^T \mathbf{H}^T] \delta \mathbf{M}^T$$

Now, we set this equal to zero, which will happen if

$$\mathbf{M} \mathbf{P}_{vv} - \boldsymbol{\Lambda}^T \mathbf{H}^T = \mathbf{0}$$

Note that it will also occur if $\delta \mathbf{M}$ and/or $\mathbf{M} \mathbf{P}_{vv} - \boldsymbol{\Lambda}^T \mathbf{H}^T$ are not full rank. We will focus on the first condition in order to find the requirements to obtain a minimum of the covariance matrix.

We need to solve a set of two simultaneous equations

$$\mathbf{M} \mathbf{P}_{vv} - \boldsymbol{\Lambda}^T \mathbf{H}^T = \mathbf{0} \quad \text{and} \quad \mathbf{I}_n - \mathbf{M} \mathbf{H} = \mathbf{0}$$

Since \mathbf{P}_{vv} is taken to be positive definite, its inverse is guaranteed to exist, and we can solve the first equation for the matrix \mathbf{M} as

$$\mathbf{M} = \boldsymbol{\Lambda}^T \mathbf{H}^T \mathbf{P}_{vv}^{-1}$$

We can substitute this result into the second equation to find that

$$\mathbf{I}_n = \boldsymbol{\Lambda}^T \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}$$

Provided that we have more observations than parameters, the inverse of $\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}$ will exist, and we can solve for the matrix of Lagrange multipliers as

$$\boldsymbol{\Lambda}^T = [\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1}$$

Finally, we take the now known Lagrange multipliers and substitute them back into the solution for the matrix \mathbf{M} , which gives

$$\mathbf{M} = [\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{P}_{vv}^{-1}$$

This is the value of \mathbf{M} that minimizes the (co)variance $\mathbf{P}_{xx,k}$ while simultaneously yielding an unbiased estimate $\hat{\mathbf{x}}_k$.

The covariance matrix can then be found as

$$\begin{aligned}
 \mathbf{P}_{xx,k} &= \mathbf{M} \mathbf{P}_{vv} \mathbf{M}^T \\
 \text{subst. for } \mathbf{M} &= \left\{ [\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{P}_{vv}^{-1} \right\} \mathbf{P}_{vv} \left\{ \mathbf{P}_{vv}^{-1} \mathbf{H} [\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1} \right\} \\
 \text{eliminate and group} &= [\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1} [\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}] [\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1} \\
 \text{cancel} &= [\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1}
 \end{aligned}$$

An alternative way of finding the covariance matrix depends on the matrix of Lagrange multipliers:

$$\begin{aligned}
 \mathbf{P}_{xx,k} &= \mathbf{M} \mathbf{P}_{vv} \mathbf{M}^T \\
 \text{subst. for } \mathbf{M} \mathbf{P}_{vv} = \boldsymbol{\Lambda}^T \mathbf{H}^T &= \boldsymbol{\Lambda}^T \mathbf{H}^T \mathbf{M}^T \\
 \text{regroup terms} &= \boldsymbol{\Lambda}^T [\mathbf{M} \mathbf{H}]^T \\
 \text{subst. for } \mathbf{M} \mathbf{H} = \mathbf{I}_n &= \boldsymbol{\Lambda}^T \\
 \text{def. of } \boldsymbol{\Lambda} &= [\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1}
 \end{aligned}$$

Even though we have found a solution to the first variational equation, it is not obvious that this solution actually minimizes the estimation error covariance.

To show that this solution does indeed yield a minimum variance estimate, we first recall that the estimate is given by

$$\hat{\mathbf{x}}_k = \mathbf{M}\mathbf{z} = [\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{z}$$

It is worth noting that this is the same solution obtained for weighted least squares if we choose $\mathbf{W} = \mathbf{P}_{vv}^{-1}$. As such, since we know that \mathbf{P}_{vv}^{-1} is full rank, we can use the same arguments as before to conclude that we have a minimum variance estimate.

However, another way to prove this is to consider a different estimate and to show that it produces a value of the cost function that is not smaller than the one produced by our current estimate.

Consider then, without loss of generality, an alternate estimate of the form

$$\tilde{\mathbf{x}}_k = \hat{\mathbf{x}}_k + \mathbf{B}\mathbf{z}$$

where \mathbf{B} is not a null matrix. By construction, this estimate is also a linear estimate.

Is this new estimate also unbiased? To see if it is, we can take the expected value of the new estimate and apply the known relationships to find that

$$\begin{aligned}
 \mathrm{E}\{\tilde{\mathbf{x}}_k\} &= \mathrm{E}\{\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{z}\} \\
 \text{separate terms} &= \mathrm{E}\{\hat{\mathbf{x}}_k\} + \mathrm{E}\{\mathbf{B}\mathbf{z}\} \\
 \text{unbiased orig. estimate} &= \mathbf{x}_k + \mathrm{E}\{\mathbf{B}\mathbf{z}\} \\
 \text{meas. model} &= \mathbf{x}_k + \mathrm{E}\{\mathbf{B}[\mathbf{H}\mathbf{x}_k + \mathbf{v}]\} \\
 \text{separate terms} &= \mathbf{x}_k + \mathrm{E}\{\mathbf{B}\mathbf{H}\mathbf{x}_k\} + \mathrm{E}\{\mathbf{B}\mathbf{v}\} \\
 \text{pull out deterministic terms} &= \mathbf{x}_k + \mathbf{B}\mathbf{H}\mathrm{E}\{\mathbf{x}_k\} + \mathbf{B}\mathrm{E}\{\mathbf{v}\} \\
 \text{unbiased est. and zero-mean noise} &= \mathbf{x}_k + \mathbf{B}\mathbf{H}\mathbf{x}_k
 \end{aligned}$$

That is, we find that $\mathbf{B}\mathbf{H} = \mathbf{0}_{n \times n}$ is the required condition on \mathbf{B} for this new estimate to be unbiased. We have already excluded the case that $\mathbf{B} = \mathbf{0}_{n \times m}$, and we know that \mathbf{H} is full rank, so it must be that the matrix \mathbf{B} cannot be full rank

in order to have an unbiased estimate.

Our new estimate is linear, and it can be made to be unbiased, but is it a minimum variance estimate?

To see if it is, let's look at the error in this new estimate, which is defined as

$$\mathbf{e}_{\tilde{x},k} = \mathbf{x}_k - \tilde{\mathbf{x}}_k$$

and take the expected value of the error

$$\mathrm{E}\{\mathbf{e}_{\tilde{x},k}\} = \mathrm{E}\{\mathbf{x}_k\} - \mathrm{E}\{\tilde{\mathbf{x}}_k\} = -\mathbf{B}\mathbf{H}\mathbf{x}_k = \mathbf{0}$$

We can also write the error also in terms of the state estimate and its expectation as

$$\begin{aligned}\mathbf{x}_k - \tilde{\mathbf{x}}_k &= [\mathbf{x}_k - \tilde{\mathbf{x}}_k] + [\mathrm{E}\{\tilde{\mathbf{x}}_k\} - \mathrm{E}\{\tilde{\mathbf{x}}_k\}] \\ &= [\mathrm{E}\{\tilde{\mathbf{x}}_k\} - \tilde{\mathbf{x}}_k] \\ &= -[\tilde{\mathbf{x}}_k - \mathrm{E}\{\tilde{\mathbf{x}}_k\}]\end{aligned}$$

such that the covariance of the estimation error for our new estimate is

$$\mathbf{P}_{\tilde{x}\tilde{x},k} = \text{E}\{\tilde{\mathbf{x}}_k - \text{E}\{\tilde{\mathbf{x}}_k\}\}[\tilde{\mathbf{x}}_k - \text{E}\{\tilde{\mathbf{x}}_k\}]^T\}$$

Now, we begin manipulating this expression with the goal of relating it to $\mathbf{P}_{xx,k}$. First, substitute for the new estimate in terms of the original estimate and substitute for the expected value of the new estimate without restricting \mathbf{B} , giving

$$\mathbf{P}_{\tilde{x}\tilde{x},k} = \text{E}\{[(\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{z}) - (\mathbf{x}_k + \mathbf{B}\mathbf{H}\mathbf{x}_k)][(\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{z}) - (\mathbf{x}_k + \mathbf{B}\mathbf{H}\mathbf{x}_k)]^T\}$$

Collect similar terms together and recall that the observations are assumed to follow $\mathbf{z} = \mathbf{H}\mathbf{x}_k + \mathbf{v}$ to get

$$\mathbf{P}_{\tilde{x}\tilde{x},k} = \text{E}\{[(\hat{\mathbf{x}}_k - \mathbf{x}_k) + \mathbf{B}\mathbf{v}][(\hat{\mathbf{x}}_k - \mathbf{x}_k) + \mathbf{B}\mathbf{v}]^T\}$$

Expand the product and distribute the expectation to each of the resulting terms

$$\begin{aligned} \mathbf{P}_{\tilde{x}\tilde{x},k} &= \text{E}\{\hat{\mathbf{x}}_k - \mathbf{x}_k\}[\hat{\mathbf{x}}_k - \mathbf{x}_k]^T + \text{E}\{\hat{\mathbf{x}}_k - \mathbf{x}_k\}\mathbf{v}^T\mathbf{B}^T \\ &\quad + \text{E}\{\mathbf{B}\mathbf{v}[\hat{\mathbf{x}}_k - \mathbf{x}_k]^T\} + \text{E}\{\mathbf{B}\mathbf{v}\mathbf{v}^T\mathbf{B}^T\} \end{aligned}$$

The first term is simply the error covariance of the original estimate

$$\mathbf{P}_{\tilde{x}\tilde{x},k} = \mathbf{P}_{xx,k} + \mathrm{E}\{\hat{\mathbf{x}}_k - \mathbf{x}_k\} \mathbf{v}^T \mathbf{B}^T + \mathrm{E}\{\mathbf{B}\mathbf{v}[\hat{\mathbf{x}}_k - \mathbf{x}_k]^T\} + \mathrm{E}\{\mathbf{B}\mathbf{v}\mathbf{v}^T \mathbf{B}^T\}$$

We need to sort out the remaining three terms to determine if our new estimate produces a smaller or larger covariance than our original estimate.

The matrix \mathbf{B} is deterministic, so we pull it out of the expectations, and we apply the definition of the measurement noise covariance matrix, leaving us with

$$\mathbf{P}_{\tilde{x}\tilde{x},k} = \mathbf{P}_k + \mathrm{E}\{\hat{\mathbf{x}}_k - \mathbf{x}_k\} \mathbf{v}^T \mathbf{B}^T + \mathbf{B} \mathrm{E}\{\mathbf{v}[\hat{\mathbf{x}}_k - \mathbf{x}_k]^T\} + \mathbf{B} \mathbf{P}_{vv} \mathbf{B}^T$$

Now we just need to determine what the form of the middle two expectations is. Note that they are just transposes of one another, so it will suffice to focus on the second of the two.

$$\begin{aligned} \mathrm{E}\{\mathbf{v}[\hat{\mathbf{x}}_k - \mathbf{x}_k]^T\} &= \mathrm{E}\{\mathbf{v}[\mathbf{P}_{xx,k} \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{z} - \mathbf{x}_k]^T\} \\ (\text{meas. model}) &= \mathrm{E}\{\mathbf{v}[\mathbf{P}_{xx,k} \mathbf{H}^T \mathbf{P}_{vv}^{-1} (\mathbf{Hx}_k + \mathbf{v}) - \mathbf{x}_k]^T\} \\ (\text{distribute}) &= \mathrm{E}\{\mathbf{v}[\mathbf{P}_{xx,k} \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{Hx}_k + \mathbf{P}_{xx,k} \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{v} - \mathbf{x}_k]^T\} \end{aligned}$$

$$\begin{aligned}
(\text{cov. def.}) \quad &= E\{\mathbf{v}[\mathbf{x}_k + \mathbf{P}_{xx,k}\mathbf{H}^T\mathbf{P}_{vv}^{-1}\mathbf{v} - \mathbf{x}_k]^T\} \\
(\text{cancel}) \quad &= E\{\mathbf{v}[\mathbf{P}_{xx,k}\mathbf{H}^T\mathbf{P}_{vv}^{-1}\mathbf{v}]^T\} \\
(\text{rearrange}) \quad &= E\{\mathbf{v}\mathbf{v}^T\}\mathbf{P}_{vv}^{-1}\mathbf{H}\mathbf{P}_{xx,k} \\
(\text{noise cov.}) \quad &= \mathbf{P}_{vv}\mathbf{P}_{vv}^{-1}\mathbf{H}\mathbf{P}_{xx,k} \\
(\text{cancel}) \quad &= \mathbf{H}\mathbf{P}_{xx,k}
\end{aligned}$$

Applying the preceding result to the covariance for the alternative update yields

$$\mathbf{P}_{\tilde{x}\tilde{x},k} = \mathbf{P}_{xx,k} + \mathbf{P}_{xx,k}\mathbf{H}^T\mathbf{B}^T + \mathbf{B}\mathbf{H}\mathbf{P}_{xx,k} + \mathbf{B}\mathbf{P}_{vv}\mathbf{B}^T$$

For the new update to be unbiased we required that $\mathbf{B}\mathbf{H} = \mathbf{0}$, so we arrive at the covariance of the new update

$$\mathbf{P}_{\tilde{x}\tilde{x},k} = \mathbf{P}_{xx,k} + \mathbf{B}\mathbf{P}_{vv}\mathbf{B}^T$$

Since \mathbf{P}_{vv} is positive definite and \mathbf{B} is not full rank, $\mathbf{B}\mathbf{P}_{vv}\mathbf{B}^T \geq \mathbf{0}$ and $\mathbf{P}_{\tilde{x}\tilde{x},k} \geq \mathbf{P}_{xx,k}$. That is, the alternate state estimate has a covariance that is no smaller than that of the original estimate. Our original estimate is therefore a minimum variance estimate.

The development of the linear, unbiased, minimum variance estimator is now complete.

To summarize, the estimate is given by

$$\hat{\mathbf{x}}_k = [\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1} \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{z}$$

and the covariance of this estimate is

$$\mathbf{P}_{xx,k} = [\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1}$$

LUMVE produces an approach to least-squares estimation that allows us to weight each measurement by our statistical confidence in each measurement. It also tells us how confident we are in the estimate of our state.

There is no appearance of a prior estimate (or its associated statistical confidence), but we had a method for including a prior estimate in weighted least squares. We can follow the same approach used for weighted least squares to include prior information in LUMVE.

To do so, we define a “zeroth” measurement to be a measurement of the state

$$\mathbf{z}_0 = \mathbf{x}_k + \mathbf{n}$$

We have used a “measurement noise” represented by \mathbf{n} to denote that this is not the same as the standard measurement noise. We assume that \mathbf{n} is zero mean with covariance $\bar{\mathbf{P}}_{xx,k}$. Additionally, we represent this zeroth measurement symbolically as $\bar{\mathbf{x}}_k$.

Given the data, measurement mapping, and measurement noise as \mathbf{z} , \mathbf{H} , and \mathbf{v} , respectively, from the LUMVE development, we define augmented data, measurement mapping, and measurement noise as

$$\bar{\mathbf{z}} = \begin{bmatrix} \bar{\mathbf{x}}_k \\ \mathbf{z} \end{bmatrix}, \quad \bar{\mathbf{H}} = \begin{bmatrix} \mathbf{I}_n \\ \mathbf{H} \end{bmatrix}, \quad \text{and} \quad \bar{\mathbf{v}} = \begin{bmatrix} \mathbf{n} \\ \mathbf{v} \end{bmatrix}$$

Since \mathbf{n} and \mathbf{v} are both zero mean, it follows that $\bar{\mathbf{v}}$ is also zero mean. We take the prior data to be uncorrelated with the measurement data, such that the covariance of $\bar{\mathbf{v}}$ is

$$\mathbf{P}_{\bar{v}\bar{v}} = \begin{bmatrix} \bar{\mathbf{P}}_{xx,k} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_{vv} \end{bmatrix}$$

Using $\bar{\mathbf{z}}$, $\bar{\mathbf{H}}$, and $\bar{\mathbf{R}}$, LUMVE gives the estimated state and its associated covariance as

$$\hat{\mathbf{x}}_k = [\bar{\mathbf{H}}^T \mathbf{P}_{\bar{v}\bar{v}}^{-1} \bar{\mathbf{H}}]^{-1} \bar{\mathbf{H}}^T \mathbf{P}_{\bar{v}\bar{v}}^{-1} \bar{\mathbf{z}}$$

$$\mathbf{P}_{xx,k} = [\bar{\mathbf{H}}^T \mathbf{P}_{\bar{v}\bar{v}}^{-1} \bar{\mathbf{H}}]^{-1}$$

Now we want to substitute for the augmented quantities and express this estimate in terms of the original variables.

$$\bar{\mathbf{H}}^T \bar{\mathbf{P}}_{\bar{v}\bar{v}}^{-1} \bar{\mathbf{H}} = \bar{\mathbf{P}}_{xx,k}^{-1} + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}$$

$$\bar{\mathbf{H}}^T \bar{\mathbf{P}}_{\bar{v}\bar{v}}^{-1} \bar{\mathbf{z}} = \bar{\mathbf{P}}_{xx,k}^{-1} \bar{\mathbf{x}}_k + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{z}$$

Putting everything together in the base formulation of our estimator equations, the LUMVE with (uncorrelated) prior information and its associated covariance are given by

$$\hat{\mathbf{x}}_k = [\bar{\mathbf{P}}_{xx,k}^{-1} + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1} [\bar{\mathbf{P}}_{xx,k}^{-1} \bar{\mathbf{x}}_k + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{z}]$$

$$\mathbf{P}_{xx,k} = [\bar{\mathbf{P}}_{xx,k}^{-1} + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1}$$

This is what is commonly referred to as the **batch processor**, because it collects all of the data together and processes them all simultaneously.

In the case where there is no prior information available, one simply sets $\bar{\mathbf{P}}_{xx,k}^{-1} = \mathbf{0}_{n \times n}$ and $\bar{\mathbf{P}}_{xx,k}^{-1} \bar{\mathbf{x}}_k = \mathbf{0}_n$. These quantities are commonly called the “information matrix” and “information state,” respectively.

2.5.1 Example: Revisiting the Robot Problem

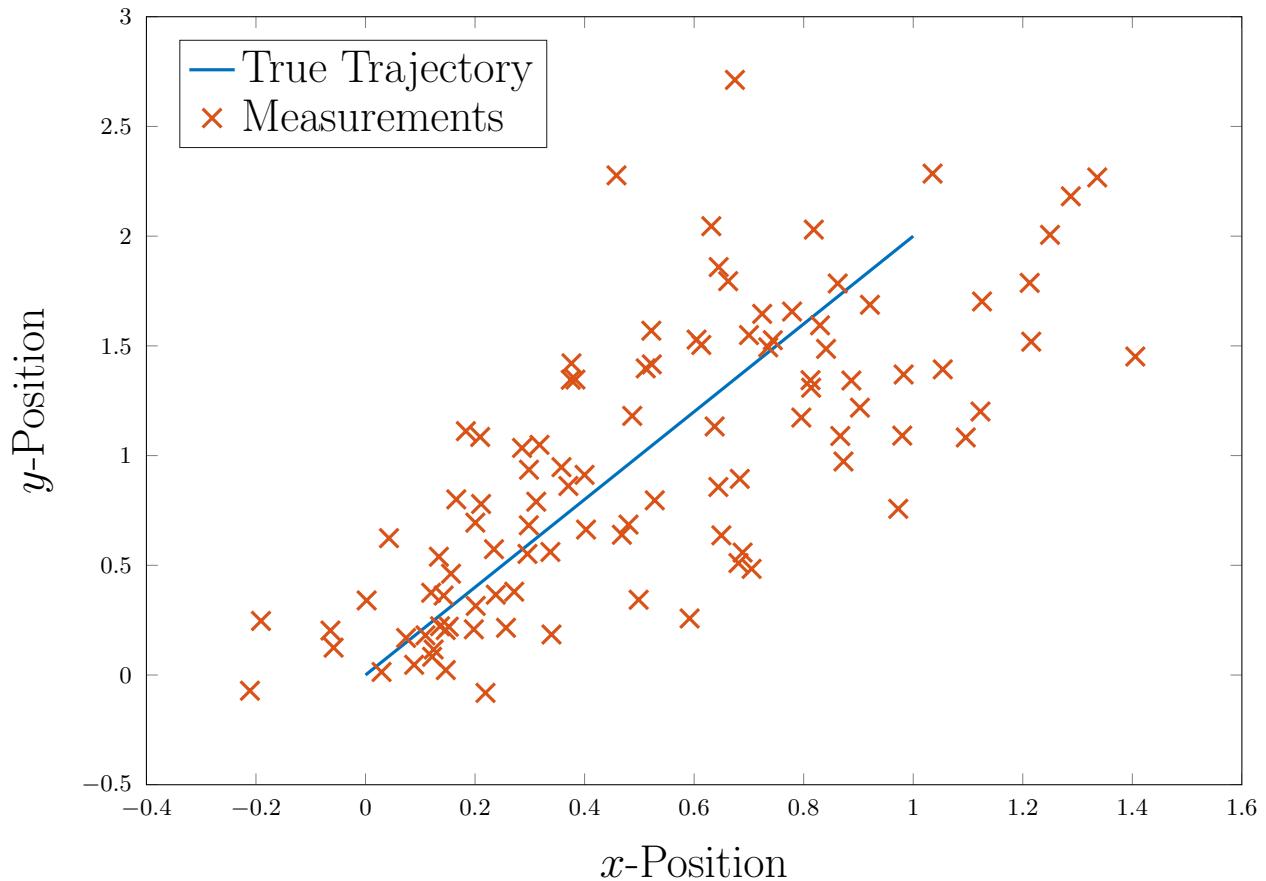
Let's revisit our robot tracking problem. This time, we want to apply the least squares, weighted least squares, and LUMVE estimates to see any differences.

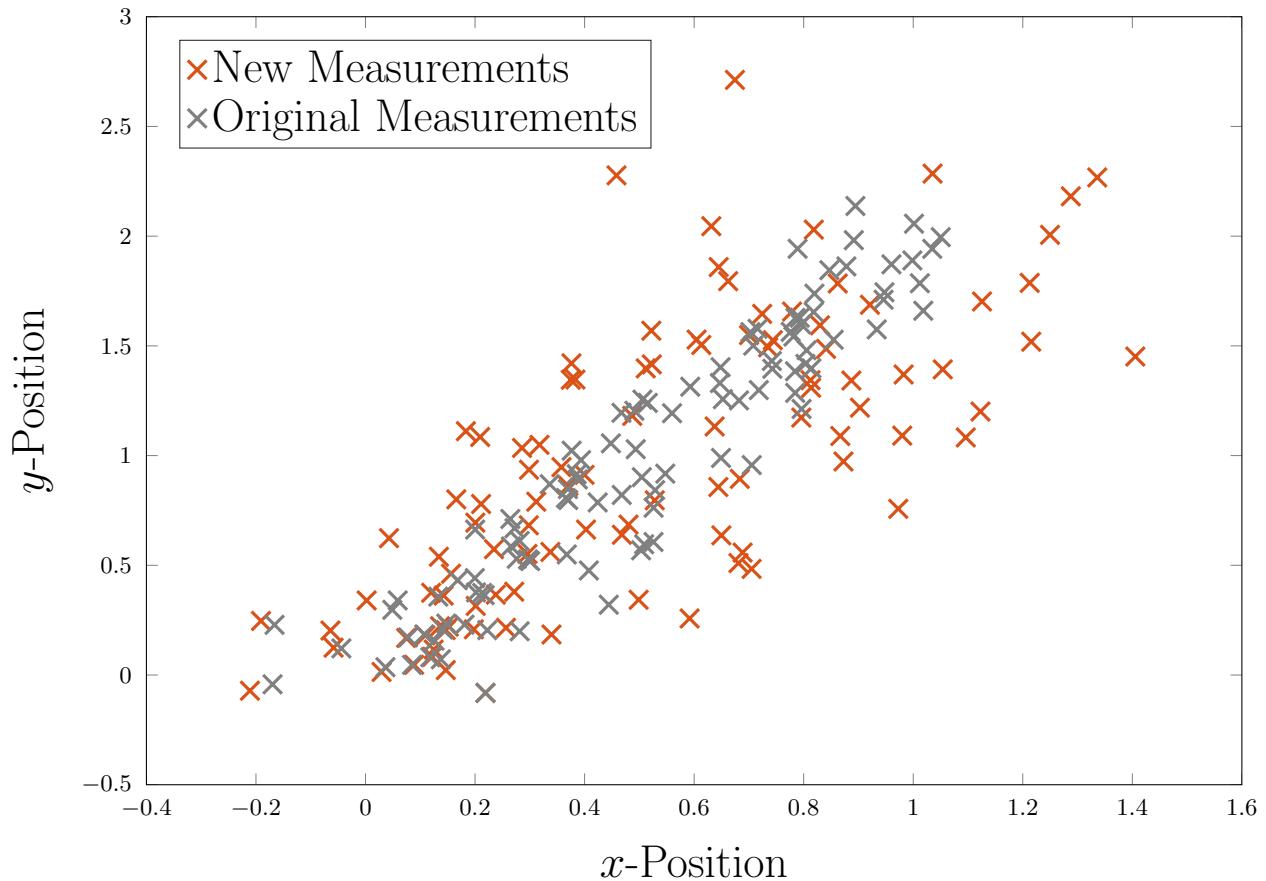
The configuration is identical to the previous time we dealt with the robot problem, except now our measurements will get progressively less accurate.

We are still assuming that the robot moves in a plane under a constant velocity model, starting from the origin.

The measurements are still formed from the position of the robot with noise taken from a Gaussian distribution. As time progresses, the noise increases from a standard deviation of 0.1 meters to 0.4 meters.

```
% Create measurements of position
sig = linspace(0.1,0.4,length(tv));
z    = zeros(2,length(tv));
for k = 1:length(tv)
    z(:,k) = X(k,1:2)' + sig(k)*randn(2,1);
end
```





For the least squares estimate, we do not have any mechanism to deal with the changing measurement noise, so this approach remains the same.

For the weighted least squares estimate, we need to select the weighting matrices \mathbf{W}_i that accompany the observations.

These weighting matrices only reflect our rough confidence in the measurements and are not necessarily selected based on statistics.

Therefore, we arbitrarily claim that the first half of the measurements are twice as believable as the second half of the measurements.

For LUMVE, we use the actual time-varying statistics of the measurement noise to accumulate our \mathbf{P}_{vv} matrix from the $\mathbf{P}_{vv,i}$ matrices, which are chosen as $\mathbf{P}_{vv,i} = \sigma_i^2 \mathbf{I}_2$.

Additionally, we assume that the measurement noise is white, such that the off-diagonal terms in \mathbf{P}_{vv} are zero.

Let's look at how the accumulation loop would be coded in MATLAB:

```

% Assemble the concatenated terms
Z = []; H = []; W = []; PvV = [];
for k = 1:length(tv)
    % time at kth observation
    tk = tv(k);

    % concatenate measurements
    Z = [Z;z(1,k);z(2,k)];

    % concatenate measurement-mapping matrices
    Htilde = [1,0,0,0;0,1,0,0];
    Phik0 = [1,0,tk-t0,0;0,1,0,tk-t0;0,0,1,0;0,0,0,1];
    H = [H;Htilde*Phik0];

    % concatenate weighting matrices for WLS
    if k < round(length(tv)/2)
        W = blkdiag(W,2.0*eye(2));
    else
        W = blkdiag(W,1.0*eye(2));
    end

    % concatenate LUMVE inverse weighting matrix
    PvV = blkdiag(PvV,sig(k)^2*eye(2));
end

```

Now, we can apply our three estimation techniques.

```
% Least-squares estimate  
xhatLS = (H'*H)\(H'*Z);  
  
% Weighted least-squares estimate  
xhatWLS = (H'*W*H)\(H'*W*Z);  
  
% LUMVE estimate  
xhatLUMVE = (H'*inv(Pvv)*H)\(H'*inv(Pvv)*Z);
```

The results from the estimates are

$$\hat{x}_0^{(\text{LS})} = \begin{bmatrix} 0.0182 \\ -0.0533 \\ 0.1003 \\ 0.2053 \end{bmatrix} \quad \hat{x}_0^{(\text{WLS})} = \begin{bmatrix} 0.0189 \\ -0.0491 \\ 0.0996 \\ 0.2040 \end{bmatrix} \quad \hat{x}_0^{(\text{LUMVE})} = \begin{bmatrix} 0.0244 \\ -0.0241 \\ 0.0989 \\ 0.1986 \end{bmatrix}$$

2.6 Sequential Least Squares

The batch processor gives us a method for accumulating all of the data and processing them together in order to formulate a statistical estimate of the system state at some

fixed time. As a reminder, the batch processor with the inclusion of prior information is

$$\hat{\mathbf{x}}_k = [\bar{\mathbf{P}}_{xx,k}^{-1} + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1} [\bar{\mathbf{P}}_{xx,k}^{-1} \bar{\mathbf{x}}_k + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{z}]$$

$$\mathbf{P}_k = [\bar{\mathbf{P}}_{xx,k}^{-1} + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1}$$

where $\hat{\mathbf{x}}_k$ is our state estimate and $\mathbf{P}_{xx,k}$ is the covariance of the estimation error, which provides us with a measure of the confidence we have in our estimate.

Note that there are several matrix inverses required to compute the estimate; in particular, the inverses of \mathbf{P}_{vv} , $\bar{\mathbf{P}}_{xx,k}$, $(\bar{\mathbf{P}}_{xx,k}^{-1} + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H})$ are required.

For a system with n states, $\mathbf{H} \in \mathbb{R}^{m \times n}$ implies that two $n \times n$ inverses are necessary. If the state dimension is large, these inverse calculations may be quite cumbersome.

Additionally, we require \mathbf{P}_{vv}^{-1} , which is an $m \times m$ inverse. For the least squares techniques to work, $m > n$ means that the computation of \mathbf{P}_{vv}^{-1} may also be challenging. Inversion of \mathbf{P}_{vv} , however, is easily overcome when the measurements are uncorrelated.

If the measurements are uncorrelated, it is straightforward to observe that

$$\mathbf{P}_{vv} = \begin{bmatrix} \mathbf{P}_{vv,1} & \mathbf{0}_{q \times q} & \cdots \\ \mathbf{0}_{q \times q} & \mathbf{P}_{vv,2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

where $\mathbf{P}_{vv,i} \in \mathbb{R}^{q \times q}$ (provided that each measurement is q -dimensional).

Even if the individual measurements are not of the same dimension, the dimension of an individual measurement is substantially smaller than the dimension of the concatenated set of measurements. If the concatenated measurement noise covariance matrix is block diagonal (which is the case for uncorrelated measurement noises), then the inverse is simply given by the block diagonal matrix

$$\mathbf{P}_{vv}^{-1} = \begin{bmatrix} \mathbf{P}_{vv,1}^{-1} & \mathbf{0}_{q \times q} & \cdots \\ \mathbf{0}_{q \times q} & \mathbf{P}_{vv,2}^{-1} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

and requires only the smaller inverse computations.

The objective of the sequential least squares method is to reduce the computational burden associated with the inverse of large-scale matrices in favor of inverting smaller matrices.

First, let us assume that we have applied the batch processor to a set of data to arrive at an estimate and its covariance at time t_j :

$$\hat{\mathbf{x}}_j = [\bar{\mathbf{P}}_{xx,j}^{-1} + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1} [\bar{\mathbf{P}}_{xx,j}^{-1} \bar{\mathbf{x}}_j + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{z}]$$

$$\mathbf{P}_{xx,j} = [\bar{\mathbf{P}}_{xx,j}^{-1} + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1}$$

Further, let us assume that we then acquire a new measurement at time $t_k > t_j$ that is of the form

$$\mathbf{z}_k = \tilde{\mathbf{H}}_k \mathbf{x}_k + \mathbf{v}_k$$

where \mathbf{v}_k is zero mean with covariance $\mathbf{P}_{vv,k}$.

In order to apply the batch processor, we would have to append this new data to the previous data set (along with the measurement mapping and covariance) and then recompute the **entire** state estimate, which means that we have lost all of the work we did

in computing the estimate $\hat{\mathbf{x}}_j$. Clearly, this is not an ideal arrangement.

Instead, it would be better if we could *propagate* our previous estimate and the covariance to time t_k and then *update* our state estimate and covariance. If we continue to acquire more data, we would then continue the cycle of propagating and updating our estimate and its covariance. This recursive technique is the idea behind a “sequential” form of LUMVE.

We will first consider the task of propagating our estimate and its associated covariance. To do so, recall that our dynamical system is of the form

$$\mathbf{x}_k = \Phi(t_k, t_j) \mathbf{x}_j$$

This temporal mapping of states can also be used to propagate the estimate obtained at time t_j to an estimate at time t_k via

$$\bar{\mathbf{x}}_k = \Phi(t_k, t_j) \hat{\mathbf{x}}_j$$

where $\hat{\mathbf{x}}_j$ is our state estimate at time t_j and $\bar{\mathbf{x}}_k$ is the propagated state estimate at time t_k , otherwise known as part of the prior information at t_k .

We also need to find a way to propagate the covariance matrix. As such, let's first consider how the estimation error propagates through the dynamical system. Define the estimation error at times t_j and t_k , respectively, as

$$\mathbf{e}_{x,j} = \mathbf{x}_j - \hat{\mathbf{x}}_j \quad \text{and} \quad \mathbf{e}_{x,k} = \mathbf{x}_k - \bar{\mathbf{x}}_k$$

Pre-multiply the state estimation error at time t_j by the state transition matrix:

$$\begin{aligned} \Phi(t_k, t_j) \mathbf{e}_{x,j} &= \Phi(t_k, t_j) \mathbf{x}_j - \Phi(t_k, t_j) \hat{\mathbf{x}}_j \\ &= \mathbf{x}_k - \bar{\mathbf{x}}_k \\ &= \mathbf{e}_{x,k} \end{aligned}$$

By definition, the covariance of the state estimation error at time t_k is given by

$$\bar{\mathbf{P}}_{xx,k} = \mathbb{E}\{\mathbf{e}_{x,k} \mathbf{e}_{x,k}^T\}$$

and since the error evolves as

$$\mathbf{e}_{x,k} = \Phi(t_k, t_j) \mathbf{e}_{x,j}$$

it follows that the covariance propagates through time as

$$\begin{aligned}
\bar{\mathbf{P}}_{xx,k} &= \text{E}\{\mathbf{e}_{x,k}\mathbf{e}_{x,k}^T\} \\
&= \text{E}\{\Phi(t_k, t_j)\mathbf{e}_{x,j}\mathbf{e}_{x,j}^T\Phi^T(t_k, t_j)\} \\
&= \Phi(t_k, t_j)\text{E}\{\mathbf{e}_{x,j}\mathbf{e}_{x,j}^T\}\Phi^T(t_k, t_j) \\
&= \Phi(t_k, t_j)\mathbf{P}_{xx,j}\Phi^T(t_k, t_j)
\end{aligned}$$

Therefore, given the state estimate and covariance at time t_j , the propagated state estimate and covariance at time t_k , which are the prior information at t_k , are

$$\begin{aligned}
\bar{\mathbf{x}}_k &= \Phi(t_k, t_j)\hat{\mathbf{x}}_j \\
\bar{\mathbf{P}}_{xx,k} &= \Phi(t_k, t_j)\mathbf{P}_{xx,j}\Phi^T(t_k, t_j)
\end{aligned}$$

Now, we proceed to the update stage of the sequential estimation algorithm.

At time t_k , we have the additional measurement

$$\mathbf{z}_k = \tilde{\mathbf{H}}_k \mathbf{x}_k + \mathbf{v}_k$$

along with the prior information contained in $\bar{\mathbf{x}}_k$ and $\bar{\mathbf{P}}_{xx,k}$.

The fused estimate is then given by LUMVE as

$$\hat{\mathbf{x}}_k = [\bar{\mathbf{P}}_{xx,k}^{-1} + \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \tilde{\mathbf{H}}_k]^{-1} [\bar{\mathbf{P}}_{xx,k}^{-1} \bar{\mathbf{x}}_k + \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \mathbf{z}_k]$$

$$\mathbf{P}_{xx,k} = [\bar{\mathbf{P}}_{xx,k}^{-1} + \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \tilde{\mathbf{H}}_k]^{-1}$$

It is important to note that this application of LUMVE is only for fusing the prior information with the single measurement \mathbf{z}_k , as opposed to previous applications where we collected the measurement data and concatenated the terms.

To circumvent the need for $n \times n$ matrix inverses, consider the Matrix Inversion Lemma (MIL), given by

Matrix Inversion Lemma

$$[\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{V}]^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}[\mathbf{V}\mathbf{A}^{-1}\mathbf{U} + \mathbf{C}^{-1}]^{-1}\mathbf{V}\mathbf{A}^{-1}$$

To apply the MIL to our given problem, let

$$\mathbf{A} = \bar{\mathbf{P}}_{xx,k}^{-1}, \quad \mathbf{U} = \tilde{\mathbf{H}}_k^T, \quad \mathbf{C} = \mathbf{P}_{vv,k}^{-1}, \quad \text{and} \quad \mathbf{V} = \tilde{\mathbf{H}}_k$$

such that the leading matrix inverse in both of the LUMVE update equations can be written as

$$[\bar{\mathbf{P}}_{xx,k}^{-1} + \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \tilde{\mathbf{H}}_k]^{-1} = \bar{\mathbf{P}}_{xx,k} - \bar{\mathbf{P}}_{xx,k} \tilde{\mathbf{H}}_k^T [\tilde{\mathbf{H}}_k \bar{\mathbf{P}}_{xx,k} \tilde{\mathbf{H}}_k^T + \mathbf{P}_{vv,k}]^{-1} \tilde{\mathbf{H}}_k \bar{\mathbf{P}}_{xx,k}$$

Note that the MIL has converted an $n \times n$ inverse on the left-hand side to a $q \times q$ inverse on the right-hand side. Provided that $q < n$, which is often the case, this represents an easier inversion. Additionally, there are fewer inverses required.

For ease of notation, it is common to define

$$\mathbf{K}_k = \bar{\mathbf{P}}_{xx,k} \tilde{\mathbf{H}}_k^T [\tilde{\mathbf{H}}_k \bar{\mathbf{P}}_{xx,k} \tilde{\mathbf{H}}_k^T + \mathbf{P}_{vv,k}]^{-1}$$

such that

$$\begin{aligned} [\bar{\mathbf{P}}_{xx,k}^{-1} + \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \tilde{\mathbf{H}}_k]^{-1} &= \bar{\mathbf{P}}_{xx,k} - \mathbf{K}_k \tilde{\mathbf{H}}_k \bar{\mathbf{P}}_{xx,k} \\ &= [\mathbf{I}_n - \mathbf{K}_k \tilde{\mathbf{H}}_k] \bar{\mathbf{P}}_{xx,k} \end{aligned}$$

Substituting this result into the update equations gives

$$\hat{\mathbf{x}}_k = [\mathbf{I}_n - \mathbf{K}_k \tilde{\mathbf{H}}_k] \bar{\mathbf{P}}_{xx,k} [\bar{\mathbf{P}}_{xx,k}^{-1} \bar{\mathbf{x}}_k + \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \mathbf{z}_k]$$

$$\mathbf{P}_{xx,k} = [\mathbf{I}_n - \mathbf{K}_k \tilde{\mathbf{H}}_k] \bar{\mathbf{P}}_{xx,k}$$

While the covariance update looks good, we still have a bit of a mess in the state estimate update. Specifically, we still see a matrix inverse for a matrix with the same dimension as the state.

To clean this up, we first note that

$$\begin{aligned}\hat{\mathbf{x}}_k &= [\mathbf{I}_n - \mathbf{K}_k \tilde{\mathbf{H}}_k] \bar{\mathbf{P}}_{xx,k} [\bar{\mathbf{P}}_{xx,k}^{-1} \bar{\mathbf{x}}_k + \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \mathbf{z}_k] \\ \text{distribute} &= [\mathbf{I}_n - \mathbf{K}_k \tilde{\mathbf{H}}_k] \bar{\mathbf{P}}_{xx,k} \bar{\mathbf{P}}_{xx,k}^{-1} \bar{\mathbf{x}}_k + [\mathbf{I}_n - \mathbf{K}_k \tilde{\mathbf{H}}_k] \bar{\mathbf{P}}_{xx,k} \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \mathbf{z}_k \\ \text{cancel} &= [\mathbf{I}_n - \mathbf{K}_k \tilde{\mathbf{H}}_k] \bar{\mathbf{x}}_k + [\mathbf{I}_n - \mathbf{K}_k \tilde{\mathbf{H}}_k] \bar{\mathbf{P}}_{xx,k} \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \mathbf{z}_k \\ \text{upd. cov.} &= [\mathbf{I}_n - \mathbf{K}_k \tilde{\mathbf{H}}_k] \bar{\mathbf{x}}_k + \mathbf{P}_{xx,k} \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \mathbf{z}_k \\ \text{distribute} &= \bar{\mathbf{x}}_k - \mathbf{K}_k \tilde{\mathbf{H}}_k \bar{\mathbf{x}}_k + \mathbf{P}_{xx,k} \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \mathbf{z}_k\end{aligned}$$

The first two terms are ok, but what about that last term? We need to find another way to express the matrix in this last term to obtain an expression that omits the inverse.

To find this needed relationship, we return to the LUMVE covariance equation and work from there, as follows:

$$\mathbf{P}_{xx,k}^{-1} = \bar{\mathbf{P}}_{xx,k}^{-1} + \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \tilde{\mathbf{H}}_k$$

1. Pre-multiply by $\mathbf{P}_{xx,k}$

$$\mathbf{I}_n = \mathbf{P}_{xx,k} \bar{\mathbf{P}}_{xx,k}^{-1} + \mathbf{P}_{xx,k} \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \tilde{\mathbf{H}}_k$$

2. Post-multiply by $\bar{\mathbf{P}}_{xx,k}$

$$\bar{\mathbf{P}}_{xx,k} = \mathbf{P}_{xx,k} + \mathbf{P}_{xx,k} \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \tilde{\mathbf{H}}_k \bar{\mathbf{P}}_{xx,k}$$

3. Post-multiply by $\tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1}$

$$\bar{\mathbf{P}}_{xx,k} \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} = \mathbf{P}_{xx,k} \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} + \mathbf{P}_{xx,k} \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \tilde{\mathbf{H}}_k \bar{\mathbf{P}}_{xx,k} \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1}$$

4. Factor out $\mathbf{P}_{xx,k}\tilde{\mathbf{H}}_k^T\mathbf{P}_{vv,k}^{-1}$ on the right-hand side

$$\bar{\mathbf{P}}_{xx,k}\tilde{\mathbf{H}}_k^T\mathbf{P}_{vv,k}^{-1} = \mathbf{P}_{xx,k}\tilde{\mathbf{H}}_k^T\mathbf{P}_{vv,k}^{-1}[\mathbf{I}_q + \tilde{\mathbf{H}}_k\bar{\mathbf{P}}_{xx,k}\tilde{\mathbf{H}}_k^T\mathbf{P}_{vv,k}^{-1}]$$

5. Pull $\mathbf{P}_{vv,k}^{-1}$ out of the bracketed term

$$\bar{\mathbf{P}}_{xx,k}\tilde{\mathbf{H}}_k^T\mathbf{P}_{vv,k}^{-1} = \mathbf{P}_{xx,k}\tilde{\mathbf{H}}_k^T\mathbf{P}_{vv,k}^{-1}[\mathbf{P}_{vv,k} + \tilde{\mathbf{H}}_k\bar{\mathbf{P}}_{xx,k}\tilde{\mathbf{H}}_k^T]\mathbf{P}_{vv,k}^{-1}$$

6. Post-multiply by $\mathbf{P}_{vv,k}$

$$\bar{\mathbf{P}}_{xx,k}\tilde{\mathbf{H}}_k^T = \mathbf{P}_{xx,k}\tilde{\mathbf{H}}_k^T\mathbf{P}_{vv,k}^{-1}[\mathbf{P}_{vv,k} + \tilde{\mathbf{H}}_k\bar{\mathbf{P}}_{xx,k}\tilde{\mathbf{H}}_k^T]$$

7. Solve for $\mathbf{P}_{xx,k}\tilde{\mathbf{H}}_k^T\mathbf{P}_{vv,k}^{-1}$

$$\mathbf{P}_{xx,k}\tilde{\mathbf{H}}_k^T\mathbf{P}_{vv,k}^{-1} = \bar{\mathbf{P}}_{xx,k}\tilde{\mathbf{H}}_k^T[\tilde{\mathbf{H}}_k\bar{\mathbf{P}}_{xx,k}\tilde{\mathbf{H}}_k^T + \mathbf{P}_{vv,k}]^{-1}$$

8. Recall the definition of \mathbf{K}_k

$$\mathbf{P}_{xx,k}\tilde{\mathbf{H}}_k^T\mathbf{P}_{vv,k}^{-1} = \mathbf{K}_k$$

Now we have the expression that we need, so let's go back to the state update equation. Previously, we found that

$$\hat{\mathbf{x}}_k = \bar{\mathbf{x}}_k - \mathbf{K}_k \tilde{\mathbf{H}}_k \bar{\mathbf{x}}_k + \mathbf{P}_{xx,k} \tilde{\mathbf{H}}_k^T \mathbf{P}_{vv,k}^{-1} \mathbf{z}_k$$

We can then substitute our new equation into the last term and rearrange a few terms to get the updated estimate as

$$\hat{\mathbf{x}}_k = \bar{\mathbf{x}}_k + \mathbf{K}_k [\mathbf{z}_k - \tilde{\mathbf{H}}_k \bar{\mathbf{x}}_k]$$

Note that the bracketed term is what we've previously called the residual; that is, it is the difference between our actual measurement and our prediction of the observation. The residual is also known as the innovation, the new information being added.

The update takes the previous best estimate and adds a term that is a gain multiplied by the innovation.

A quick summary of the sequential form of LUMVE:

- at time t_j , we have (by some means) a state estimate and its covariance

$$\hat{\mathbf{x}}_j \quad \text{and} \quad \mathbf{P}_{xx,j}$$

- at time t_k , a new measurement, \mathbf{z}_k , is received; it is modeled as

$$\mathbf{z}_k = \tilde{\mathbf{H}}_k \mathbf{x}_k + \mathbf{v}_k$$

where \mathbf{v}_k is zero mean, white noise, with covariance $\mathbf{P}_{vv,k}$

- we propagate the estimate and covariance to time t_k

$$\bar{\mathbf{x}}_k = \Phi(t_k, t_j) \hat{\mathbf{x}}_j$$

$$\bar{\mathbf{P}}_{xx,k} = \Phi(t_k, t_j) \mathbf{P}_{xx,j} \Phi^T(t_k, t_j)$$

- we then update the estimate to incorporate the new measurement

$$\hat{\mathbf{x}}_k = \bar{\mathbf{x}}_k + \mathbf{K}_k [\mathbf{z}_k - \tilde{\mathbf{H}}_k \bar{\mathbf{x}}_k]$$

$$\mathbf{P}_{xx,k} = [\mathbf{I}_n - \mathbf{K}_k \tilde{\mathbf{H}}_k] \bar{\mathbf{P}}_{xx,k}$$

where

$$\mathbf{K}_k = \bar{\mathbf{P}}_{xx,k} \tilde{\mathbf{H}}_k^T [\tilde{\mathbf{H}}_k \bar{\mathbf{P}}_{xx,k} \tilde{\mathbf{H}}_k^T + \mathbf{P}_{vv,k}]^{-1}$$

- for any additional data, we cycle the propagation and update steps

This sequentialized version of LUMVE only requires inverses of matrices with the same dimension as the measurement. It does, however, make use of the assumption that the measurement noise is a white sequence.

2.6.1 Example: Revisiting the Robot Problem (again)

Let's revisit our robot tracking problem one more time. This time, we want to apply the batch and sequential LUMVE approaches to show that they yield the same results.

The configuration is identical to the previous time we dealt with the robot problem. We are still assuming that the robot moves in a plane under a constant velocity model, starting from the origin. The measurements are of the position of the robot with noise taken from a Gaussian distribution. As time progresses, the noise increases from a standard deviation of 0.1 meters to 0.4 meters.

The one difference is that we will make use of prior information, which is given by

$$\bar{\mathbf{x}}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \bar{\mathbf{P}}_{xx,0} = \begin{bmatrix} 10^2 & 0 & 0 & 0 \\ 0 & 10^2 & 0 & 0 \\ 0 & 0 & 2^2 & 0 \\ 0 & 0 & 0 & 2^2 \end{bmatrix}$$

where the units are in [m] for position, [m/s] for velocity, [m²] for position variance, [(m/s)²] for velocity variance, and [m²/s] for position-velocity covariance. In code, the initial information is given by

```
% initial information
xbar0 = [0.0;0.0;0.0;0.0];
Pbar0 = diag([10.0;10.0;2.0;2.0].^2);
```

We also have the initial true state and the continuous-time dynamics

```
% initial true state of the robot
x0 = [0;0;0.1;0.2];

% dynamics of the robot (continuous time)
F = [0,0,1,0;0,0,0,1;0,0,0,0;0,0,0,0];
```

We will simulate the truth for 10 sec at 10 Hz so that we can generate true data

```
% timing variables  
tv = (0.0:0.1:10.0)';
```

The truth is propagated using `ode45`

```
% integrate the eoms for the true object  
opts = odeset('AbsTol',1e-9,'RelTol',1e-9);  
[T,X] = ode45(@eom_robot,tv,x0,opts,F);
```

which allows measurements of the position with a time-varying measurement noise to be generated according to

```
% create measurements of position  
sig = linspace(0.1,0.4,length(tv));  
z = zeros(2,length(tv));  
for k = 1:length(tv)  
    z(:,k) = X(k,1:2)' + sig(k)*randn(2,1);  
end
```

First, we will compute the estimate using LUMVE (batch) for the estimate at t_0

```

% assemble the concatenated terms
Z = []; H = []; Pvv = [];
for k = 1:length(tv)
    % time at kth observation
    tk = tv(k);

    % concatenate measurements
    Z = [Z;z(1,k);z(2,k)];

    % concatenate measurement-mapping matrices
    Htilde = [1,0,0,0;0,1,0,0];
    Phik0 = [1,0,tk-t0,0;0,1,0,tk-t0;0,0,1,0;0,0,0,1];
    H = [H;Htilde*Phik0];

    % concatenate LUMVE inverse weighting matrix
    Pvv = blkdiag(Pvv,sig(k)^2*eye(2));
end

% LUMVE estimate
xhatB = (inv(Pbar0) + H'*inv(Pvv)*H)\(Pbar0\xbar0 + H'*(Pvv\Z));
PxxB = inv(inv(Pbar0) + H'*inv(Pvv)*H);

```

In order to have a comparison for the sequential implementation, we propagate the estimate and the covariance which are at time t_0 to the final time, time t_f , using the state

transition matrix

```
% map the LUMVE estimate to the final time  
Phif0 = [1,0,tf-t0,0;0,1,0,tf-t0;0,0,1,0;0,0,0,1];  
xhatBf = Phif0*xhatB;  
PxxBf = Phif0*PxxB*Phif0';
```

We are now ready to redo this estimation procedure using the sequential form of LUMVE, which is given by

```
% redo the process using sequential LUMVE  
tj = t0;  
xhatj = xbar0;  
Pxxj = Pbar0;  
for k = 1:length(tv);  
    % time at kth observation  
    tk = tv(k);  
  
    % kth observations  
    zk = z(:,k);  
  
    % state transition matrix from tj to tk  
    Phikj = [1,0,tk-tj,0;0,1,0,tk-tj;0,0,1,0;0,0,0,1];
```

```

% propagate estimate and covariance
xbark = Phikj*xhatj;
Pbark = Phikj*Pj*Phikj';

% update estimate and covariance
Htildek = [1,0,0,0;0,1,0,0];
Pvvk    = sig(k)^2*eye(2);
Kk      = Pbark*Htildek'/(Htildek*Pbark*Htildek' + Pvvk);
xhatk   = xbark + Kk*(zk - Htildek*xbar);
Pxxk    = Pbark - Kk*Htildek*Pbark;

% reset for next step
tj      = tk;
xhatj  = xhatk;
Pxxj   = Pxxk;
end

```

The following figures illustrate the application of the batch and sequential forms of LUMVE applied to this problem. We will look at the batch estimates mapped to the final time and plotted as constant in time. This is *only* for visualization as these estimates do not move in time.

The thing to notice is that the estimates and covariances from both forms of LUMVE

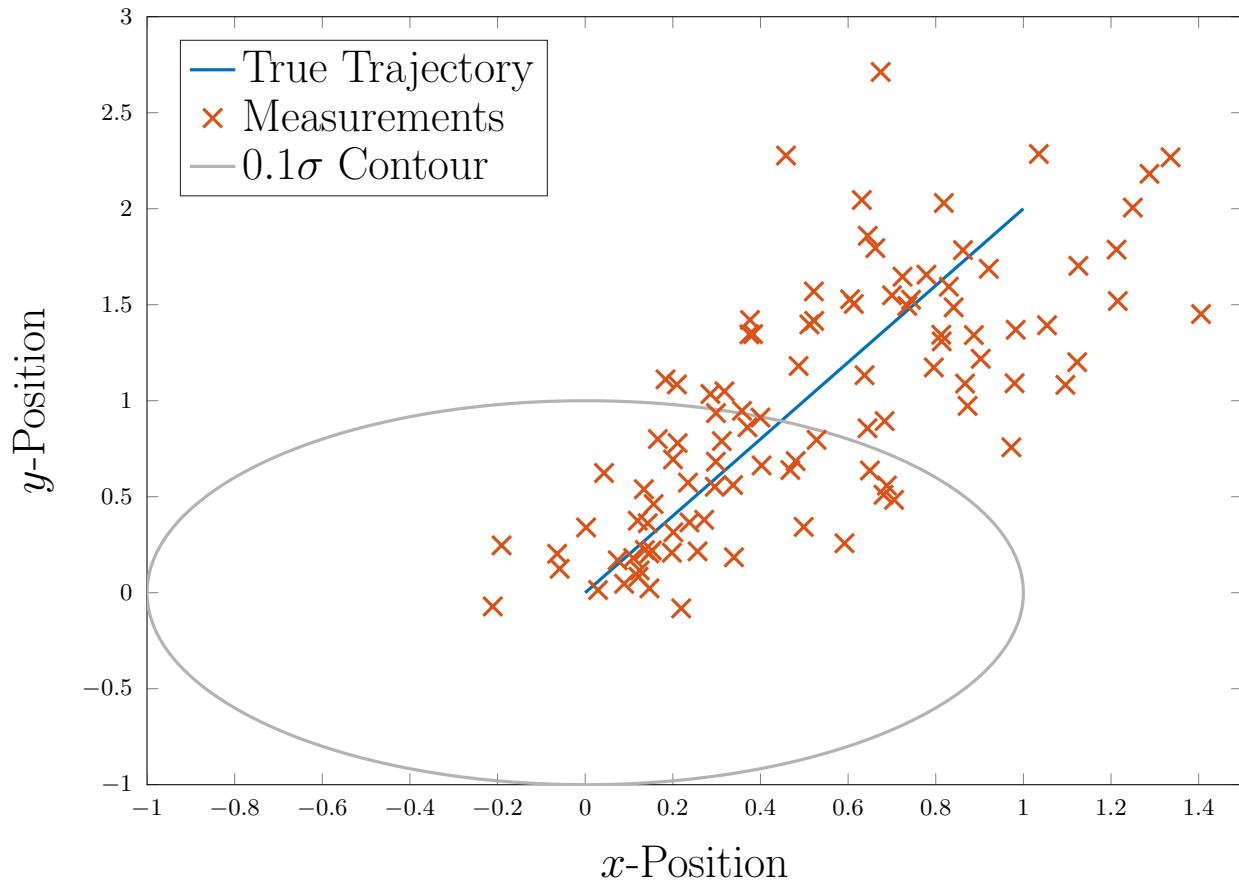
converge to the same solutions after all of the data have been processed.

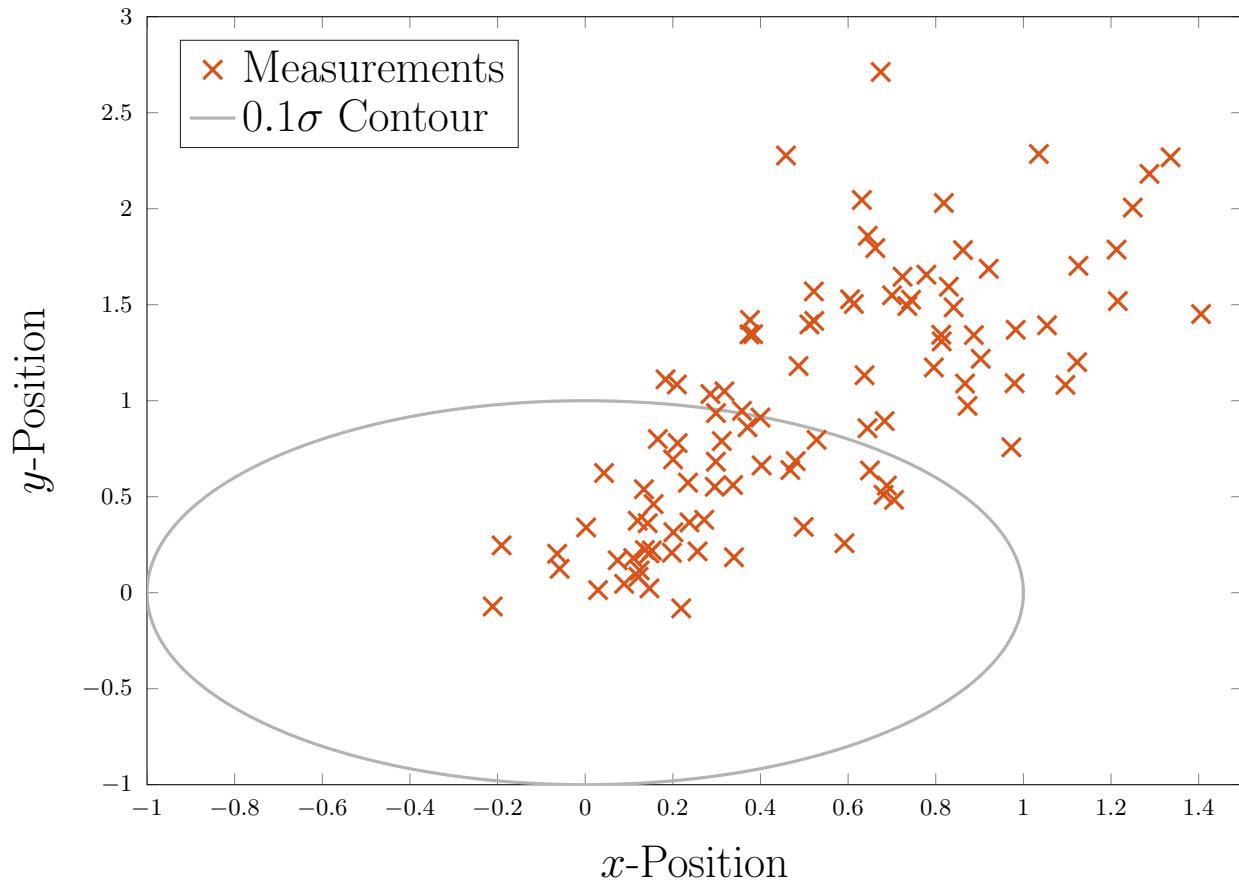
You may also notice that we've added a covariance contour onto the first two plots. This is to graphically represent our prior estimate. Because the covariance is quite large, which indicates low confidence in our initial estimate, the covariance contour associated with 0.1σ is shown.

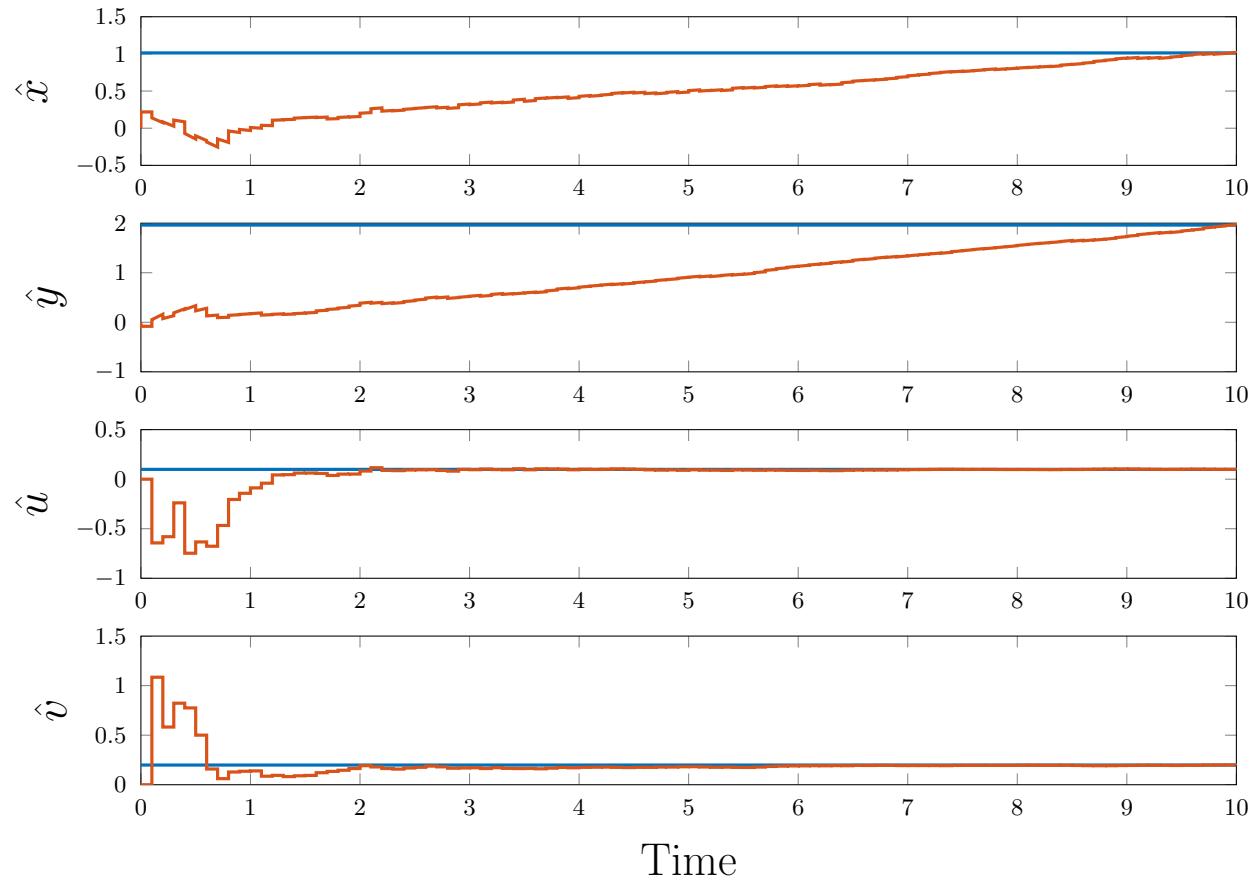
How can you plot covariance contours? Assume that $\xi = \cos \theta$ and $\eta = \sin \theta$ for $0 \leq \theta \leq 2\pi$. Given a mean, $\mathbf{m} \in \mathbb{R}^2$, and a covariance, $\mathbf{P} \in \mathbb{R}^{2 \times 2}$, compute the Cholesky factor, \mathbf{S} , such that $\mathbf{P} = \mathbf{S}\mathbf{S}^T$. Then, the points on an $s\sigma$ level contour are found for each θ as

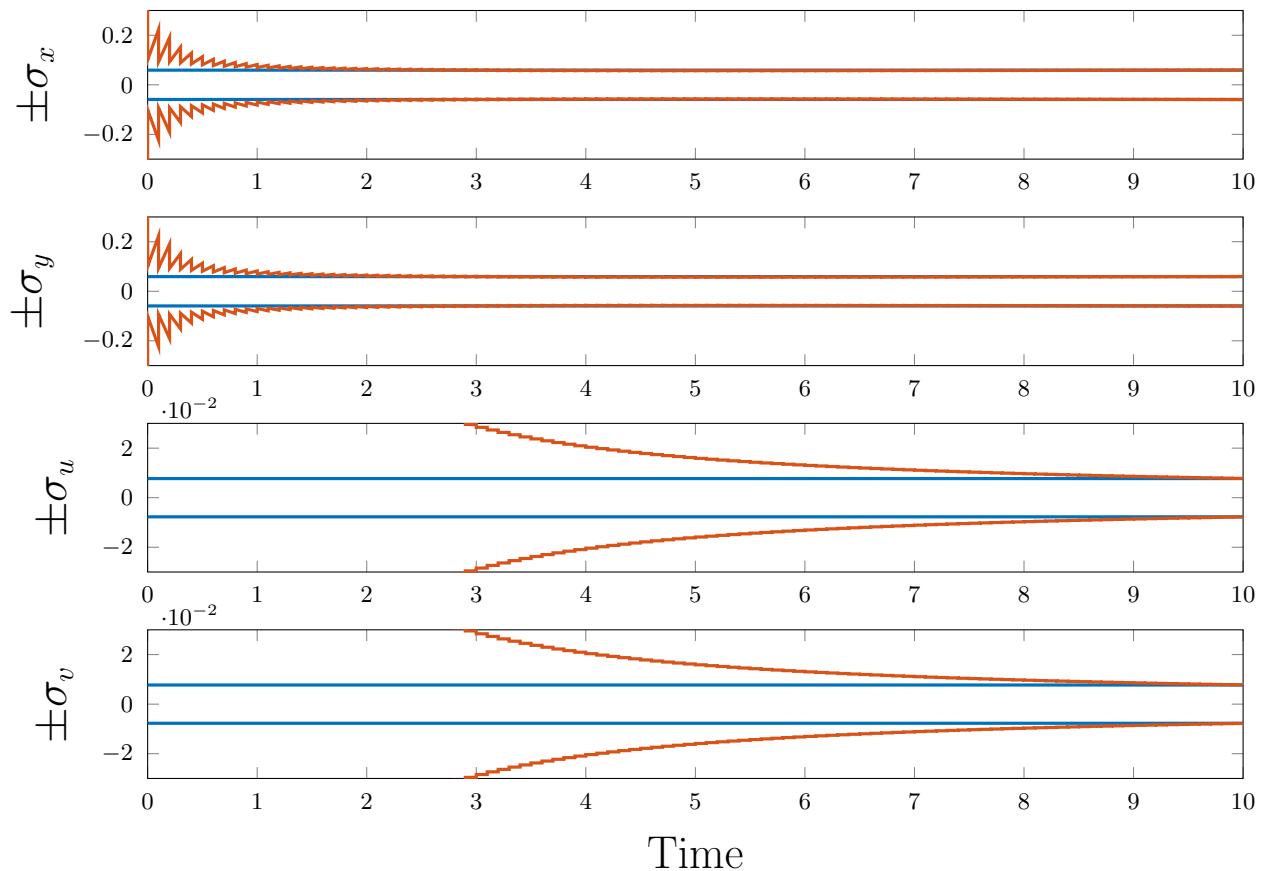
$$\begin{bmatrix} x \\ y \end{bmatrix} = \mathbf{m} + s\mathbf{S} \begin{bmatrix} \xi \\ \eta \end{bmatrix}$$

It is important to note that the mean and covariance only represent the first two central moments of the prior distribution even though we are illustrating them in the typical Gaussian manner.









Key point: Prior information is *required* for the sequential form of LUMVE, as we have currently posed it, but it is not required for the batch form of LUMVE.

If there is no prior information, what do we propagate? If we could propagate “zero information,” what would the update look like?

What problems, if any, do we encounter in trying to process data against the propagated mean and covariance?

There is a way to overcome this deficiency in the sequential form of LUMVE that is called the *information filter*, and we will revisit this topic later on.

2.7 The Batch Processor Revisited

What if the system is nonlinear? Everything that we’ve developed for least-squares estimation makes the assumption that we are dealing with a linear system. The answer is a bit anticlimactic, but perhaps expected: linearize and iterate.

Nonlinearities can be present in the dynamics or the measurements, which means that we need to deal with both, ultimately. We will now take the system to be described by

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t))$$

$$z_i = \boldsymbol{h}(\boldsymbol{x}_i) + \boldsymbol{v}_i$$

where $E\{\boldsymbol{v}_i\} = \mathbf{0}$ and $E\{\boldsymbol{v}_i \boldsymbol{v}_j^T\} = \boldsymbol{P}_{vv,i}\delta_{ij}$. We have assumed here that the noise is white. This will help establish a computational algorithm later on.

Since we have a linear method and a nonlinear system, we're going to linearize these equations. Therefore, consider a reference described by

$$\dot{\boldsymbol{x}}^*(t) = \boldsymbol{f}(\boldsymbol{x}^*(t))$$

$$z_i^* = \boldsymbol{h}(\boldsymbol{x}_i^*)$$

Note that we compute the reference measurement *without* noise.

To linearize about the reference, define deviations in the states and the measurements as

$$\delta \mathbf{x}(t) = \mathbf{x}(t) - \mathbf{x}^*(t)$$

$$\delta z_i = z_i - z_i^*$$

Differentiating the first equation with respect to time and using our nonlinear dynamics and measurements yields

$$\delta \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) - \mathbf{f}(\mathbf{x}^*(t))$$

$$\delta z_i = \mathbf{h}(\mathbf{x}_i) - \mathbf{h}(\mathbf{x}_i^*) + \mathbf{v}_i$$

This is where we apply linearization via a first-order Taylor series expansion of the first terms about the reference state

$$\delta \dot{\mathbf{x}}(t) \approx \mathbf{f}(\mathbf{x}^*(t)) + \mathbf{F}(\mathbf{x}^*(t))\delta \mathbf{x}(t) - \mathbf{f}(\mathbf{x}^*(t))$$

$$\delta z_i \approx \mathbf{h}(\mathbf{x}_i^*) + \tilde{\mathbf{H}}(\mathbf{x}_i^*)\delta \mathbf{x}_i - \mathbf{h}(\mathbf{x}_i^*) + \mathbf{v}_i$$

We can eliminate the similar terms to find

$$\delta \dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}^*(t))\delta \mathbf{x}(t)$$

$$\delta \mathbf{z}_i = \tilde{\mathbf{H}}(\mathbf{x}_i^*)\delta \mathbf{x}_i + \mathbf{v}_i$$

To make use of our previously developed least-squares approaches, we also need to convert the continuous time evolution of the deviation into a discrete time evolution. We do this via the state transition matrix, such that

$$\delta \mathbf{x}_i = \Phi(t_i, t_k)\delta \mathbf{x}_k$$

$$\delta \mathbf{z}_i = \tilde{\mathbf{H}}(\mathbf{x}_i^*)\delta \mathbf{x}_i + \mathbf{v}_i$$

where, from the properties of the state transition matrix, we know that $\Phi(t_i, t_k)$ is found by integrating the matrix differential equation

$$\dot{\Phi}(t, t_k) = \mathbf{F}(\mathbf{x}^*(t))\Phi(t, t_k)$$

from $t = t_k$ to $t = t_i$ with the initial condition $\Phi(t_k, t_k) = \mathbf{I}_n$.

The “deviation” equations are now in the same form that we started with to develop the LUMVE solution, except that we’ve assumed that the measurement noise is white. Strictly speaking, we aren’t required to make this assumption, but it’s a pretty standard assumption, and we will see that it leads to a more computationally efficient solution.

The important element to note is that we are dealing with *deviations*. When we apply the LUMVE solution to our deviations, we need to use measurements of the form

$$\begin{aligned}\delta \mathbf{z}_i &= \mathbf{z}_i - \mathbf{z}_i^* \\ &= \mathbf{z}_i - \mathbf{h}(\mathbf{x}_i^*)\end{aligned}$$

where \mathbf{z}_i are the *actual* measurements. Likewise, the prior estimate is in the form of a deviation from the reference, i.e.,

$$\delta \bar{\mathbf{x}}_k = \bar{\mathbf{x}}_k - \mathbf{x}_k^*$$

meaning that we will provide $\delta \bar{\mathbf{x}}_k$ as our prior estimate and not $\bar{\mathbf{x}}_k$.

Then, the LUMVE solution is given by

$$\delta\hat{\mathbf{x}}_k = [\bar{\mathbf{P}}_{xx,k}^{-1} + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1} [\bar{\mathbf{P}}_{xx,k}^{-1} \delta\bar{\mathbf{x}}_k + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \delta\mathbf{z}]$$

$$\mathbf{P}_{xx,k} = [\bar{\mathbf{P}}_{xx,k}^{-1} + \mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H}]^{-1}$$

The estimated state produced by LUMVE is an estimated deviation, $\delta\hat{\mathbf{x}}_k$, meaning that the estimate of the “full state” is given by

$$\hat{\mathbf{x}}_k = \mathbf{x}_k^* + \delta\hat{\mathbf{x}}_k$$

The \mathbf{P}_{vv} matrix and the \mathbf{H} matrix are still concatenations of the individual terms. For \mathbf{P}_{vv} , the formulation remains the same, assuming white noise in this case. For \mathbf{H} , however, we now need to use the reference state to compute the measurement Jacobian.

The concatenation of terms, therefore, looks like

$$\delta\mathbf{z} = \begin{bmatrix} \mathbf{z}_1 - \mathbf{h}(\mathbf{x}_1^*) \\ \mathbf{z}_2 - \mathbf{h}(\mathbf{x}_2^*) \\ \vdots \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} \tilde{\mathbf{H}}(\mathbf{x}_1^*)\Phi(t_1, t_k) \\ \tilde{\mathbf{H}}(\mathbf{x}_2^*)\Phi(t_2, t_k) \\ \vdots \end{bmatrix} \quad \mathbf{P}_{vv} = \begin{bmatrix} \mathbf{P}_{vv,1} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{P}_{vv,2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

where it is also worth noting that the state transition matrix is, in general, numerically integrated along the reference trajectory; that is,

$$\dot{\mathbf{x}}^*(t) = \mathbf{f}(\mathbf{x}^*(t))$$

$$\dot{\Phi}(t, t_k) = \mathbf{F}(\mathbf{x}^*(t))\Phi(t, t_k)$$

are simultaneously integrated (subject to appropriate initial conditions) to generate both the reference states and the state transition matrix required at each measurement time in order to assemble δz and H .

We can leverage the white-noise property to develop a more computationally friendly version of LUMVE.

To do so, let's define a few new terms as

$$\lambda = \bar{P}_{xx,k}^{-1}\delta\bar{x}_k + H^T P_{vv}^{-1}\delta z$$

$$\Lambda = \bar{P}_{xx,k}^{-1} + H^T P_{vv}^{-1}H$$

which are commonly called the information state and the information matrix.

The LUMVE estimate is then given by the solution to the normal equations

$$\boldsymbol{\Lambda} \delta \hat{\mathbf{x}}_k = \boldsymbol{\lambda}$$

and the covariance is given by

$$\mathbf{P}_{xx,k} = \boldsymbol{\Lambda}^{-1}$$

We will now use the assumption that the noise is time-wise uncorrelated, such that \mathbf{P}_{vv} is block diagonal. This means that \mathbf{P}_{vv}^{-1} is block diagonal with entries of the form $\mathbf{P}_{vv,\ell}^{-1}$.

When we have a time-wise uncorrelated noise, it is straightforward to see that

$$\mathbf{H}^T \mathbf{P}_{vv}^{-1} \delta \mathbf{z} = \sum_{\ell=1}^m [\tilde{\mathbf{H}}(\mathbf{x}_\ell^*) \boldsymbol{\Phi}(t_\ell, t_k)]^T \mathbf{P}_{vv,\ell}^{-1} \delta \mathbf{z}_\ell$$

$$\mathbf{H}^T \mathbf{P}_{vv}^{-1} \mathbf{H} = \sum_{\ell=1}^m [\tilde{\mathbf{H}}(\mathbf{x}_\ell^*) \boldsymbol{\Phi}(t_\ell, t_k)]^T \mathbf{P}_{vv,\ell}^{-1} [\tilde{\mathbf{H}}(\mathbf{x}_\ell^*) \boldsymbol{\Phi}(t_\ell, t_k)]$$

This allows us to compute $\boldsymbol{\lambda}$ and $\boldsymbol{\Lambda}$ through accumulation by summation as

$$\begin{aligned}\boldsymbol{\lambda} &= \bar{\mathbf{P}}_{xx,k}^{-1} \delta \hat{\mathbf{x}}_k + \sum_{\ell=1}^m [\tilde{\mathbf{H}}(\mathbf{x}_\ell^*) \boldsymbol{\Phi}(t_\ell, t_k)]^T \mathbf{P}_{vv,\ell}^{-1} \delta \mathbf{z}_\ell \\ \boldsymbol{\Lambda} &= \bar{\mathbf{P}}_{xx,k}^{-1} + \sum_{\ell=1}^m [\tilde{\mathbf{H}}(\mathbf{x}_\ell^*) \boldsymbol{\Phi}(t_\ell, t_k)]^T \mathbf{P}_{vv,\ell}^{-1} [\tilde{\mathbf{H}}(\mathbf{x}_\ell^*) \boldsymbol{\Phi}(t_\ell, t_k)]\end{aligned}$$

Then, our LUMVE solution is

$$\delta \hat{\mathbf{x}}_k = \boldsymbol{\Lambda}^{-1} \boldsymbol{\lambda}$$

$$\mathbf{P}_{xx,k} = \boldsymbol{\Lambda}^{-1}$$

Since we have a nonlinear system, we want to iterate. We do this by resetting some of our starting values: the reference state, the prior estimate (as an estimated deviation), and the prior covariance. Note that these are the only values we have starting off.

Let $(\mathbf{x}_k^*)_{n-1}$ represent the reference state of iteration $n - 1$. The reference state is reset

using the estimated value of the deviation:

$$(\boldsymbol{x}_k^*)_n = (\boldsymbol{x}_k^*)_{n-1} + \delta \hat{\boldsymbol{x}}_k$$

This is the same as saying that our reference state of iteration n is equal to the full estimated state after iteration $n - 1$ is completed.

We must also reset the prior information, and we must be careful not to change the prior information. Therefore, if the prior estimate and covariance at iteration $n - 1$ are given by

$$(\delta \bar{\boldsymbol{x}}_k)_{n-1} \quad \text{and} \quad (\bar{\boldsymbol{P}}_{xx,k})_{n-1}$$

then the prior estimate and covariance at iteration n are

$$(\delta \bar{\boldsymbol{x}}_k)_n = (\delta \bar{\boldsymbol{x}}_k)_{n-1} - \delta \hat{\boldsymbol{x}}_k$$

$$(\bar{\boldsymbol{P}}_{xx,k})_n = (\bar{\boldsymbol{P}}_{xx,k})_{n-1}$$

Why? To preserve the prior information, we need to ensure that the full estimated state between iterations remains constant. That is

$$(\boldsymbol{x}_k^*)_n + (\delta \bar{\boldsymbol{x}}_k)_n = (\boldsymbol{x}_k^*)_{n-1} + (\delta \bar{\boldsymbol{x}}_k)_{n-1}$$

The left-hand side is the full estimated state at iteration n , and the right-hand side is the full estimated state at iteration $n - 1$.

We know that the reference state for iteration n is updated from iteration $n - 1$ by

$$(\boldsymbol{x}_k^*)_n = (\boldsymbol{x}_k^*)_{n-1} + \delta \hat{\boldsymbol{x}}_k$$

Therefore, the condition for preserving the full estimated state becomes

$$(\boldsymbol{x}_k^*)_{n-1} + \delta \hat{\boldsymbol{x}}_k + (\delta \bar{\boldsymbol{x}}_k)_n = (\boldsymbol{x}_k^*)_{n-1} + (\delta \bar{\boldsymbol{x}}_k)_{n-1}$$

We can now cancel like terms and solve for the estimate (deviation) at iteration n that preserves the full state estimate from iteration $n - 1$ to iteration n . This gives us the equation that we started with for the prior estimate update, i.e.,

$$(\delta \bar{\boldsymbol{x}}_k)_n = (\delta \bar{\boldsymbol{x}}_k)_{n-1} - \delta \hat{\boldsymbol{x}}_k$$

We also need to preserve the uncertainty information. This is readily accomplished by starting iteration n with the same covariance used to start iteration $n - 1$, i.e.,

$$(\bar{\boldsymbol{P}}_{xx,k})_n = (\bar{\boldsymbol{P}}_{xx,k})_{n-1}$$

Let's enumerate the steps for the iterative batch processor algorithm (remember that the index k in the LUMVE solution is arbitrary, so we will use $k = 0$ for convenience in this listing):

1. Input a set of data described by values of \mathbf{z}_i at times t_i with associated measurement noise covariances $\mathbf{P}_{vv,i}$, where $t_i \geq t_0 \forall i \in \{1, 2, \dots, m\}$.
2. Begin with a reference state \mathbf{x}_0^* , a prior estimate $\delta\bar{\mathbf{x}}_0$, and a prior covariance $\bar{\mathbf{P}}_{xx,0}$, all at time t_0 .
3. Set $n = 1$, and initialize an iteration loop
 - (a) Set $\ell = 1$, and initialize an accumulation loop with

$$t_{\ell-1} = t_0 \quad \mathbf{x}^*(t_{\ell-1}) = \mathbf{x}_0^* \quad \boldsymbol{\lambda} = \bar{\mathbf{P}}_{xx,0}^{-1} \delta\bar{\mathbf{x}}_0$$

$$\Phi(t_{\ell-1}, t_0) = \mathbf{I}_n \quad \boldsymbol{\Lambda} = \bar{\mathbf{P}}_{xx,0}^{-1}$$

Note that $\boldsymbol{\lambda} = \mathbf{0}_{n \times 1}$ and $\boldsymbol{\Lambda} = \mathbf{0}_{n \times n}$ if there is no prior estimate.

- i. Parse the ℓ^{th} measurement to get t_ℓ , \mathbf{z}_ℓ , and $\mathbf{P}_{vv,\ell}$.

ii. Integrate the reference state and state transition matrix from $t_{\ell-1}$ to t_ℓ

$$\dot{\mathbf{x}}^*(t) = \mathbf{f}(\mathbf{x}^*(t)) \quad \text{s.t.i.c. } \mathbf{x}^*(t_{\ell-1})$$

$$\dot{\Phi}(t, t_0) = \mathbf{F}(\mathbf{x}^*(t))\Phi(t, t_0) \quad \text{s.t.i.c. } \Phi(t_{\ell-1}, t_0)$$

This gives $\mathbf{x}_\ell^* = \mathbf{x}^*(t_\ell)$ and $\Phi(t_\ell, t_0)$.

iii. Accumulate the current observation

$$\boldsymbol{\lambda} = \boldsymbol{\lambda} + [\tilde{\mathbf{H}}(\mathbf{x}_\ell^*)\Phi(t_\ell, t_0)]^T \mathbf{P}_{vv,\ell}^{-1} [\mathbf{z}_\ell - \mathbf{h}(\mathbf{x}_\ell^*)]$$

$$\boldsymbol{\Lambda} = \boldsymbol{\Lambda} + [\tilde{\mathbf{H}}(\mathbf{x}_\ell^*)\Phi(t_\ell, t_0)]^T \mathbf{P}_{vv,\ell}^{-1} [\tilde{\mathbf{H}}(\mathbf{x}_\ell^*)\Phi(t_\ell, t_0)]$$

iv. If $\ell < m$, set $\ell = \ell + 1$ and return to Step 3(a)i with $\mathbf{x}^*(t_{\ell-1}) = \mathbf{x}^*(t_\ell)$ and $\Phi(t_{\ell-1}, t_0) = \Phi(t_\ell, t_0)$. Otherwise, exit the accumulation loop.

(b) Solve the normal equations to find the estimate and compute the covariance:

$$\boldsymbol{\Lambda} \delta \hat{\mathbf{x}}_0 = \boldsymbol{\lambda} \quad \mathbf{P}_{xx,0} = \boldsymbol{\Lambda}^{-1}$$

- (c) If the process has converged, exit the iteration loop. Otherwise, set $n = n + 1$ and return to Step 3a with

$$\boldsymbol{x}_0^* = \boldsymbol{x}_0^* + \delta \hat{\boldsymbol{x}}_0$$

$$\delta \bar{\boldsymbol{x}}_0 = \delta \bar{\boldsymbol{x}}_0 - \delta \hat{\boldsymbol{x}}_0$$

$$\bar{\boldsymbol{P}}_{xx,0} = \bar{\boldsymbol{P}}_{xx,0}$$

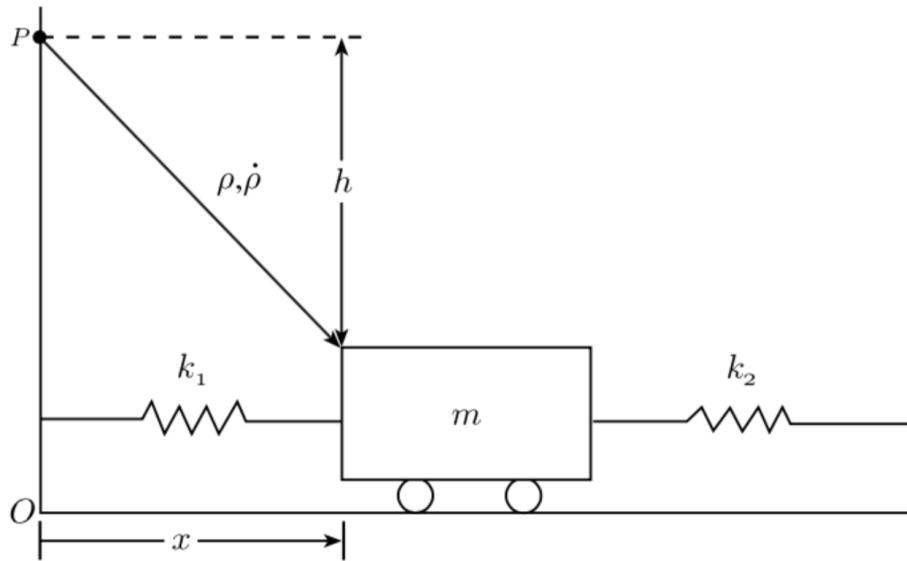
4. Output the converged reference trajectory \boldsymbol{x}_0^* and the covariance $\boldsymbol{P}_{xx,0}$.

2.7.1 A Spring-Mass Problem

Let's take a look at a problem from Tapley, Schutz, and Born and apply the iterative batch processor to see if we can replicate the results that they give.

A block of mass m is attached to two parallel vertical walls by two springs. The spring constants are k_1 and k_2 .

An observer is placed at a height h on the left wall (at position P). This observer measures the range ρ and the range-rate $\dot{\rho}$ of the mass (taken to be a point mass).



If the horizontal distance of the mass from the left wall is denoted by x , the objective is to use the range and range-rate information to estimate the position x and the velocity \dot{x} .

Let's put together all of the pieces that we need to use the iterative batch processor.

First, we'll consider all of the dynamics-related quantities.

The equation of motion governing the position of the mass is

$$\ddot{x} = -\frac{k_1 + k_2}{m}x$$

If we define states to be x and $v = \dot{x}$, and define $\omega^2 = (k_1 + k_2)/m$, then the first-order form of the dynamics is

$$\begin{bmatrix} \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \\ -\omega^2 x \end{bmatrix}$$

This is the nonlinear system form of the dynamics, i.e. $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$, but these dynamics are linear, so we can also write them as

$$\dot{\mathbf{x}}(t) = \mathbf{F}(t)\mathbf{x}(t) \quad \text{where} \quad \mathbf{F}(t) = \mathbf{F} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix}$$

If you take the partials of the “nonlinear” system, you will find that the Jacobian matrix is state-independent and that it is identical to the linear system dynamics matrix.

We also need the state transition matrix to complete our description of the dynamical system.

We can compute this by numerically integrating the dynamics of the state transition matrix or, since this is a linear time-invariant system, we can also find the state transition matrix using the matrix exponential.

The matrix exponential option is certainly viable here, but it is not quite as clean as the cases we had before since $\mathbf{F}^2 \neq \mathbf{0}$.

However, we have a system that is a harmonic oscillator, so we would expect the solution to the second-order system to be of the form

$$x(t) = A \cos \omega t + B \sin \omega t$$

This also gives us a velocity solution by differentiation as

$$v(t) = B\omega \cos \omega t - A\omega \sin \omega t$$

If the mass is at position x_0 and velocity v_0 at time $t_0 = 0$, then the coefficients are

$$A = x_0 \quad \text{and} \quad B = v_0/\omega$$

and the solution is

$$\begin{bmatrix} x(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} \cos \omega(t - t_0) & \frac{1}{\omega} \sin \omega(t - t_0) \\ -\omega \sin \omega(t - t_0) & \cos \omega(t - t_0) \end{bmatrix} \begin{bmatrix} x_0 \\ v_0 \end{bmatrix}$$

Therefore, the state transition matrix can be determined exactly as

$$\Phi(t, t_0) = \begin{bmatrix} \cos \omega(t - t_0) & \frac{1}{\omega} \sin \omega(t - t_0) \\ -\omega \sin \omega(t - t_0) & \cos \omega(t - t_0) \end{bmatrix}$$

This is everything related to the dynamics that we need to determine a batch LUMVE solution.

So, we now turn towards the measurement-related information.

In particular, we need a nonlinear function representing the measurements, and we need a measurement Jacobian for the linearization of the nonlinear function.

For this problem, we are considering range and range-rate measurements from an observer that is located at the point P (refer back to the diagram). These are given as functions of the position and velocity of the mass, as well as the height of the observer, as

$$\rho = \sqrt{x^2 + h^2}$$

$$\dot{\rho} = \frac{xv}{\sqrt{x^2 + h^2}}$$

The range-rate equation can be confirmed by differentiating the range equation with respect to time.

Putting these two equations together, we have a nonlinear function of the state describing the measurements as

$$\mathbf{h}(x) = \begin{bmatrix} \sqrt{x^2 + h^2} \\ \frac{xv}{\sqrt{x^2 + h^2}} \end{bmatrix}$$

Remember that the data is modeled according to

$$z_i = \mathbf{h}(\mathbf{x}_i) + \mathbf{v}_i$$

where $\text{E}\{\mathbf{v}_i\} = \mathbf{0}$ and $\text{E}\{\mathbf{v}_i \mathbf{v}_i^T\} = \mathbf{P}_{vv,i}$.

The only thing left in order to be able to determine the batch LUMVE solution is the measurement Jacobian, which is defined as

$$\tilde{\mathbf{H}}(\mathbf{x}) = \left[\frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}} \right]$$

Taking the derivatives, it follows that

$$\tilde{\mathbf{H}}(\mathbf{x}) = \begin{bmatrix} x & 0 \\ \frac{v}{\sqrt{x^2 + h^2}} - \frac{x^2 v}{(\sqrt{x^2 + h^2})^3} & \frac{x}{\sqrt{x^2 + h^2}} \end{bmatrix}$$

We now have all of the pieces to put together a batch LUMVE solution; moreover, since we have a nonlinear measurement function, we are going to apply the batch LUMVE

solution iteratively.

To apply the batch LUMVE, we need numbers and data. We will take the mass, spring constants, observer altitude, and true position and velocity of the mass to be

$$m = 1.5 \text{ kg} \quad k_1 = 2.5 \text{ N/m} \quad k_2 = 3.7 \text{ N/m}$$

$$h = 5.4 \text{ m} \quad x_0 = 3.0 \text{ m} \quad v_0 = 0.0 \text{ m/s}$$

We will also use an initial reference, a prior estimate, and a prior covariance of

$$\boldsymbol{x}_0^* = \begin{bmatrix} 4.0 \\ 0.2 \end{bmatrix} \quad \delta \bar{\boldsymbol{x}}_0 = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix} \quad \bar{\boldsymbol{P}}_{xx,0} = \begin{bmatrix} 1000 & 0 \\ 0 & 100 \end{bmatrix}$$

with position units of m for position and m/s for velocity.

The range and range-rate data, along with the times at which they are acquired, are shown in the following table. The measurement noise covariance matrix associated with the data is taken to be $\boldsymbol{P}_{vv,i} = \boldsymbol{I}_2$.

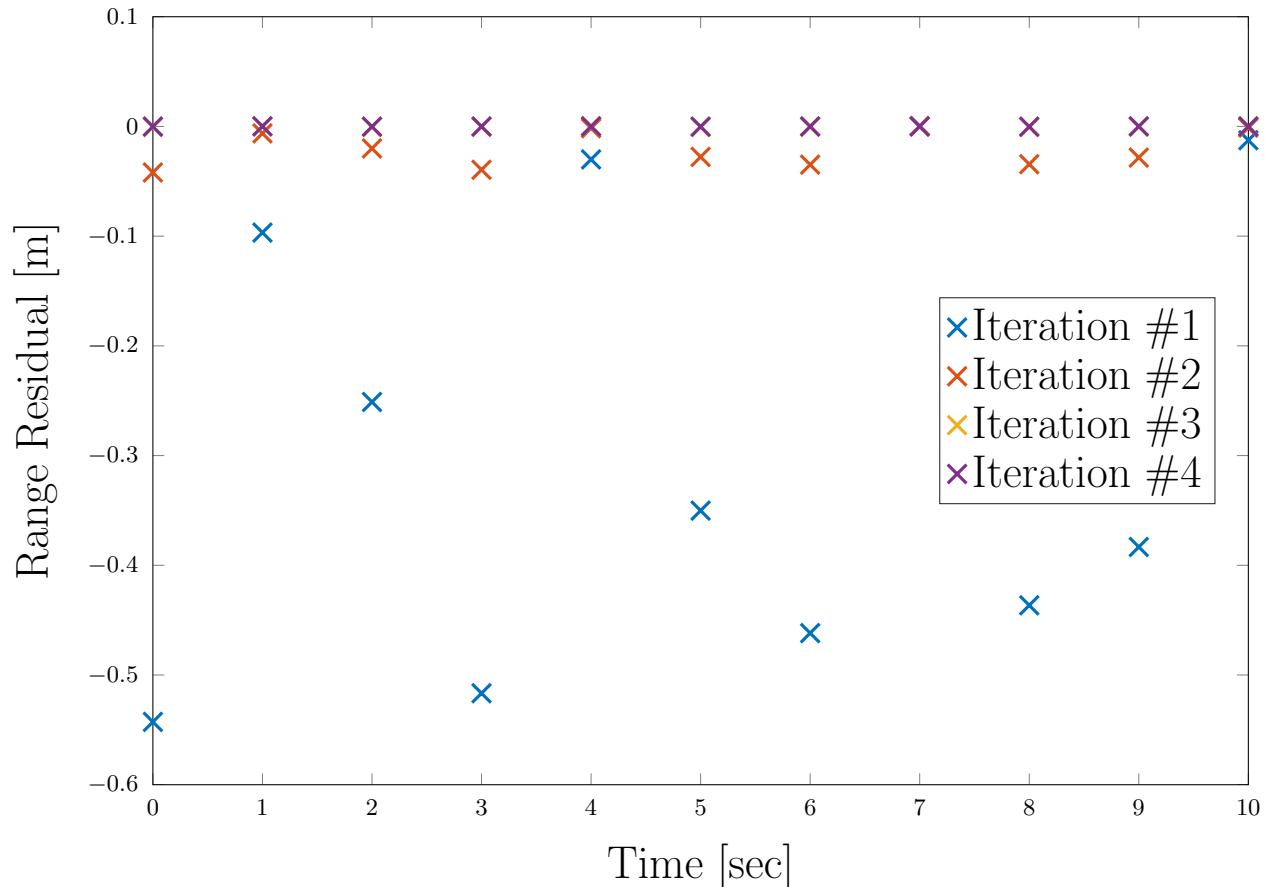
We will apply four iterations to this data. Usually, you would continue until some stopping condition, but you can also apply a fixed number of iterations.

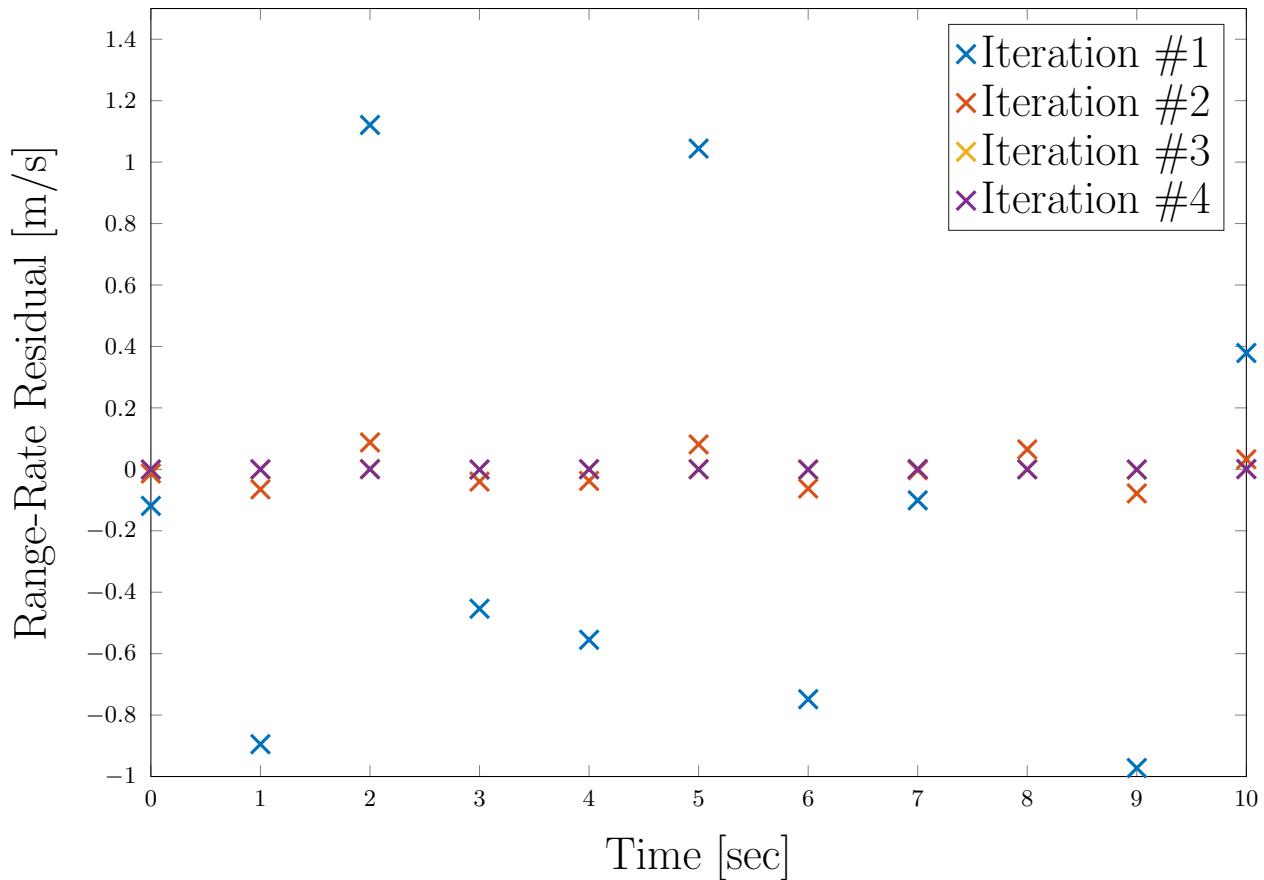
Time [s]	Range [m]	Range-Rate [m/s]
0.00	6.17737808459220	0.0000000000000000
1.00	5.56327661282686	1.312858634955140
2.00	5.69420161397342	-1.544881143816120
3.00	6.15294262127432	0.534923988815733
4.00	5.46251322092491	0.884698415328368
5.00	5.83638064328625	-1.561232489180540
6.00	6.08236452736002	1.009799431575470
7.00	5.40737619817037	0.317051170392150
8.00	5.97065615746125	-1.374530709756060
9.00	5.97369258835895	1.367681694432360
10.00	5.40669060248179	-0.302111588503166

For each iteration, we will plot the range and range-rate residuals, which are given by

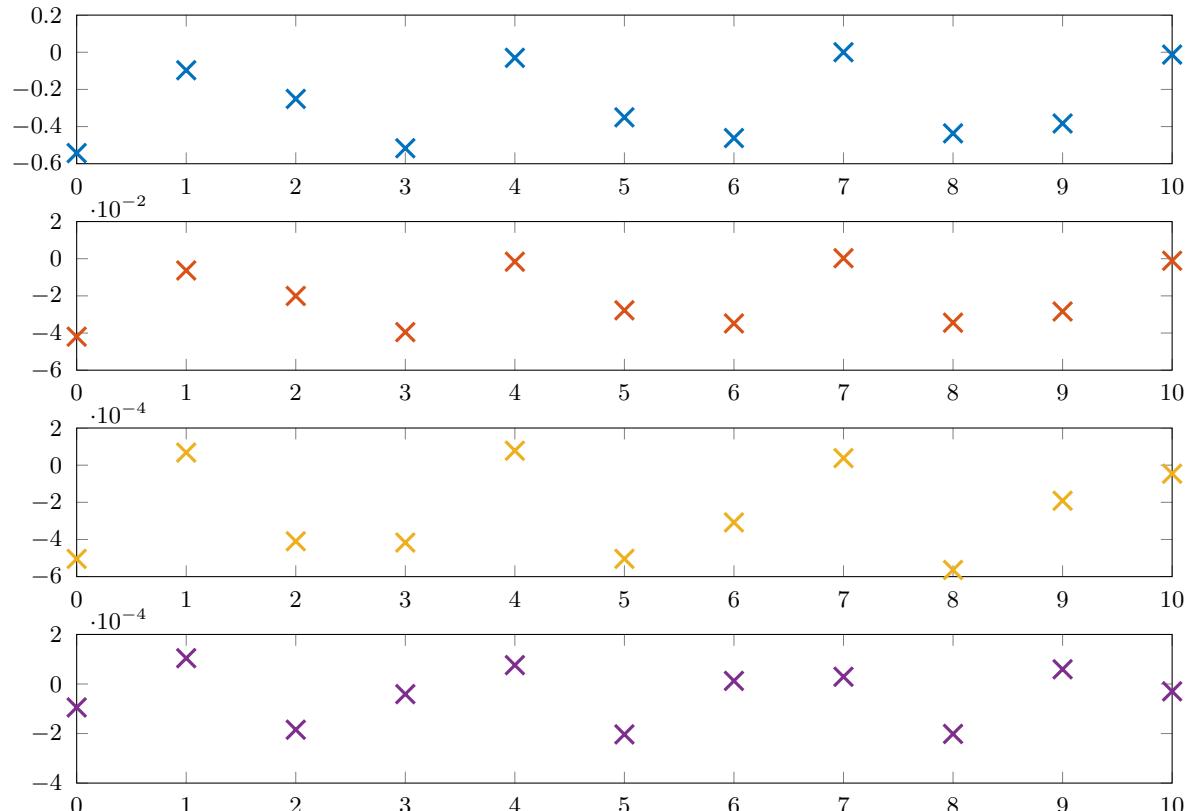
$$\delta \mathbf{z}_i = \mathbf{z}_i - \mathbf{h}(\mathbf{x}_i^*)$$

For the fourth iteration, we will then analyze statistics of the residuals and provide the output solution from the iterative batch processor.

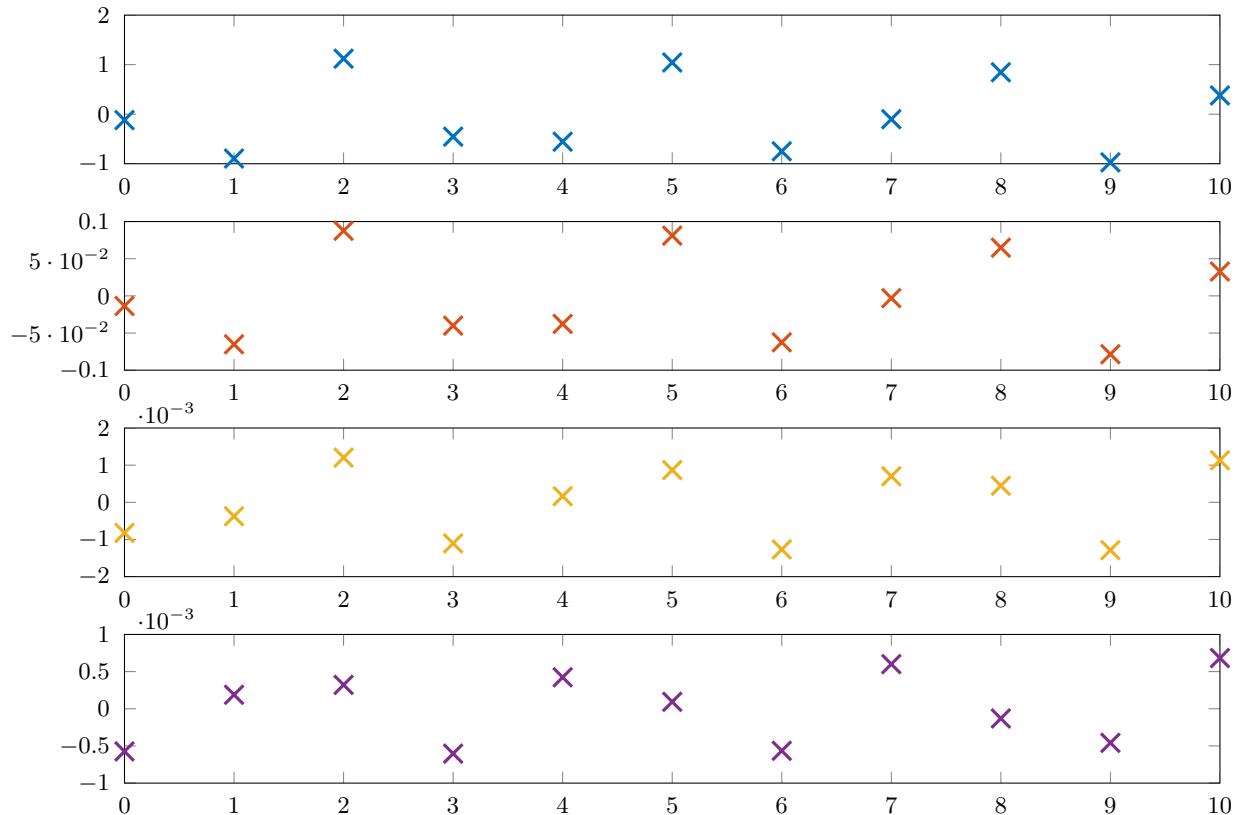




Range Residuals at Each Iteration [m]



Range-Rate Residuals at Each Iteration [m/s]



The mean, root-mean-square, and standard deviation of the range residuals at the fourth iteration are, respectively,

$$E\{\delta\rho\} = -4.3000 \times 10^{-5} \text{ m}$$

$$\sqrt{E\{\delta\rho^2\}} = 1.1628 \times 10^{-4} \text{ m}$$

$$\sqrt{E\{(\delta\rho - E\{\delta\rho\})^2\}} = 1.0804 \times 10^{-4} \text{ m}$$

The mean, root-mean-square, and standard deviation of the range-rate residuals at the fourth iteration are, respectively,

$$E\{\delta\dot{\rho}\} = -1.7577 \times 10^{-6} \text{ m/s}$$

$$\sqrt{E\{\delta\dot{\rho}^2\}} = 4.6661 \times 10^{-4} \text{ m/s}$$

$$\sqrt{E\{(\delta\dot{\rho} - E\{\delta\dot{\rho}\})^2\}} = 4.6661 \times 10^{-4} \text{ m/s}$$

The reference state (now our estimated full state) and the covariance after the fourth iteration are

$$\boldsymbol{x}_0^* = \begin{bmatrix} 3.0002 \text{ m} \\ 1.1818 \times 10^{-3} \text{ m/s} \end{bmatrix}$$

$$\boldsymbol{P}_0 = \begin{bmatrix} 1.6935 \times 10^{-1} \text{ m}^2 & 1.2775 \times 10^{-2} \text{ m}^2/\text{s} \\ 1.2775 \times 10^{-2} \text{ m}^2/\text{s} & 5.8448 \times 10^{-1} \text{ m}^2/\text{s}^2 \end{bmatrix}$$

We can also represent the information in the covariance matrix by the standard deviation of the position, the standard deviation of the velocity, and the correlation, which, after the fourth iteration, are

$$\sigma_{x_0} = 4.1152 \times 10^{-1} \text{ m}$$

$$\sigma_{v_0} = 7.6451 \times 10^{-1} \text{ m/s}$$

$$\rho_{x_0 v_0} = 4.0607 \times 10^{-2}$$

2.8 Some Remarks on Least Squares Methods

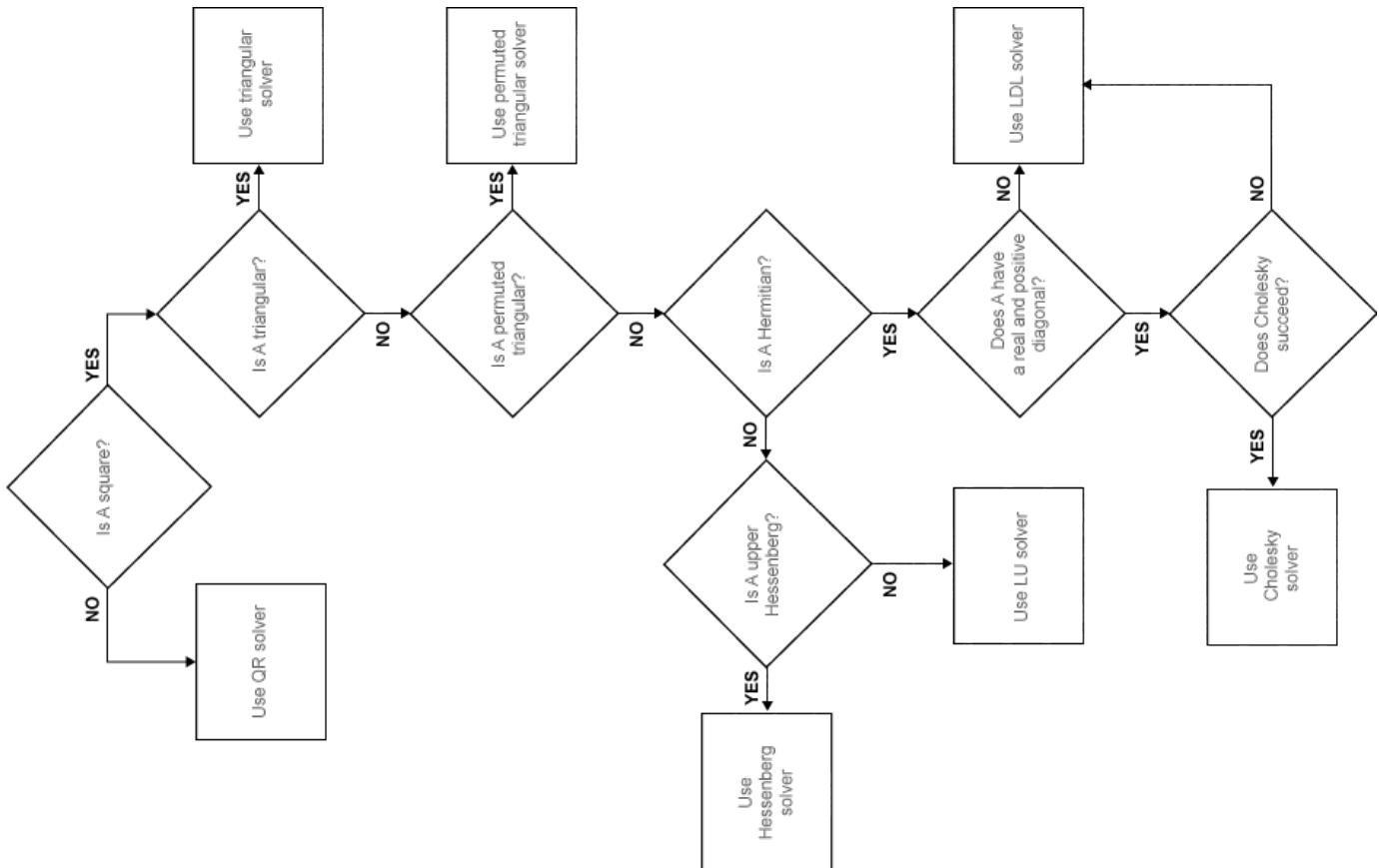
What we have covered in the least squares methods forms the basis for least squares estimation, but it is not the end of the line.

Fundamentally, the least squares solutions come down to computing the solution to the normal equation

$$\Lambda \hat{\boldsymbol{x}} = \boldsymbol{\lambda}$$

This, of course, can be “solved” by inverting the matrix on the left and multiplying through by the inverse, and this is basically what we have assumed will always work.

In MATLAB, we rely on the “backslash” operator to solve this system of equations for the estimate, which is actually a shortcut for performing a wide array of numerical methods for solving the system. The following flowchart illustrates the process for the “backslash” operator, from which we see that this is a complex operation that is taking place.



The continuation of least squares methods follows a similar trend. Further developments, which we will not focus on here, aim to make the process robust, especially in the solution of the normal equation.

Cholesky methods and orthogonal transformation methods (such as Givens transformations and the Householder transformation) are among the most popular methods for solving ill-conditioned problems.

For additional reading, consult the book by Tapley, Schutz, and Born.

Minimum Mean Square Error Estimation

So far, we have developed a method to process a batch of data in either a direct application of LUMVE for linear systems or through the use of an iterative application of LUMVE for a nonlinear system.

In the case where we have initial information available, we can use that information to initialize LUMVE, either in batch form or sequential form.

We will now turn our attention to a different way to develop sequential estimation ap-

proaches. The resulting method will be sequential in nature, but it will be based on the celebrated Kalman filter framework for minimum mean square error (MMSE) estimation.

The following is a paraphrasing of the preface to Bill Lear's unpublished book on Kalman filtering: Kalman Filtering Techniques.

A Kalman filter is a computer algorithm that is used to process error corrupted measurement data. The purpose of the processing is to better determine the parameters or variables associated with the process that generates the measurements. For example, a radar station tracking the Space Shuttle makes range, azimuth and elevation angle measurements. Using a Kalman filter to process these measurements, one can determine the position, velocity and acceleration of the Shuttle, and also determine what bias errors are adding to the measurements.

Kalman filters are useful in modern navigation problems, particularly those problems requiring instantaneous real-time solutions. [For instance, a Kalman filter navigation algorithm] processed the Earth-based Doppler data of the Lunar Module (LM) as it descended and ascended from the surface of the moon. Based on this program, a real-time navigation position correction was voice-

linked to the astronauts as they descended to the surface of the moon. This enabled pinpoint landing accuracy.

Kalman filters are useful in many areas other than navigation. They can be used to determine irregularities in the Earth's gravity field. They can be used to determine the density of the atmosphere from altitude measurements of a falling sphere. They can be used to analyze the stock market (don't get your hopes up, they don't predict well). They can process calibration measurements to better determine the state of a chemical process.

There are many variations of Kalman filters to which various people attach their names. But that is all they are, variations. The man who started it all is Rudy Kalman.

3.1 The Kalman Filter

We will now take a system to be described by dynamics of the form

$$\dot{\mathbf{x}}(t) = \mathbf{F}_x(t)\mathbf{x}(t) + \mathbf{F}_w(t)\mathbf{w}(t)$$

where

$$\mathrm{E}\{\boldsymbol{w}(t)\} = \mathbf{0} \quad \text{and} \quad \mathrm{E}\{\boldsymbol{w}(t)\boldsymbol{w}^T(\tau)\} = \boldsymbol{Q}_{ww}(t)\delta(t - \tau)$$

The state of the system is denoted by $\boldsymbol{x}(t)$, which is taken to have a mean at t_0 of $\boldsymbol{m}_x(t_0) = \boldsymbol{m}_{x,0}$ and a covariance covariance at t_0 of $\boldsymbol{P}_{xx}(t_0) = \boldsymbol{P}_{xx,0}$.

The term $\boldsymbol{w}(t)$ is a white-noise process providing stochastic excitation to the deterministic dynamics. This is *roughly* the corollary to the measurement noise that we have previously dealt with, but it is now a continuous-time process.

White noise is physically unrealizable as it is characterized by constant power across all frequencies, but it is a useful mathematical model of stochastic perturbations. The power spectral density, $\boldsymbol{Q}_{ww}(t)$, is actually constant for white-noise processes, but there is nothing that will stop us from assuming it is time-varying in the following developments.

For this system, $\boldsymbol{F}_x(t)$ and $\boldsymbol{F}_w(t)$ are deterministic mapping matrices that, respectively, map the states and the process noise into the dynamics of the states.

For the Kalman filter, we will concern ourselves with how the mean and covariance

of the state evolve through time and how they change upon receiving new measurement data. These will be the propagation and update stages of the Kalman filter.

The mean of the state as a function of time is defined by

$$\mathbf{m}_x(t) = \text{E} \{ \mathbf{x}(t) \}$$

Taking the time rate of change and interchanging the order of differentiation and expectation yields

$$\dot{\mathbf{m}}_x(t) = \text{E} \{ \dot{\mathbf{x}}(t) \}$$

Applying the system dynamics within the expectation, it follows that

$$\begin{aligned}\dot{\mathbf{m}}_x(t) &= \text{E} \{ \mathbf{F}_x(t) \mathbf{x}(t) + \mathbf{F}_w(t) \mathbf{w}(t) \} \\ &= \text{E} \{ \mathbf{F}_x(t) \mathbf{x}(t) \} + \text{E} \{ \mathbf{F}_w(t) \mathbf{w}(t) \}\end{aligned}$$

From the fact that $\mathbf{F}_x(t)$ and $\mathbf{F}_w(t)$ are deterministic and recalling that the process noise is taken to be zero-mean, the mean evolves according to

$$\dot{\mathbf{m}}_x(t) = \mathbf{F}_x(t) \mathbf{m}_x(t)$$

This is our forward evolution equation for the mean; we now turn to developing a similar equation for the covariance. To proceed, we first define the error in the state to be the difference of the truth from the mean, i.e.,

$$\mathbf{e}_x(t) = \mathbf{x}(t) - \mathbf{m}_x(t)$$

which gives the error dynamics as

$$\dot{\mathbf{e}}_x(t) = \dot{\mathbf{x}}(t) - \dot{\mathbf{m}}_x(t)$$

Substituting for the true dynamics of the state and the dynamics of the mean yields

$$\begin{aligned}\dot{\mathbf{e}}_x(t) &= [\mathbf{F}_x(t)\mathbf{x}(t) + \mathbf{F}_w(t)\mathbf{w}(t)] - [\mathbf{F}_x(t)\mathbf{m}_x(t)] \\ &= \mathbf{F}_x(t)\mathbf{e}_x(t) + \mathbf{F}_w(t)\mathbf{w}(t)\end{aligned}$$

The solution of the linear differential equation for the error is

$$\mathbf{e}_x(t) = \Phi(t, t_{k-1})\mathbf{e}_x(t_{k-1}) + \int_{t_{k-1}}^t \Phi(t, \tau)\mathbf{F}_w(\tau)\mathbf{w}(\tau)d\tau$$

where $\Phi(t, t_{k-1})$ is the state transition matrix that satisfies

$$\dot{\Phi}(t, t_{k-1}) = \mathbf{F}_x(t)\Phi(t, t_{k-1}), \quad \Phi(t_{k-1}, t_{k-1}) = \mathbf{I}_n$$

It is important to note that the time t_{k-1} is purely arbitrary. That is, t_{k-1} can represent any starting condition off of which the evolution of the error is based.

The state estimation error covariance is found via

$$\mathbf{P}_{xx}(t) = \text{E} \left\{ \mathbf{e}_x(t) \mathbf{e}_x^T(t) \right\}$$

Substituting for the expression for the estimation error, it follows that

$$\begin{aligned} \mathbf{P}_{xx}(t) &= \text{E} \left\{ \Phi(t, t_{k-1}) \mathbf{e}_x(t_{k-1}) \mathbf{e}_x^T(t_{k-1}) \Phi^T(t, t_{k-1}) \right. \\ &\quad + \Phi(t, t_{k-1}) \mathbf{e}_x(t_{k-1}) \int_{t_{k-1}}^t \mathbf{w}^T(\tau) \mathbf{F}_w^T(\tau) \Phi^T(t, \tau) d\tau \\ &\quad \left. + \left[\int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbf{w}(\tau) d\tau \right] \mathbf{e}_x^T(t_{k-1}) \Phi^T(t, t_{k-1}) \right\} \end{aligned}$$

$$+ \int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbf{w}(\tau) d\tau \int_{t_{k-1}}^t \mathbf{w}^T(\sigma) \mathbf{F}_w^T(\sigma) \Phi^T(t, \sigma) d\sigma \Big\}$$

Note that we have used σ instead of τ in the last integral. Since τ is a dummy variable, there is nothing preventing us from changing the variable to something different. We also note that the middle two terms of the preceding expression have variables that are not dependent on τ , such that they can be brought into the integrals. Finally, we note that the last term can be rewritten as a double integral, such that these manipulations lead to

$$\begin{aligned} \mathbf{P}_{xx}(t) = & \text{E} \left\{ \Phi(t, t_{k-1}) \mathbf{e}_x(t_{k-1}) \mathbf{e}_x^T(t_{k-1}) \Phi^T(t, t_{k-1}) \right. \\ & + \int_{t_{k-1}}^t \Phi(t, t_{k-1}) \mathbf{e}_x(t_{k-1}) \mathbf{w}^T(\tau) \mathbf{F}_w^T(\tau) \Phi^T(t, \tau) d\tau \\ & + \int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbf{w}(\tau) \mathbf{e}_x^T(t_{k-1}) \Phi^T(t, t_{k-1}) d\tau \\ & \left. + \int_{t_{k-1}}^t \int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbf{w}(\tau) \mathbf{w}^T(\sigma) \mathbf{F}_w^T(\sigma) \Phi^T(t, \sigma) d\sigma d\tau \right\} \end{aligned}$$

At this point, we can now distribute the expectation to each of the four terms, keeping in mind that $\Phi(\cdot, \cdot)$ and $\mathbf{F}_w(\cdot)$ are deterministic, such that

$$\begin{aligned}
\mathbf{P}_{xx}(t) &= \Phi(t, t_{k-1}) \mathbb{E} \left\{ \mathbf{e}_x(t_{k-1}) \mathbf{e}_x^T(t_{k-1}) \right\} \Phi^T(t, t_{k-1}) \\
&\quad + \int_{t_{k-1}}^t \Phi(t, t_{k-1}) \mathbb{E} \left\{ \mathbf{e}_x(t_{k-1}) \mathbf{w}^T(\tau) \right\} \mathbf{F}_w^T(\tau) \Phi^T(t, \tau) d\tau \\
&\quad + \int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbb{E} \left\{ \mathbf{w}(\tau) \mathbf{e}_x^T(t_{k-1}) \right\} \Phi^T(t, t_{k-1}) d\tau \\
&\quad + \int_{t_{k-1}}^t \int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbb{E} \left\{ \mathbf{w}(\tau) \mathbf{w}^T(\sigma) \right\} \mathbf{F}_w^T(\sigma) \Phi^T(t, \sigma) d\sigma d\tau
\end{aligned}$$

From the definition of the estimation error covariance,

$$\mathbf{P}_{xx}(t_{k-1}) = \mathbb{E} \left\{ \mathbf{e}_x(t_{k-1}) \mathbf{e}_x^T(t_{k-1}) \right\}$$

Assume that the process noise is uncorrelated to the state at time t_{k-1} , such that

$$\mathbb{E} \left\{ \mathbf{e}_x(t_{k-1}) \mathbf{w}^T(\tau) \right\} = \mathbf{0} \quad \text{and} \quad \mathbb{E} \left\{ \mathbf{w}(\tau) \mathbf{e}_x^T(t_{k-1}) \right\} = \mathbf{0}$$

From the definition of the process noise power spectral density

$$\mathbb{E} \{ \mathbf{w}(\tau) \mathbf{w}^T(\sigma) \} = \mathbf{Q}_{ww}(\tau) \delta(\tau - \sigma)$$

Applying the previous three relationships to the covariance equation gives

$$\begin{aligned} \mathbf{P}_{xx}(t) &= \Phi(t, t_{k-1}) \mathbf{P}_{xx}(t_{k-1}) \Phi^T(t, t_{k-1}) \\ &\quad + \int_{t_{k-1}}^t \int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbf{Q}_{ww}(\tau) \delta(\tau - \sigma) \mathbf{F}_w^T(\sigma) \Phi^T(t, \sigma) d\sigma d\tau \\ &= \Phi(t, t_{k-1}) \mathbf{P}_{xx}(t_{k-1}) \Phi^T(t, t_{k-1}) \\ &\quad + \int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbf{Q}_{ww}(\tau) \left[\int_{t_{k-1}}^t \mathbf{F}_w^T(\sigma) \Phi^T(t, \sigma) \delta(\tau - \sigma) d\sigma \right] d\tau \end{aligned}$$

We can now apply the sifting property of the Dirac delta to the inner integral of the second term, yielding

$$\mathbf{P}_{xx}(t) = \Phi(t, t_{k-1}) \mathbf{P}_{xx}(t_{k-1}) \Phi^T(t, t_{k-1}) + \int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbf{Q}_{ww}(\tau) \mathbf{F}_w^T(\tau) \Phi^T(t, \tau) d\tau$$

The second term is what we call the process noise covariance matrix, i.e.,

$$\mathbf{P}_{xx}(t) = \Phi(t, t_{k-1}) \mathbf{P}_{xx}(t_{k-1}) \Phi^T(t, t_{k-1}) + \mathbf{P}_{ww}(t)$$

where

$$\mathbf{P}_{ww}(t) = \int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbf{Q}_{ww}(\tau) \mathbf{F}_w^T(\tau) \Phi^T(t, \tau) d\tau$$

As posed, the integral that remains in the expression for the process noise covariance hinders the propagation of the estimation error covariance. To remedy this, consider temporal differentiation of $\mathbf{P}_{ww}(t)$ as

$$\dot{\mathbf{P}}_{ww}(t) = \frac{d}{dt} \left\{ \int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbf{Q}_{ww}(\tau) \mathbf{F}_w^T(\tau) \Phi^T(t, \tau) d\tau \right\}$$

Since the upper limit of integration is a function of time (it is t itself), we must apply Leibniz' rule in order to take the derivative of the integral. This gives us

$$\begin{aligned} \dot{\mathbf{P}}_{ww}(t) &= \int_{t_{k-1}}^t \frac{d}{dt} \left\{ \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbf{Q}_{ww}(\tau) \mathbf{F}_w^T(\tau) \Phi^T(t, \tau) \right\} d\tau \\ &\quad + \Phi(t, t) \mathbf{F}_w(t) \mathbf{Q}_{ww}(t) \mathbf{F}_w^T(t) \Phi^T(t, t) \end{aligned}$$

Recall the properties of the state transition matrix that

$$\dot{\Phi}(t, \tau) = \mathbf{F}_x(t)\Phi(t, \tau) \quad \text{and} \quad \Phi(t, t) = \mathbf{I}_n$$

and note that the only term that is functionally dependent on the variable of differentiation is the state transition matrix, such that

$$\begin{aligned}\dot{\mathbf{P}}_{ww}(t) &= \mathbf{F}_x(t) \int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbf{Q}_{ww}(\tau) \mathbf{F}_w^T(\tau) \Phi^T(t, \tau) d\tau \\ &\quad + \int_{t_{k-1}}^t \Phi(t, \tau) \mathbf{F}_w(\tau) \mathbf{Q}_{ww}(\tau) \mathbf{F}_w^T(\tau) \Phi^T(t, \tau) d\tau \mathbf{F}_x^T(t) + \mathbf{F}_w(t) \mathbf{Q}_{ww}(t) \mathbf{F}_w^T(t)\end{aligned}$$

Finally, using the definition of $\mathbf{P}_{ww}(t)$ to substitute for the two integral terms, it is found that the process noise covariance matrix satisfies the differential equation

$$\dot{\mathbf{P}}_{ww}(t) = \mathbf{F}_x(t) \mathbf{P}_{ww}(t) + \mathbf{P}_{ww}(t) \mathbf{F}_x^T(t) + \mathbf{F}_w(t) \mathbf{Q}_{ww}(t) \mathbf{F}_w^T(t)$$

with the initial condition of $\mathbf{P}_{ww}(t_{k-1}) = \mathbf{0}_{n \times n}$.

This completes one method for propagating the covariance matrix.

Another method comes from differentiating $\mathbf{P}_{xx}(t)$ with respect to time to give

$$\dot{\mathbf{P}}_{xx}(t) = \frac{d}{dt} \left\{ \Phi(t, t_{k-1}) \mathbf{P}_{xx}(t_{k-1}) \Phi^T(t, t_{k-1}) + \mathbf{P}_{ww}(t) \right\}$$

Carrying out the differentiation, we find

$$\dot{\mathbf{P}}_{xx}(t) = \dot{\Phi}(t, t_{k-1}) \mathbf{P}_{xx}(t_{k-1}) \Phi^T(t, t_{k-1}) + \Phi(t, t_{k-1}) \mathbf{P}(t_{k-1}) \dot{\Phi}^T(t, t_{k-1}) + \dot{\mathbf{P}}_{ww}(t)$$

where $\mathbf{P}_{xx}(t_{k-1})$ is a fixed initial condition.

Applying the properties of the state transition matrix and the equation derived for $\dot{\mathbf{P}}_{ww}(t)$, it follows that

$$\begin{aligned} \dot{\mathbf{P}}_{xx}(t) &= \mathbf{F}_x(t) \Phi(t, t_{k-1}) \mathbf{P}_{xx}(t_{k-1}) \Phi^T(t, t_{k-1}) + \Phi(t, t_{k-1}) \mathbf{P}_{xx}(t_{k-1}) \Phi^T(t, t_{k-1}) \mathbf{F}_x^T(t) \\ &\quad + \mathbf{F}_x(t) \mathbf{P}_{ww}(t) + \mathbf{P}_{ww}(t) \mathbf{F}_x^T(t) + \mathbf{F}_w(t) \mathbf{Q}_{ww}(t) \mathbf{F}_w^T(t) \end{aligned}$$

We can collect all terms that are pre-multiplied by $\mathbf{F}_x(t)$, and we can collect terms that are post-multiplied by \mathbf{F}_x^T ; this give us

$$\begin{aligned}\dot{\mathbf{P}}_{xx}(t) &= \mathbf{F}_x(t) [\Phi(t, t_{k-1}) \mathbf{P}_{xx}(t_{k-1}) \Phi^T(t, t_{k-1}) + \mathbf{P}_{ww}(t)] \\ &\quad + [\Phi(t, t_{k-1}) \mathbf{P}_{xx}(t_{k-1}) \Phi^T(t, t_{k-1}) + \mathbf{P}_{ww}(t)] \mathbf{F}_x^T(t) + \mathbf{F}_w(t) \mathbf{Q}_{ww}(t) \mathbf{F}_w^T(t)\end{aligned}$$

The terms in square brackets are simply the estimation error covariance at time t , such that the covariance matrix satisfies the differential equation

$$\dot{\mathbf{P}}_{xx}(t) = \mathbf{F}_x(t) \mathbf{P}_{xx}(t) + \mathbf{P}_{xx}(t) \mathbf{F}_x^T(t) + \mathbf{F}_w(t) \mathbf{Q}_{ww}(t) \mathbf{F}_w^T(t)$$

with an initial condition of $\mathbf{P}_{xx}(t_{k-1}) = \mathbf{P}_{xx,k-1}$.

Now, we have two methods for propagating our covariance.

First method for covariance propagation:

- Propagate state transition matrix (s.t.i.c. $\Phi(t_{k-1}, t_{k-1}) = \mathbf{I}_n$)

$$\dot{\Phi}(t, t_{k-1}) = \mathbf{F}_x(t) \Phi(t, t_{k-1})$$

- Propagate process noise covariance matrix (s.t.i.c. $\mathbf{P}_{ww}(t_{k-1}) = \mathbf{0}_{n \times n}$)

$$\dot{\mathbf{P}}_{ww}(t) = \mathbf{F}_x(t)\mathbf{P}_{ww}(t) + \mathbf{P}_{ww}(t)\mathbf{F}_x^T(t) + \mathbf{F}_w(t)\mathbf{Q}_{ww}(t)\mathbf{F}_w^T(t)$$

- Calculate the propagated covariance matrix

$$\mathbf{P}_{xx}(t) = \Phi(t, t_{k-1})\mathbf{P}_{xx}(t_{k-1})\Phi^T(t, t_{k-1}) + \mathbf{P}_{ww}(t)$$

Second method for covariance propagation:

- Propagate the covariance matrix (s.t.i.c. $\mathbf{P}_{xx}(t_{k-1}) = \mathbf{P}_{xx,k-1}$)

$$\dot{\mathbf{P}}_{xx}(t) = \mathbf{F}_x(t)\mathbf{P}_{xx}(t) + \mathbf{P}_{xx}(t)\mathbf{F}_x^T(t) + \mathbf{F}_w(t)\mathbf{Q}_{ww}(t)\mathbf{F}_w^T(t)$$

In either case, we begin with initial conditions on the mean and covariance from the previous update, i.e.,

$$\mathbf{m}_x(t_{k-1}) = \mathbf{m}_{x,k-1}^+ \quad \text{and} \quad \mathbf{P}_{xx}(t_{k-1}) = \mathbf{P}_{xx,k-1}^+$$

Then, we propagate our equations for the mean and covariance from $t = t_{k-1}$, when the previous update was made, to the time of the next measurement, $t = t_k$. At this point,

the propagated mean and covariance are now called the *a priori* mean and covariance and are given by

$$\mathbf{m}_{x,k}^- = \mathbf{m}_x(t_k) \quad \text{and} \quad \mathbf{P}_{xx,k}^- = \mathbf{P}_{xx}(t_k)$$

At time t_k a measurement is made available, which is given by \mathbf{z}_k . This measurement is a function of the state and is imperfect (noisy). This measurement is taken to be of the form

$$\mathbf{z}_k = \mathbf{H}_{x,k} \mathbf{x}_k + \mathbf{H}_{v,k} \mathbf{v}_k$$

where the measurement noise is represented by \mathbf{v}_k , which is assumed to be a zero mean white-noise sequence with covariance $\mathbf{P}_{vv,k}$, or

$$\mathrm{E}\{\mathbf{v}_k\} = \mathbf{0} \quad \text{and} \quad \mathrm{E}\{\mathbf{v}_k \mathbf{v}_\ell^T\} = \mathbf{P}_{vv,k} \delta_{k\ell}$$

The mean and covariance prior to incorporation of this new information are given by

$$\mathbf{m}_{x,k}^- = \mathrm{E}\{\mathbf{x}_k\}$$

$$\mathbf{P}_{xx,k}^- = \mathrm{E}\left\{(\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{x}_k - \mathbf{m}_{x,k}^-)^T\right\}$$

We want to find a way to use the new information to *update* the mean and covariance of our state, to update our estimated state and our confidence in the estimated state.

The Kalman filter is a *linear* filter, in that the update is constructed using a similar approach to what we did with LUMVE; as such, consider an update of the form

$$\mathbf{m}_{x,k}^+ = \mathbf{a}_k + \mathbf{K}_k \mathbf{z}_k$$

which is the combination of a constant vector and a linear function of the data.

Define the *a priori* and *a posteriori* estimation errors as

$$\mathbf{e}_{x,k}^- = \mathbf{x}_k - \mathbf{m}_{x,k}^- \quad \text{and} \quad \mathbf{e}_{x,k}^+ = \mathbf{x}_k - \mathbf{m}_{x,k}^+$$

From these equations, it is straightforward to show that

$$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{e}_{x,k}^- - \mathbf{e}_{x,k}^+$$

which allows us to equate expressions for the updated mean to find that

$$\mathbf{m}_{x,k}^- + \mathbf{e}_{x,k}^- - \mathbf{e}_{x,k}^+ = \mathbf{a}_k + \mathbf{K}_k \mathbf{z}_k$$

If we take the expected value of this equation under the assumption of an unbiased prior and enforce the condition that we want an unbiased posterior, it follows that

$$\mathbf{m}_{x,k}^- = \mathbf{a}_k + \mathbf{K}_k \mathbf{m}_{z,k}^-$$

where

$$\mathbf{m}_{z,k}^- = \text{E}\{\mathbf{z}_k\}$$

is the expected value of the measurement with respect to any non-deterministic inputs prior to the inclusion of this new piece of information.

Now, we can solve for the \mathbf{a}_k that guarantees an unbiased posterior estimate, which is given by

$$\mathbf{a}_k = \mathbf{m}_{x,k}^- - \mathbf{K}_k \mathbf{m}_{z,k}^-$$

Our update equation can now be expressed as

$$\begin{aligned}\mathbf{m}_{x,k}^+ &= \mathbf{m}_{x,k}^- - \mathbf{K}_k \mathbf{m}_{z,k}^- + \mathbf{K}_k \mathbf{z}_k \\ &= \mathbf{m}_{x,k}^- + \mathbf{K}_k [\mathbf{z}_k - \mathbf{m}_{z,k}^-]\end{aligned}$$

Note that no specification of linearity of the measurement process needs to be made for this equation to hold; $\mathbf{m}_{z,k}^-$ is simply the mean of the measurement with respect to the state and noise distributions.

However, if we make the specification that the measurement is linear, then

$$\mathbf{m}_{z,k}^- = \text{E}\{\mathbf{H}_{x,k}\mathbf{x}_k + \mathbf{H}_{v,k}\mathbf{v}_k\} = \mathbf{H}_{x,k}\mathbf{m}_{x,k}^-$$

and the update reduces to

$$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k [\mathbf{z}_k - \mathbf{H}_{x,k}\mathbf{m}_{x,k}^-]$$

In the future, we will make use of the more general form of the update that does not require the linear measurement condition.

With this update, we now want to examine the characteristics of the estimation error, where the goal is to determine the posterior estimation error covariance.

The *a posteriori* state estimation error is

$$\mathbf{e}_{x,k}^+ = \mathbf{e}_{x,k}^- - \mathbf{K}_k (\mathbf{z}_k - \mathbf{m}_{z,k}^-)$$

or, by substituting for the definitions of the errors,

$$(\mathbf{x}_k - \mathbf{m}_{x,k}^+) = (\mathbf{x}_k - \mathbf{m}_{x,k}^-) - \mathbf{K}_k(\mathbf{z}_k - \mathbf{m}_{z,k}^-)$$

Let $\mathbf{P}_{xx,k}^-$ and $\mathbf{P}_{xx,k}^+$ be defined as

$$\mathbf{P}_{xx,k}^- = \text{E} \left\{ (\mathbf{e}_{x,k}^-)(\mathbf{e}_{x,k}^-)^T \right\} \quad \text{and} \quad \mathbf{P}_{xx,k}^+ = \text{E} \left\{ (\mathbf{e}_{x,k}^+)(\mathbf{e}_{x,k}^+)^T \right\}$$

Substituting from the estimation error, it follows that

$$\begin{aligned} \mathbf{P}_{xx,k}^+ &= \text{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{x}_k - \mathbf{m}_{x,k}^-)^T - (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \mathbf{K}_k^T \right. \\ &\quad \left. - \mathbf{K}_k(\mathbf{z}_k - \mathbf{m}_{z,k}^-)(\mathbf{x}_k - \mathbf{m}_{x,k}^-)^T + \mathbf{K}_k(\mathbf{z}_k - \mathbf{m}_{z,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \mathbf{K}_k^T \right\} \end{aligned}$$

We can now distribute the expectation to each term in the preceding expression. We will also make a key assumption at this point that the gain matrix is deterministic. It then

follows that the *a posteriori* covariance is

$$\begin{aligned}\mathbf{P}_{xx,k}^+ &= \mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{x}_k - \mathbf{m}_{x,k}^-)^T \right\} - \mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \right\} \mathbf{K}_k^T \\ &\quad - \mathbf{K}_k \mathbb{E} \left\{ (\mathbf{z}_k - \mathbf{m}_{z,k}^-)(\mathbf{x}_k - \mathbf{m}_{x,k}^-)^T \right\} + \mathbf{K}_k \mathbb{E} \left\{ (\mathbf{z}_k - \mathbf{m}_{z,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \right\} \mathbf{K}_k^T\end{aligned}$$

Let the prior state covariance, cross-covariance (with the measurement), and measurement covariance (also called the innovation or residual covariance) be defined as

$$\mathbf{P}_{xx,k}^- = \mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{x}_k - \mathbf{m}_{x,k}^-)^T \right\}$$

$$\mathbf{P}_{xz,k}^- = \mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \right\}$$

$$\mathbf{P}_{zz,k}^- = \mathbb{E} \left\{ (\mathbf{z}_k - \mathbf{m}_{z,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \right\}$$

Substituting for the above relationships, the covariance update becomes

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- \mathbf{K}_k^T - \mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

Note that no specification of linearity of the measurement process needs to be made for this equation to hold. We did, however, have to assume that the gain is deterministic, and this will become important later on. For now, just keep this in mind.

It is also worth noting that no particular form for the gain matrix, \mathbf{K}_k , has been specified. This means that our expression for the covariance update holds for whatever gain we use, so long as the update is linear. Such a gain is what we will refer to as a “linear gain.”

The Kalman filter belongs to the class of linear minimum mean square error estimators. The linear aspect is addressed through the use of a linear update relationship; we will now specify the gain \mathbf{K}_k to be the one that minimizes the mean square of the *a posteriori* state estimation error; i.e., the gain is determined by minimizing the cost function

$$J = \text{E} \left\{ (\mathbf{e}_{x,k}^+)^T (\mathbf{e}_{x,k}^+) \right\} = \text{trace E} \left\{ (\mathbf{e}_{x,k}^+) (\mathbf{e}_{x,k}^+)^T \right\} = \text{trace } \mathbf{P}_{xx,k}^+$$

Now, we substitute for our form of the posterior covariance matrix to find that

$$\begin{aligned} J &= \text{trace } \{\mathbf{P}_{xx,k}^-\} - \text{trace } \{\mathbf{P}_{xz,k}^- \mathbf{K}_k^T\} - \text{trace } \{\mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T\} + \text{trace } \{\mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T\} \\ &= \text{trace } \{\mathbf{P}_{xx,k}^-\} - 2 \text{trace } \{\mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T\} + \text{trace } \{\mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T\} \end{aligned}$$

To proceed, we need to know how to take the derivative of the trace of a matrix. For the derivatives that we require, it can be shown that

$$\frac{\partial \text{trace}\{\mathbf{BAC}\}}{\partial \mathbf{A}} = \mathbf{B}^T \mathbf{C} \quad \text{and} \quad \frac{\partial \text{trace}\{\mathbf{ABA}^T\}}{\partial \mathbf{A}} = \mathbf{A}[\mathbf{B} + \mathbf{B}^T]$$

Now, we can take the derivative of our performance index in a term-by-term fashion and put the pieces together to get the derivative of the performance index with respect to the gain matrix as

$$\frac{\partial J}{\partial \mathbf{K}_k} = \mathbf{0}_{n \times m} - 2\mathbf{P}_{xz,k}^- + \mathbf{K}_k [\mathbf{P}_{zz,k}^- + (\mathbf{P}_{zz,k}^-)^T] = -2\mathbf{P}_{xz,k}^- + 2\mathbf{K}_k \mathbf{P}_{zz,k}^-$$

where the last equality follows from the fact that $\mathbf{P}_{zz,k}^-$ is symmetric. Therefore, the gain that renders the performance index stationary is given by the solution to

$$\frac{\partial J}{\partial \mathbf{K}_k} = -2\mathbf{P}_{xz,k}^- + 2\mathbf{K}_k \mathbf{P}_{zz,k}^- = \mathbf{0}_{n \times m}$$

which yields the Kalman gain as

$$\mathbf{K}_k = \mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1}$$

The Kalman gain is a particular choice of the linear gain, but so far, it is only shown to satisfy the first-derivative condition for a minimum. Therefore, we still need to determine if this gain actually minimizes the cost function.

To show this, we will consider a different gain and show that any other choice of the linear gain leads to a larger value of the cost function. As such, consider a new linear gain given by $\bar{\mathbf{K}}_k = \mathbf{K}_k + \Delta\mathbf{K}_k$. In this case, we know that the posterior covariance (remember that our posterior covariance equation is valid for *any* linear gain) is

$$\bar{\mathbf{P}}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- \bar{\mathbf{K}}_k^T - \bar{\mathbf{K}}_k (\mathbf{P}_{xz,k}^-)^T + \bar{\mathbf{K}}_k \mathbf{P}_{zz,k}^- \bar{\mathbf{K}}_k^T$$

Now, we simply apply the new gain matrix and expand out terms to find

$$\begin{aligned} \bar{\mathbf{P}}_{xx,k}^+ &= \mathbf{P}_k^- - \mathbf{P}_{xz,k}^- \mathbf{K}_k^T - \mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T - \mathbf{P}_{xz,k}^- \Delta\mathbf{K}_k^T \\ &\quad - \Delta\mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T + \Delta\mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T + \mathbf{K}_k \mathbf{P}_{zz,k}^- \Delta\mathbf{K}_k^T + \Delta\mathbf{K}_k \mathbf{P}_{zz,k}^- \Delta\mathbf{K}_k^T \end{aligned}$$

Recognizing that the first four terms in the preceding express are exactly our posterior covariance when the gain \mathbf{K}_k is used, and denoting it as $\mathbf{P}_{xx,k}^+$ still, it follows that

$$\bar{\mathbf{P}}_{xx,k}^+ = \mathbf{P}_{xx,k}^+ - \mathbf{P}_{xz,k}^- \Delta\mathbf{K}_k^T - \Delta\mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T$$

$$+ \Delta \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T + \mathbf{K}_k \mathbf{P}_{zz,k}^- \Delta \mathbf{K}_k^T + \Delta \mathbf{K}_k \mathbf{P}_{zz,k}^- \Delta \mathbf{K}_k^T$$

Recall that \mathbf{K}_k is the Kalman gain, for which we have an expression. Applying the known relationship for the Kalman gain and reducing yields

$$\bar{\mathbf{P}}_{xx,k}^+ = \mathbf{P}_{xx,k}^+ + \Delta \mathbf{K}_k \mathbf{P}_{zz,k}^- \Delta \mathbf{K}_k^T$$

To show that this leads to a higher cost, we take the trace

$$\text{trace } \bar{\mathbf{P}}_{xx,k}^+ = \text{trace } \mathbf{P}_{xx,k}^+ + \text{trace } \{\Delta \mathbf{K}_k \mathbf{P}_{zz,k}^- \Delta \mathbf{K}_k^T\}$$

Since $\mathbf{P}_{zz,k}^- > \mathbf{0}$, $\Delta \mathbf{K}_k \mathbf{P}_{zz,k}^- \Delta \mathbf{K}_k^T \geq \mathbf{0}$, which means that

$$\text{trace } \{\Delta \mathbf{K}_k \mathbf{P}_{zz,k}^- \Delta \mathbf{K}_k^T\} > 0$$

which means that

$$\text{trace } \bar{\mathbf{P}}_{xx,k}^+ \geq \text{trace } \mathbf{P}_{xx,k}^+$$

Thus, there is no linear gain that leads to a lower cost than what is produced by using the Kalman gain; thus, we can conclude that the Kalman gain does indeed minimize the

posterior mean square error.

To apply the Kalman filter, the measurement-dependent quantities

$$\mathbf{m}_{z,k}^- = \text{E} \{ \mathbf{z}_k \}$$

$$\mathbf{P}_{xz,k}^- = \text{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \right\}$$

$$\mathbf{P}_{zz,k}^- = \text{E} \left\{ (\mathbf{z}_k - \mathbf{m}_{z,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \right\}$$

are needed. These expressions are very general, but they require expectations. For certain measurement models, we can compute the expectations analytically.

Consider the case where the measurement is linear in the state and subjected to additive measurement noise via

$$\mathbf{z}_k = \mathbf{H}_{x,k} \mathbf{x}_k + \mathbf{H}_{v,k} \mathbf{v}_k$$

where the first- and second-moment statistics of the measurement noise are

$$\text{E} \{ \mathbf{v}_k \} = \mathbf{0} \quad \text{and} \quad \text{E} \{ \mathbf{v}_k \mathbf{v}_\ell^T \} = \mathbf{P}_{vv,k} \delta_{k\ell}$$

Taking the expected value of both sides of the measurement model, taking both $\mathbf{H}_{x,k}$ and $\mathbf{H}_{v,k}$ to be deterministic, and recalling that the measurement noise is zero mean, it follows that

$$\mathbf{m}_{z,k}^- = \mathbf{H}_{x,k} \mathbb{E}\{\mathbf{x}_k\}$$

or, using the expected value of the state, we have

$$\mathbf{m}_{z,k}^- = \mathbf{H}_{x,k} \mathbf{m}_{x,k}^-$$

Now, consider the cross-covariance, which is defined as

$$\mathbf{P}_{xz,k}^- = \mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \right\}$$

Looking first at the term $(\mathbf{z}_k - \mathbf{m}_{z,k}^-)$ and substituting from the measurement model and expected measurement, it follows that

$$\mathbf{z}_k - \mathbf{m}_{z,k}^- = \mathbf{H}_{x,k}(\mathbf{x}_k - \mathbf{m}_{x,k}^-) + \mathbf{H}_{v,k}\mathbf{v}_k$$

Thus, the cross-covariance becomes

$$\mathbf{P}_{xz,k}^- = \mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{x}_k - \mathbf{m}_{x,k}^-)^T \mathbf{H}_{x,k}^T \right\} + \mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)\mathbf{v}_k^T \mathbf{H}_{v,k}^T \right\}$$

Since $\mathbf{H}_{x,k}$ and $\mathbf{H}_{v,k}$ are taken to be deterministic,

$$\mathbf{P}_{xz,k}^- = \mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{x}_k - \mathbf{m}_{x,k}^-)^T \right\} \mathbf{H}_{x,k}^T + \mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-) \mathbf{v}_k^T \right\} \mathbf{H}_{v,k}^T$$

Assuming that the state is not correlated to the measurement noise, i.e.,

$$\mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-) \mathbf{v}_k^T \right\} = \mathbf{0}_{n \times m}$$

it follows that the cross-covariance for linear measurements with additive noise is

$$\mathbf{P}_{xz,k}^- = \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T$$

Finally, consider the measurement covariance (also known as the innovation covariance or the residual covariance)

$$\mathbf{P}_{zz,k}^- = \mathbb{E} \left\{ (\mathbf{z}_k - \mathbf{m}_{z,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \right\}$$

Using the previously developed result of

$$\mathbf{z}_k - \mathbf{m}_{z,k}^- = \mathbf{H}_{x,k}(\mathbf{x}_k - \mathbf{m}_{x,k}^-) + \mathbf{H}_{v,k}\mathbf{v}_k$$

and recalling the previous properties/assumptions that

- $\mathbf{H}_{x,k}$ and $\mathbf{H}_{v,k}$ are deterministic
- the state is not correlated with the measurement noise
- the covariance of the measurement noise is given by $\mathbf{P}_{vv,k}$

gives the innovation covariance for linear measurements as

$$\mathbf{P}_{zz,k}^- = \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T$$

This completes the Kalman filter!

As a note, when a distinction is made between the innovation covariance and the residual covariance, the former is computed using *a priori* values, and the latter is computed using *a posteriori* values. As such, the residual covariance would be given by

$$\mathbf{P}_{zz,k}^+ = \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^+ \mathbf{H}_{x,k}^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T$$

This quantity is not leveraged in operation of a Kalman filter, but it is sometimes used in the analysis of the filter.

To summarize, we put everything together in a single table

System Model	$\dot{\mathbf{x}}(t) = \mathbf{F}_x(t)\mathbf{x}(t) + \mathbf{F}_w(t)\mathbf{w}(t)$
Meas. Model	$\mathbf{z}_k = \mathbf{H}_{x,k}\mathbf{x}_k + \mathbf{H}_{v,k}\mathbf{v}_k$
Init. Cond.	$\mathbf{m}_{x,0} = \text{E}\{\mathbf{x}(t_0)\}$ $\mathbf{P}_{xx,0} = \text{E}\{(\mathbf{x}(t_0) - \mathbf{m}_{x,0})(\mathbf{x}(t_0) - \mathbf{m}_{x,0})^T\}$
Mean Prop.	$\dot{\mathbf{m}}_x(t) = \mathbf{F}_x(t)\mathbf{m}_x(t)$
Cov. Prop.	$\dot{\mathbf{P}}_{xx}(t) = \mathbf{F}_x(t)\mathbf{P}_{xx}(t) + \mathbf{P}_{xx}(t)\mathbf{F}_x^T(t) + \mathbf{F}_w(t)\mathbf{Q}_{ww}(t)\mathbf{F}_w^T(t)$
Exp. Meas.	$\mathbf{m}_{z,k}^- = \mathbf{H}_{x,k}\mathbf{m}_{x,k}^-$
Cross Cov.	$\mathbf{P}_{xz,k}^- = \mathbf{P}_{xx,k}^-\mathbf{H}_{x,k}^T$
Innov. Cov.	$\mathbf{P}_{zz,k}^- = \mathbf{H}_{x,k}\mathbf{P}_{xx,k}^-\mathbf{H}_{x,k}^T + \mathbf{H}_{v,k}\mathbf{P}_{vv,k}\mathbf{H}_{v,k}^T$
Kalman Gain	$\mathbf{K}_k = \mathbf{P}_{xz,k}^-(\mathbf{P}_{zz,k}^-)^{-1}$
Mean Upd.	$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k[\mathbf{z}_k - \mathbf{m}_{z,k}^-]$
Cov. Upd.	$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^-\mathbf{K}_k^T - \mathbf{K}_k(\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k\mathbf{P}_{zz,k}^-\mathbf{K}_k^T$

3.1.1 Car Example

Suppose you are tasked with characterizing the performance of a “constant acceleration” electric motor in a car, and to accomplish this task, you set up an experiment where the vehicle accelerates starting from rest and a laser rangefinder provides data on the motion of the car through time. The range measurements are used to estimate the car’s range and its first two associated rates at any point during its 10 second test. As such, define the state vector to be

$$\mathbf{x}^T = [\rho \ \dot{\rho} \ \ddot{\rho}]$$

The company’s engineer tells you that the engine should accelerate at 7 m/s^2 , so we take the initial estimate to be

$$\mathbf{m}_{x,0}^T = [0 \ 0 \ 7]$$

According to the system model for the Kalman filter, we will add some error to this mean to generate the initial truth in simulation. As such, we need to have some initial uncertainty. We will take the truth to be Gaussian-distributed with the mean as given previously and the covariance to be (with appropriate units)

$$\mathbf{P}_{xx,0} = \text{diag}\{14^2 \ 5^2 \ 1^2\}$$

Constant acceleration of the vehicle tells us that

$$\mathbf{F}_x(t) = \mathbf{F}_x = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Additionally, to account for noise in our dynamic model, we define the process noise power spectral density and noise mapping matrix to be

$$\mathbf{Q}_{ww}(t) = 0.01 \quad \text{and} \quad \mathbf{F}_w(t) = [0 \ 0 \ 1]^T$$

It is important to note that we will not use the process noise in our truth generation. Instead, we will use it as it is most often used... to inflate the covariance matrix during propagation.

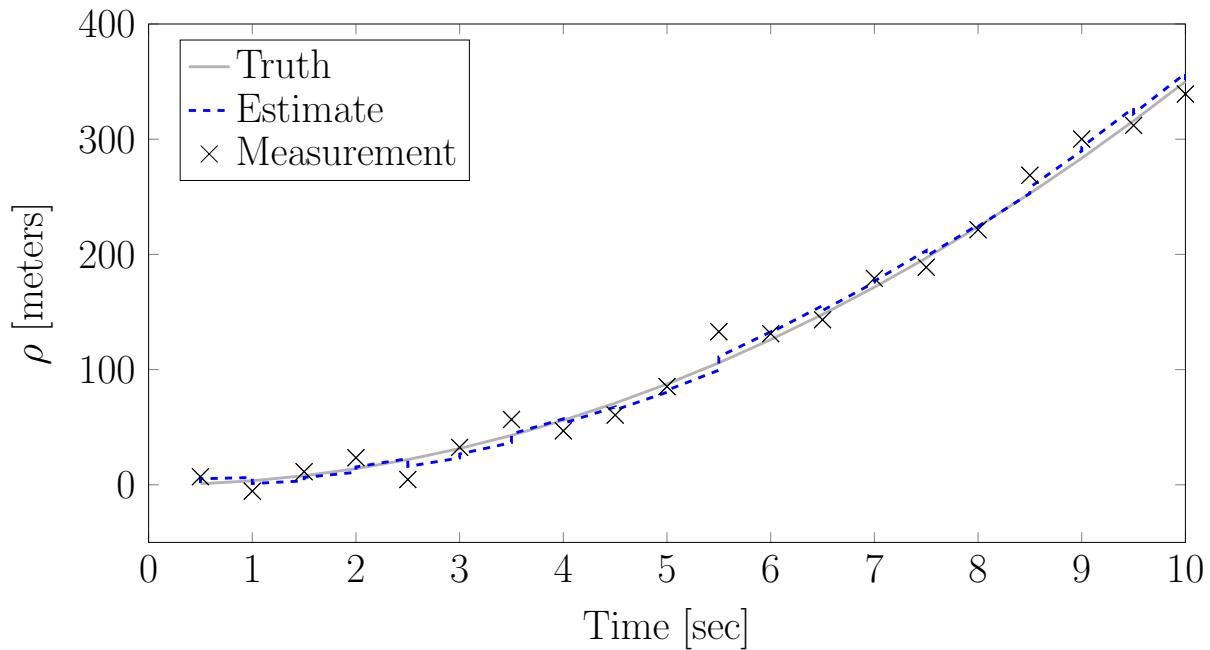
The rangefinder generates noisy range measurements with a measurement noise standard deviation of 10 m at a rate of 2 Hz. This tells us that

$$\mathbf{P}_{vv,k} = 10^2$$

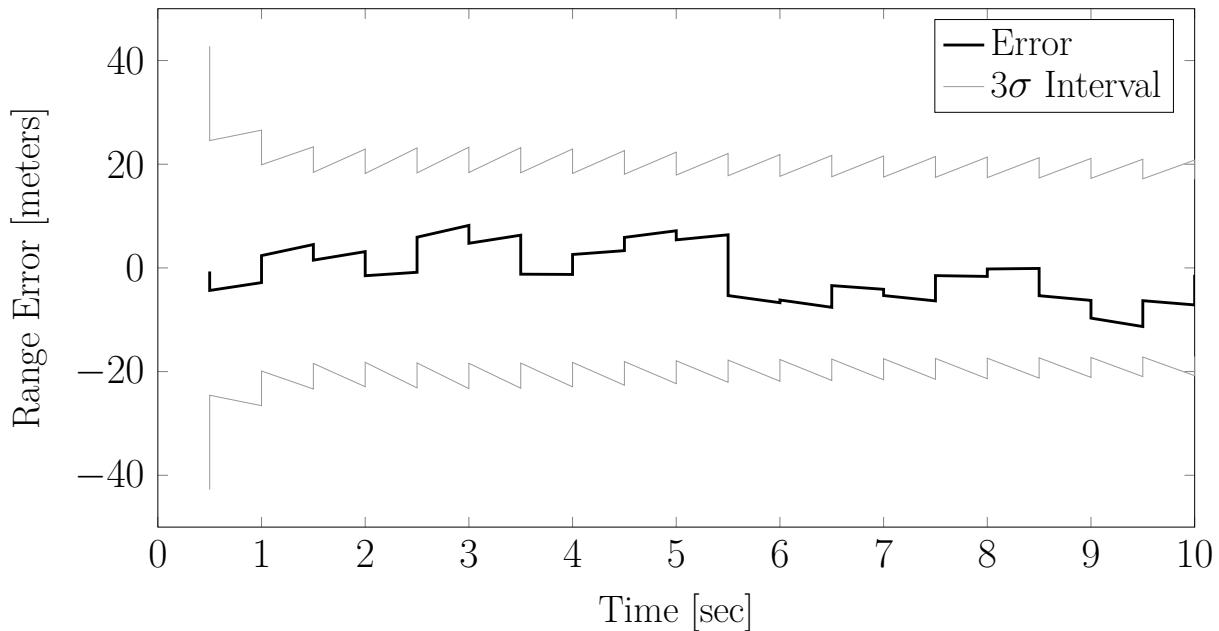
with appropriate units. Since we are only taking range measurements with additive noise,

$$\mathbf{H}_{x,k} = [1 \ 0 \ 0] \quad \text{and} \quad \mathbf{H}_{v,k} = 1$$

The measured, true, and estimated range of the car can be seen in the figure below. It seems like our Kalman filter did its job! It tracks along the trajectory according to our measurements.



More appropriate for evaluating the performance of the filter, perhaps, is the error in our estimate. This can be seen plotted below.



Note that the error in our estimate tends to stay within about 10 meters. Here σ is the square root of the (1, 1) element of the covariance matrix at each step (that is, the

element of the state estimation covariance which corresponds to the variance in ρ).

It is common to plot these curves for each state element, although we only have the first state shown in this example. Generally, we are looking for the estimation error to stay within the 3σ interval most of the time. It is also worth noting that we are plotting both the prior and the posterior estimation error and 3σ interval. This allows us to visualize what is happening during the update of the Kalman filter.

Let's take a look at how we code this Kalman filter in MATLAB.

Define a random number seed, integrator options, and timing parameters:

```
% Set random number seed
rng(100)

% Integrator options
opts = odeset('AbsTol',1e-6,'RelTol',1e-6);

% Simulation timing information
dt    = 0.5;
tv    = 0:dt:100;
```

Specify initial conditions for the filter, and generate a random true initial state:

```
% Initial estimate and random truth  
mx0 = [0.0;0.0;7.0];  
Pxx0 = diag([14.0,5.0,1.0].^2);  
x0 = mx0 + chol(Pxx0) '*randn(3,1);
```

The dynamics and measurements models get specified next:

```
% Dynamics model  
Fx = [0.0,1.0,0.0;0.0,0.0,1.0;0.0,0.0,0.0];  
Fw = [0.0;0.0;1.0];  
Qww = 1e-2;  
  
% Measurement model  
Hx = [1.0,0.0,0.0];  
Hv = 1.0;  
Pvv = 10.0^2;
```

We set up storage space and counters to help us store results for analysis:

```
% Initialize storage space for analysis
xcount = 0;
xstore = zeros(3,2*length(tv)-1);
txstore = zeros(1,2*length(tv)-1);
mxstore = zeros(3,2*length(tv)-1);
sxstore = zeros(3,2*length(tv)-1);
zcount = 0;
zstore = zeros(1,length(tv)-1);
tzstore = zeros(1,length(tv)-1);
mzstore = zeros(1,length(tv)-1);
szstore = zeros(1,length(tv)-1);
```

The first thing we store is information about the initial truth and estimate:

```
% Store initial data
xcount = xcount + 1;
xstore(:,xcount) = x0;
txstore(:,xcount) = tv(1);
mxstore(:,xcount) = mx0;
sxstore(:,xcount) = sqrt(diag(Pxx0));
```

We now initialize variables to start running the filter, and loop over time:

```
% Initialize variables for filter
xkm1 = x0;
mxkm1 = mx0;
Pxxkm1 = Pxx0;

% Loop over time vector
for i = 2:length(tv)
```

Propagate the truth from t_{k-1} to t_k :

```
% Propagate truth
[~, X] = ode45(@car_eoms, [tv(i-1), tv(i)], xkm1, opts, Fx, Qww, Fw);
xk = X(end, 1:3);
```

Generate a noisy measurement at t_k :

```
% Generate measurement
zk = Hx*xk + chol(Pvv) '*randn;
```

Predict according to the Kalman filter equations:

```
% Propagate step of Kalman filter
[~,X] = ode45(@car_eoms,[tv(i-1),tv(i)],[mxkm;Pxxkm(:, :)],opts,Fx,Qww,Fw);
mxkm = X(end,1:3)';
Pxxkm = reshape(X(end,4:end)',3,3);
```

Store the *a priori* mean and standard deviations to look at later:

```
% Store a priori mean and covariance
xcount = xcount + 1;
xstore(:,xcount) = xk;
txstore(:,xcount) = tv(i);
mxstore(:,xcount) = mxkm;
sxstore(:,xcount) = sqrt(diag(Pxxkm));
```

Update according to the Kalman filter equations:

```
% Update step of Kalman filter
mzkm = Hx*mxkm;
Pxzkm = Pxxkm*Hx';
Pzzkm = Hx*Pxxkm*Hx' + Hv*Pvv*Hv';
Kk = Pxzkm/Pzzkm;
mxkp = mxkm + Kk*(zk - mzkm);
Pxxkp = Pxxkm - Pxzkm*Kk' - Kk*Pxzkm' + Kk*Pzzkm*Kk';
```

Store the *a posteriori* mean and standard deviations to look at later:

```
% Store a posteriori mean and stdevs
xcount = xcount + 1;
xstore(:,xcount) = xk;
txstore(:,xcount) = tv(i);
mxstore(:,xcount) = mxkp;
sxstore(:,xcount) = sqrt(diag(Pxxkp));
```

Also store predicted measurement and standard deviations to look at later:

```
% Store data, estimate, and stdev
zcount = zcount + 1;
zstore(:,zcount) = zk;
tzstore(:,zcount) = tv(i);
mzstore(:,zcount) = mzkm;
szstore(:,zcount) = sqrt(diag(Pzzkm));
```

Set up the recursion for the next time step (and don't forget to close that loop).

```
% Cycle for the next time step
xkm1 = xk;
mxkm1 = mxkp;
Pxxkm1 = Pxxkp;
end
```

At this point, we have completed the loop of propagating between measurements and processing the acquired measurement data. In doing so, we have stored the true state of the system, our estimated mean and covariance (via standard deviations). We have also

stored the true data that we simulated, our predictions of that data, and our innovation covariance (via standard deviations). This information gives us the ability to analyze several aspects of the filter that we've applied to this problem.

The filter also relies on a file containing the equations of motion. For this problem, that file is:

```
function [dxdt] = car_eoms(t,x,Fx,Qww,Fw)

% dynamics of the problem
dxdt = Fx*x(1:3);

% if propagating the covariance, the state is larger
if length(x) > 3
    % extract covariance from the integration state
    P      = reshape(x(4:end)',3,3);
    % covariance dynamics
    dPdt = Fx*P + P*Fx' + Fw*Qww*Fw';
    % append output with covariance time derivative
    dxdt = [dxdt; dPdt(:)];
end
```

3.1.2 Variations on the Covariance Update

Let's look at a few alternative forms of the covariance update.

Our original covariance update equation is

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- \mathbf{K}_k^T - \mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

It is important to remember that no form of the gain, \mathbf{K}_k , was specified, other than it be a linear gain, in arriving at the preceding equation.

In the following variations, we will explore what happens to the covariance update equation when we utilize linear measurements and/or the Kalman gain.

For the first variation, we will assume linear measurements, such that

$$\mathbf{P}_{xz,k}^- = \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T$$

$$\mathbf{P}_{zz,k}^- = \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T$$

Next, we will substitute these expressions for $\mathbf{P}_{xz,k}^-$ and $\mathbf{P}_{zz,k}^-$ into the covariance update equation, giving

$$\begin{aligned}\mathbf{P}_{xx,k}^+ &= \mathbf{P}_{xx,k}^- - \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T \mathbf{K}_k - \mathbf{K}_k \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \\ &\quad + \mathbf{K}_k \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T \mathbf{K}_k^T + \mathbf{K}_k \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T \mathbf{K}_k^T\end{aligned}$$

Finally, a simple rearrangement produces

$$\mathbf{P}_{xx,k}^+ = [\mathbf{I}_n - \mathbf{K}_k \mathbf{H}_{x,k}] \mathbf{P}_{xx,k}^- [\mathbf{I}_n - \mathbf{K}_k \mathbf{H}_{x,k}]^T + \mathbf{K}_k \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T \mathbf{K}_k^T$$

which is known as the Joseph form of the covariance update. We still have not specifically given a value of the linear gain for this equation to be valid, but we have required the use of linear measurements to arrive at this equation.

For our second variation, we will assume linear measurements and we will leverage the Kalman gain instead of an unspecified linear gain. We once again starting from the original covariance update given by

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- \mathbf{K}_k^T - \mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

Recall that the Kalman gain is given by

$$\mathbf{K}_k = \mathbf{P}_{xz,k}^-(\mathbf{P}_{zz,k}^-)^{-1} \quad \text{such that} \quad \mathbf{K}_k \mathbf{P}_{zz,k}^- = \mathbf{P}_{xz,k}^-$$

Substituting this expression into the last term of the covariance update gives

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- \mathbf{K}_k^T - \mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T + \mathbf{P}_{xz,k}^- \mathbf{K}_k^T$$

from which we see that the second and fourth terms are equal and opposite. Canceling these terms and substituting for the expression for the cross-covariance for linear measurements, it follows that

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{K}_k \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^-$$

Finally, factor our the prior covariance matrix

$$\mathbf{P}_{xx,k}^+ = [\mathbf{I}_n - \mathbf{K}_k \mathbf{H}_{x,k}] \mathbf{P}_{xx,k}^-$$

This is the “standard” form of the covariance update that is usually seen in most papers/books. It is also a form that we discovered in the sequentialization of LUMVE. This form requires the Kalman gain to be the gain that is used, and it also requires linear

measurements.

For our last alternative form of the covariance update, we will leverage the Kalman gain, but we will avoid making the assumption of linear measurements. Starting again from the original covariance update of

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- \mathbf{K}_k^T - \mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

and substituting everywhere for \mathbf{K}_k as the Kalman gain, it follows that

$$\begin{aligned}\mathbf{P}_{xx,k}^+ &= \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- (\mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1})^T - \mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1} (\mathbf{P}_{xz,k}^-)^T \\ &\quad + \mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1} \mathbf{P}_{zz,k}^- (\mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1})^T \\ &= \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1} (\mathbf{P}_{xz,k}^-)^T \\ &= \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1} \mathbf{P}_{zz,k}^- (\mathbf{P}_{zz,k}^-)^{-1} (\mathbf{P}_{xz,k}^-)^T\end{aligned}$$

where the last equality follows from inserting an identity matrix in the form of $\mathbf{P}_{zz,k}^- (\mathbf{P}_{zz,k}^-)^{-1}$ into the second term. Now, we recognize that the Kalman gain can be used twice in the

second term, leading to

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

This form of the covariance update relies on the Kalman gain, but it does not require the assumption of a linear measurement.

All of these forms are algebraically equivalent (assuming that the same assumptions are applied to each one), but they have slightly different numerical properties.

3.1.3 A Property of the Innovation/Residual

A common quantity to investigate in the Kalman filter is the innovation/residual, which, for linear measurements, is defined as

$$\mathbf{e}_{z,k}^- = \mathbf{z}_k - \mathbf{H}_{x,k} \mathbf{m}_{x,k}^-$$

Here, we will make the distinction that the innovation is computed using the *a priori* estimated state, whereas the residual is computed using the *a posteriori* estimated state.

The innovation is zero mean, which is easily shown by taking the expected value, substituting for the measurement model, and assuming that the mapping matrices are deterministic; that is,

$$\begin{aligned}
 \mathbb{E}\{\boldsymbol{e}_{z,k}^-\} &= \mathbb{E}\{\boldsymbol{z}_k - \mathbf{H}_k \boldsymbol{m}_k^-\} \\
 &= \mathbb{E}\{\mathbf{H}_{x,k} \boldsymbol{x}_k + \mathbf{H}_{v,k} \boldsymbol{v}_k - \mathbf{H}_k \boldsymbol{m}_k^-\} \\
 &= \mathbb{E}\{\mathbf{H}_{x,k} \boldsymbol{e}_k^- + \mathbf{H}_{v,k} \boldsymbol{v}_k\} \\
 &= \mathbf{H}_{x,k} \mathbb{E}\{\boldsymbol{e}_k^-\} + \mathbf{H}_{v,k} \mathbb{E}\{\boldsymbol{v}_k\} \\
 &= \mathbf{0}_m
 \end{aligned}$$

The last equality follows from the fact that we have constructed an unbiased estimator and the fact that the measurement noise is zero mean.

We have also found that the innovations covariance for linear measurements is

$$\mathbf{P}_{zz,k}^- = \mathbb{E}\{\boldsymbol{e}_{z,k}^- (\boldsymbol{e}_{z,k}^-)^T\} = \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k}^- \mathbf{H}_{v,k}^T$$

Now, let's define the residual (also sometimes called the post-fit residual) as

$$\mathbf{e}_{z,k}^+ = \mathbf{z}_k - \mathbf{H}_{x,k} \mathbf{m}_{x,k}^+$$

From the update equation for the Kalman filter, it follows that

$$\begin{aligned}\mathbf{e}_{z,k}^+ &= \mathbf{z}_k - \mathbf{H}_{x,k} [\mathbf{m}_{x,k}^- + \mathbf{K}_k (\mathbf{z}_k - \mathbf{H}_{x,k} \mathbf{m}_{x,k}^-)] \\ &= [\mathbf{z}_k - \mathbf{H}_{x,k} \mathbf{m}_{x,k}^-] - \mathbf{H}_{x,k} \mathbf{K}_k [\mathbf{z}_k - \mathbf{H}_{x,k} \mathbf{m}_{x,k}^-] \\ &= [\mathbf{I}_m - \mathbf{H}_{x,k} \mathbf{K}_k] [\mathbf{z}_k - \mathbf{H}_{x,k} \mathbf{m}_{x,k}^-] \\ &= [\mathbf{I}_m - \mathbf{H}_{x,k} \mathbf{K}_k] \mathbf{e}_{z,k}^-\end{aligned}$$

Recalling that the residual is zero mean and assuming that \mathbf{H}_k and \mathbf{K}_k are deterministic, it follows directly that the post-fit residual is zero mean, i.e.,

$$\mathbb{E}\{\mathbf{e}_{z,k}^+\} = \mathbf{0}_m$$

It then follows from the expression for the post-fit residual that the post-fit residual covariance is

$$\mathbf{P}_{zz,k}^+ = \mathbb{E}\{\mathbf{e}_{z,k}^+ (\mathbf{e}_{z,k}^+)^T\} = [\mathbf{I}_m - \mathbf{H}_{x,k} \mathbf{K}_k] \mathbf{P}_{zz,k}^- [\mathbf{I}_m - \mathbf{H}_{x,k} \mathbf{K}_k]^T$$

This is one possible expression for the post-fit residual covariance, but we can also come up with a slightly more useful expression.

Let's go back to our expression for the post-fit residual and substitute for the Kalman gain

$$\begin{aligned}
 \mathbf{e}_{z,k}^+ &= [\mathbf{I}_m - \mathbf{H}_{x,k} \mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1}] \mathbf{e}_{z,k}^- \\
 &= [\mathbf{P}_{zz,k}^- - \mathbf{H}_{x,k} \mathbf{P}_{xz,k}^-] (\mathbf{P}_{zz,k}^-)^{-1} \mathbf{e}_{z,k}^- \\
 &= [\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k}^- \mathbf{H}_{v,k}^T - \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T] (\mathbf{P}_{zz,k}^-)^{-1} \mathbf{e}_{z,k}^- \\
 &= \mathbf{H}_{v,k} \mathbf{P}_{vv,k}^- \mathbf{H}_{v,k}^T (\mathbf{P}_{zz,k}^-)^{-1} \mathbf{e}_{z,k}^-
 \end{aligned}$$

This result allows us to establish another relationship for the residual covariance; namely,

$$\mathbf{P}_{zz,k}^+ = [\mathbf{H}_{v,k} \mathbf{P}_{vv,k}^- \mathbf{H}_{v,k}^T] (\mathbf{P}_{zz,k}^-)^{-1} [\mathbf{H}_{v,k} \mathbf{P}_{vv,k}^- \mathbf{H}_{v,k}^T]^T$$

We are now ready for the residual property that we want to establish.

Consider the squared Mahalanobis distance using the residual and its covariance as

$$(\mathbf{e}_{z,k}^+)^T (\mathbf{P}_{zz,k}^+)^{-1} (\mathbf{e}_{z,k}^+)$$

If we substitute for our expressions for the residual and its covariance in terms of the innovation and its covariance, it follows that

$$(\mathbf{e}_{z,k}^+)^T (\mathbf{P}_{zz,k}^+)^{-1} (\mathbf{e}_{z,k}^+) = (\mathbf{e}_{z,k}^-)^T (\mathbf{P}_{zz,k}^-)^{-1} (\mathbf{e}_{z,k}^-)$$

provided that $[\mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T]$ is invertible.

What we have found is that the posterior squared Mahalanobis distance is exactly equal to the prior squared Mahalanobis distance! This can be used as a consistency check when running a Kalman filter, but it is important to keep in mind that this result is derived under the assumption of linear measurements and utilization of the Kalman gain.

3.1.4 Singular Measurement Noise

An interesting situation arises when the measurement noise is *so* small that the measurement noise covariance is numerically zero (or very close to it).

Theoretically, this cannot occur since the measurement noise covariance is a symmetric, positive definite matrix, but numerically it occurs when you have very precise measurements.

At first, it would appear that $\mathbf{P}_{vv,k} = \mathbf{0}_{m \times m}$ causes no problems since $\mathbf{P}_{vv,k}^{-1}$ does not appear in the Kalman filter, but let's dig a bit deeper.

Consider the covariance update given by

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{K}_k \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^-$$

which is the “textbook” form of the covariance update that relies on the optimal gain and linear measurements.

When the measurement noise covariance matrix is zero, the Kalman gain becomes

$$\mathbf{K}_k = \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T [\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T]^{-1}$$

and the covariance update can be written as

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T [\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T]^{-1} \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^-$$

Now, consider the posterior projection of the state covariance into the measurement domain, i.e.,

$$\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^+ \mathbf{H}_{x,k}^T$$

Substituting for the form of the covariance update in this situation, it follows that

$$\begin{aligned} \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^+ \mathbf{H}_{x,k}^T &= \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T - \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T [\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T]^{-1} \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T \\ &= \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T - \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T \\ &= \mathbf{0}_{m \times m} \end{aligned}$$

We have arrived at a very intriguing outcome. The posterior covariance *must be* singular! What, if any, are the ramifications of this singularity?

The covariance at the next time step is propagated according to

$$\mathbf{P}_{xx,k+1}^- = \Phi(t_{k+1}, t_k) \mathbf{P}_{xx,k}^+ \Phi^T(t_{k+1}, t_k) + \mathbf{P}_{ww}(t_{k+1})$$

If the time steps are close together, then

$$\Phi(t_{k+1}, t_k) \approx \mathbf{I}_n \quad \text{and} \quad \mathbf{P}_{ww}(t_{k+1}) \approx \mathbf{0}_{n \times n}$$

This means that the covariance update may begin to fail. That is, the innovation covariance may come out to be numerically close to zero, and the Kalman gain cannot be computed.

Ultimately, it means that we have to be very careful when we have precise measurements. We'll revisit this subject later on, but it is good to always keep this in mind.

We have an algorithm for the Kalman filter; however, that does not mean that it solves every problem or that numerical issues cannot degrade the performance of our filter.

3.2 The Extended Kalman Filter

The Kalman filter operates on linear dynamical/observational systems, but oftentimes we must deal with nonlinear dynamics, nonlinear measurements, or both. For instance, an object under the influence of two-body dynamics obeys a nonlinear differential equa-

tion in Cartesian coordinates. We often cannot observe the actual state of the object either. Instead, we tend to observe some nonlinear function of the state, such as range. We therefore want to modify our Kalman filter to be able to handle these nonlinearities. The extended Kalman filter (EKF) handles nonlinearities through the use of linearization.

As with the Kalman filter, the filter is comprised of two stages: propagation and update. We will proceed to develop the EKF in the same fashion as the Kalman filter, by first developing the evolutionary equations for the propagation of the mean and covariance and then developing update relationships for the mean and covariance.

Consider the nonlinear dynamical system subjected to random excitations

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{w}(t))$$

where

$$\text{E}\{\mathbf{w}(t)\} = \mathbf{0}_w \quad \text{and} \quad \text{E}\{\mathbf{w}(t)\mathbf{w}^T(\tau)\} = \mathbf{Q}_{ww}(t)\delta(t - \tau)$$

This model of the dynamics is quite a bit different from the one that we used to develop the Kalman filter. The dynamics are, in general, a nonlinear function of both the state

and the process noise. In some cases, you might come across a variation of this model given by

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t)) + \boldsymbol{F}_w(t)\boldsymbol{w}(t)$$

which is referred to as the “additive noise model.” This model retains the nonlinearity due to the state, but embeds a linear mapping of the process noise. Moving forward, we will make use of the fully nonlinear model.

We start by considering the mean of the state as a function of time, which is, by definition,

$$\boldsymbol{m}_x(t) = \text{E}\{\boldsymbol{x}(t)\}$$

Taking the time rate of change and interchanging the order of differentiation and expectation yields

$$\dot{\boldsymbol{m}}_x(t) = \text{E}\{\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{w}(t))\}$$

In general, we need to compute the expected value of the nonlinear function, which is challenging. The EKF approaches this challenge by linearizing the nonlinear dynamics

function. As such, consider a first-order Taylor series expansion (FOTSE) of $\mathbf{f}(\cdot, \cdot)$ about the expected value of the state and the expected value of the process noise, or

$$\mathbf{f}(\mathbf{x}(t), \mathbf{w}(t)) \approx \mathbf{f}(\mathbf{m}_x(t), \mathbf{0}_w) + \mathbf{F}_x(\mathbf{m}_x(t))(\mathbf{x}(t) - \mathbf{m}_x(t)) + \mathbf{F}_w(\mathbf{m}_x(t))\mathbf{w}(t)$$

where the dynamics Jacobians, $\mathbf{F}_x(\cdot)$ and $\mathbf{F}_w(\cdot)$, are defined as

$$\mathbf{F}_a(\mathbf{m}_x(t)) = \left[\frac{\partial \mathbf{f}(\mathbf{x}(t), \mathbf{w}(t))}{\partial \mathbf{a}(t)} \middle|_{\substack{\mathbf{x}(t)=\mathbf{m}_x(t) \\ \mathbf{w}(t)=\mathbf{0}_w}} \right]$$

for \mathbf{a} representing either \mathbf{x} or \mathbf{w} . This FOTSE can now be substituted into the dynamics of the mean (at which point the solution is approximate) to yield

$$\dot{\mathbf{m}}_x(t) = \text{E} \{ \mathbf{f}(\mathbf{m}_x(t), \mathbf{0}_w) + \mathbf{F}_x(\mathbf{m}_x(t))\mathbf{e}_x(t) + \mathbf{F}_w(\mathbf{m}_x(t))\mathbf{w}(t) \}$$

where $\mathbf{e}_x(t) = \mathbf{x}(t) - \mathbf{m}_x(t)$ is the estimation error. The expected value can be applied to each term individually, and assuming that $\mathbf{f}(\mathbf{m}_x(t), \mathbf{0}_w)$, $\mathbf{F}_x(\mathbf{m}_x(t))$, and $\mathbf{F}_w(\mathbf{m}_x(t))$ are deterministic, it follows that

$$\dot{\mathbf{m}}_x(t) = \mathbf{f}(\mathbf{m}_x(t), \mathbf{0}_w) + \mathbf{F}_x(\mathbf{m}_x(t))\text{E} \{ \mathbf{e}_x(t) \} + \mathbf{F}_w(\mathbf{m}_x(t))\text{E} \{ \mathbf{w}(t) \}$$

This assumption is complicit with the assumption that $\mathbf{m}_x(t)$ is deterministic.

Recalling that the process noise is taken to be zero-mean and assuming that the estimate is unbiased (equivalently, assuming that $\mathbf{e}_x(t)$ is a zero-mean process), it follows that the mean satisfies the differential equation

$$\dot{\mathbf{m}}_x(t) = \mathbf{f}(\mathbf{m}_x(t), \mathbf{0}_w)$$

This is the differential equation governing the forward evolution of the mean.

We now turn to developing a method for propagating the covariance. From the definition of the error, the error dynamics are

$$\begin{aligned}\dot{\mathbf{e}}_x(t) &= \dot{\mathbf{x}}(t) - \dot{\mathbf{m}}_x(t) \\ &= \mathbf{f}(\mathbf{x}(t), \mathbf{w}(t)) - \mathbf{f}(\mathbf{m}_x(t), \mathbf{0}_w)\end{aligned}$$

Substituting the FOTSE of the dynamics function that we have from earlier into the error dynamics and simplifying yields

$$\dot{\mathbf{e}}_x(t) = \mathbf{F}_x(\mathbf{m}_x(t))\mathbf{e}_x(t) + \mathbf{F}_w(\mathbf{m}_x(t))\mathbf{w}(t)$$

This error equation is very similar to the one that we used in the development of the Kalman filter, except it is key to note that \mathbf{F}_x and \mathbf{F}_w are not just dependent on time, as they were for the Kalman filter development; they have potential dependencies on the estimated state of the system, as well.

The solution of the linear differential equation for the error is

$$\mathbf{e}_x(t) = \Phi(t, t_{k-1})\mathbf{e}_x(t_{k-1}) + \int_{t_{k-1}}^t \Phi(t, \tau)\mathbf{F}_w(\mathbf{m}_x(\tau))\mathbf{w}(\tau)d\tau$$

where $\Phi(t, t_{k-1})$ is the state transition matrix, which obeys the differential equation

$$\dot{\Phi}(t, t_{k-1}) = \mathbf{F}_x(\mathbf{m}_x(t))\Phi(t, t_{k-1}), \quad \Phi(t_{k-1}, t_{k-1}) = \mathbf{I}_n$$

The dependence of \mathbf{F}_x on \mathbf{m}_x becomes important here. Since the differential equation of the state transition matrix depends on the mean of the state, the differential equation for the state transition matrix is coupled to the differential equation of the estimated state. This means that these two differential equations must be integrated together.

Leveraging the definition of the state estimation error covariance as

$$\mathbf{P}_{xx}(t) = \text{E} \left\{ \mathbf{e}_x(t) \mathbf{e}_x^T(t) \right\}$$

we can “simply” follow the same procedure that we developed for the Kalman filter to find propagation equations for the covariance, except that we follow that process with $\mathbf{F}_x(\mathbf{m}_x(t))$ in place of $\mathbf{F}_x(t)$ and $\mathbf{F}_w(\mathbf{m}_x(t))$ in place of $\mathbf{F}_w(t)$. This leads us, as with the Kalman filter, to two separate methods for propagating the covariance.

First method for covariance propagation:

- Propagate state transition matrix (s.t.i.c. $\Phi(t_{k-1}, t_{k-1}) = \mathbf{I}_n$)

$$\dot{\Phi}(t, t_{k-1}) = \mathbf{F}_x(\mathbf{m}_x(t))\Phi(t, t_{k-1})$$
- Propagate process noise covariance matrix (s.t.i.c. $\mathbf{P}_{ww}(t_{k-1}) = \mathbf{0}_{n \times n}$)

$$\dot{\mathbf{P}}_{ww}(t) = \mathbf{F}_x(\mathbf{m}_x(t))\mathbf{P}_{ww}(t) + \mathbf{P}_{ww}(t)\mathbf{F}_x^T(\mathbf{m}_x(t)) + \mathbf{F}_w(\mathbf{m}_x(t))\mathbf{Q}_{ww}(t)\mathbf{F}_w^T(\mathbf{m}_x(t))$$
- Calculate the propagated covariance matrix

$$\mathbf{P}_{xx}(t) = \Phi(t, t_{k-1})\mathbf{P}_{xx}(t_{k-1})\Phi^T(t, t_{k-1}) + \mathbf{P}_{ww}(t)$$

Second method for covariance propagation:

- Propagate the covariance matrix (s.t.i.c. $\mathbf{P}_{xx}(t_{k-1}) = \mathbf{P}_{xx,k-1}$)

$$\dot{\mathbf{P}}_{xx}(t) = \mathbf{F}_x(\mathbf{m}_x(t))\mathbf{P}_{xx}(t) + \mathbf{P}_{xx}(t)\mathbf{F}_x^T(\mathbf{m}_x(t)) + \mathbf{F}_w(\mathbf{m}_x(t))\mathbf{Q}_{ww}(t)\mathbf{F}_w^T(\mathbf{m}_x(t))$$

For either option, we begin the propagation stage with the initial conditions

$$\mathbf{m}_x(t_{k-1}) = \mathbf{m}_{x,k-1}^+ \quad \text{and} \quad \mathbf{P}_{xx}(t_{k-1}) = \mathbf{P}_{xx,k-1}^+$$

which are simply the *a posteriori* estimate and covariance after the inclusion of any available newly acquired data at time t_{k-1} .

The values obtained after propagation (numerical integration and/or covariance mapping) become our *a priori* mean and covariance, $\mathbf{m}_{x,k}^-$ and $\mathbf{P}_{xx,k}^-$, when we encounter new measurement data.

At time t_k a measurement is made available, which is given by \mathbf{z}_k . This measurement is, in general, a nonlinear function of the state and is imperfect (noisy). This measurement is taken to be of the form

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k)$$

where

$$\mathrm{E}\{\boldsymbol{v}_k\} = \mathbf{0}_v \quad \text{and} \quad \mathrm{E}\{\boldsymbol{v}_k \boldsymbol{v}_\ell^T\} = \boldsymbol{P}_{vv,k} \delta_{k\ell}$$

The measurement noise is represented by \boldsymbol{v}_k , which is assumed to be a zero mean white-noise sequence with covariance $\boldsymbol{P}_{vv,k}$.

As with the dynamics model, is not uncommon to come across an additive noise model of the form

$$\boldsymbol{z}_k = \boldsymbol{h}(\boldsymbol{x}_k) + \boldsymbol{H}_{v,k} \boldsymbol{v}_k$$

which retains nonlinearity of the measurement with respect to the state. As this model is encapsulated by the fully nonlinear model, we will make use of the fully nonlinear model moving forward.

When we developed the update equations for the Kalman filter, we did so in a fairly generic manner. As such, we can leverage those results here, such that the starting point

for the update equations of the extended Kalman filter are

$$\mathbf{m}_k^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{m}_{z,k}^-)$$

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- \mathbf{K}_k^T - \mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

In arriving at these expressions, we are using the first two moments, employing a linear update law, and constructing an unbiased estimator. We are not assuming the gain is the Kalman gain, and we are not making any linear assumptions on the measurement process. We do, however, need to be able to compute the expectations

$$\mathbf{m}_{z,k}^- = \text{E} \{ \mathbf{z}_k \}$$

$$\mathbf{P}_{xz,k}^- = \text{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \right\}$$

$$\mathbf{P}_{zz,k}^- = \text{E} \left\{ (\mathbf{z}_k - \mathbf{m}_{z,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \right\}$$

to complete the update process. And, if we have these values and wish to use the Kalman gain, we can find that gain as

$$\mathbf{K}_k = \mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1}$$

The difference between a linear system and a nonlinear system is in how we compute the expected values.

For the EKF, linearization via the FOTSE is performed. For instance, the FOTSE of the nonlinear measurement model produces

$$\mathbf{h}(\mathbf{x}_k, \mathbf{v}_k) \approx \mathbf{h}(\mathbf{m}_{x,k}^-, \mathbf{0}_v) + \mathbf{H}_x(\mathbf{m}_{x,k}^-)(\mathbf{x}_k - \mathbf{m}_{x,k}^-) + \mathbf{H}_v(\mathbf{m}_{x,k}^-)\mathbf{v}_k$$

where the measurement Jacobians, $\mathbf{H}_x(\cdot)$ and $\mathbf{H}_v(\cdot)$, are defined as

$$\mathbf{H}_a(\mathbf{m}_{x,k}^-) = \left[\frac{\partial \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k)}{\partial \mathbf{a}_k} \middle|_{\begin{array}{l} \mathbf{x}_k = \mathbf{m}_{x,k}^- \\ \mathbf{v}_k = \mathbf{0}_v \end{array}} \right]$$

for \mathbf{a} representing either \mathbf{x} or \mathbf{v} . This FOTSE can now be used to simplify the expectation calculations, but it is important to recognize that the results will be approximations of the true expected values.

Taking the expected value of the FOTSE yields

$$\mathbf{m}_{z,k}^- = \mathbb{E}\{\mathbf{h}(\mathbf{m}_{x,k}^-, \mathbf{0}_v) + \mathbf{H}_x(\mathbf{m}_{x,k}^-)\mathbf{e}_{x,k}^- + \mathbf{H}_v(\mathbf{m}_{x,k}^-)\mathbf{v}_k\}$$

where $\mathbf{e}_{x,k}^- = \mathbf{x}_k - \mathbf{m}_{x,k}^-$ is the *a priori* estimation error. Distributing the expected value to each term and assuming that $\mathbf{h}(\cdot, \cdot)$, $\mathbf{H}_x(\cdot)$, and $\mathbf{H}_v(\cdot)$ are deterministic, it follows that

$$\mathbf{m}_{z,k}^- = \mathbf{h}(\mathbf{m}_{x,k}^-, \mathbf{0}_v) + \mathbf{H}_x(\mathbf{m}_{x,k}^-) \mathbb{E}\{\mathbf{e}_{x,k}^-\} + \mathbf{H}_v(\mathbf{m}_{x,k}^-) \mathbb{E}\{\mathbf{v}_k\}$$

Recalling that our desired estimator is unbiased and that the measurement noise is zero-mean, it follows that the EKF approximation of the expected measurement is

$$\mathbf{m}_{z,k}^- = \mathbf{h}(\mathbf{m}_{x,k}^-, \mathbf{0}_v)$$

We also need to develop expressions for the cross-covariance and innovation covariance. To do so, let's first consider the innovation for the EKF, given by

$$\mathbf{z}_k - \mathbf{m}_{z,k}^- = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k) - \mathbf{h}(\mathbf{m}_{x,k}^-, \mathbf{0}_v)$$

and apply the FOTSE of the nonlinear measurement function to yield

$$\mathbf{z}_k - \mathbf{m}_{z,k}^- = \mathbf{H}_x(\mathbf{m}_{x,k}^-)(\mathbf{x}_k - \mathbf{m}_{x,k}^-) + \mathbf{H}_v(\mathbf{m}_{x,k}^-)\mathbf{v}_k$$

Now, we consider the cross-covariance, which is defined as

$$\mathbf{P}_{xz,k}^- = \mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \right\}$$

From the reduction of the innovation via the FOTSE, it follows that

$$\mathbf{P}_{xz,k}^- = \mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{x}_k - \mathbf{m}_{x,k}^-)^T \right\} \mathbf{H}_x^T(\mathbf{m}_{x,k}^-) + \mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)\mathbf{v}_k^T \right\} \mathbf{H}_v^T(\mathbf{m}_{x,k}^-)$$

which follows from the assumption that $\mathbf{H}_x(\cdot)$ and $\mathbf{H}_v(\cdot)$ are deterministic.

The first expectation in this expression is clearly the state estimation error covariance matrix; the second expectation describes the correlations between the state and the measurement noise. Much like with the Kalman filter, we now assume that the state is not correlated to the measurement noise, i.e.,

$$\mathbb{E} \left\{ (\mathbf{x}_k - \mathbf{m}_{x,k}^-)\mathbf{v}_k^T \right\} = \mathbf{0}_{n \times v}$$

and it follows that the cross-covariance for nonlinear measurements is

$$\mathbf{P}_{xz,k}^- = \mathbf{P}_{xx,k}^- \mathbf{H}_x^T(\mathbf{m}_{x,k}^-)$$

To complete the expectations for the EKF, we need to consider the innovation covariance matrix, or

$$\mathbf{P}_{zz,k}^- = \mathbb{E} \left\{ (\mathbf{z}_k - \mathbf{m}_{z,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T \right\}$$

Making use of the previously developed result that

$$\mathbf{z}_k - \mathbf{m}_{z,k}^- = \mathbf{H}_x(\mathbf{m}_{x,k}^-)(\mathbf{x}_k - \mathbf{m}_{x,k}^-) + \mathbf{H}_v(\mathbf{m}_{x,k}^-)\mathbf{v}_k$$

and recalling the previous properties/assumptions that

- $\mathbf{H}_x(\cdot)$ and $\mathbf{H}_v(\cdot)$ are deterministic
- the state is not correlated with the measurement noise
- the covariance of the measurement noise is given by $\mathbf{P}_{vv,k}$

gives the innovation covariance for nonlinear measurements as

$$\mathbf{P}_{zz,k}^- = \mathbf{H}_x(\mathbf{m}_{x,k}^-)\mathbf{P}_{xx,k}^-\mathbf{H}_x^T(\mathbf{m}_{x,k}^-) + \mathbf{H}_v(\mathbf{m}_{x,k}^-)\mathbf{P}_{vv,k}\mathbf{H}_v^T(\mathbf{m}_{x,k}^-)$$

There is clear similarity of the expectation results to those found in the Kalman filter. In fact, if one makes the linear measurement assumption, the Jacobians for the nonlinear model become the mapping matrices of the linear model, and the same results we found for the linear model are recovered by the EKF. It is, however, important to keep in mind that the EKF formulation of expectations is approximate.

To summarize, we put everything together in a single table

System Model	$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{w}(t))$
Meas. Model	$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k)$
Init. Cond.	$\mathbf{m}_{x,0} = \text{E}\{\mathbf{x}(t_0)\}$ $\mathbf{P}_{xx,0} = \text{E}\{(\mathbf{x}(t_0) - \mathbf{m}_{x,0})(\mathbf{x}(t_0) - \mathbf{m}_{x,0})^T\}$
Mean Prop.	$\dot{\mathbf{m}}_x(t) = \mathbf{f}(\mathbf{m}_x(t), \mathbf{0}_w)$
Cov. Prop.	$\dot{\mathbf{P}}_{xx}(t) = \mathbf{F}_x(\mathbf{m}_x(t))\mathbf{P}_{xx}(t) + \mathbf{P}_{xx}(t)\mathbf{F}_x^T(\mathbf{m}_x(t))$ $+ \mathbf{F}_w(\mathbf{m}_x(t))\mathbf{Q}_{ww}(t)\mathbf{F}_w^T(\mathbf{m}_x(t))$
Exp. Meas.	$\mathbf{m}_{z,k}^- = \mathbf{h}(\mathbf{m}_{x,k}^-, \mathbf{0}_v)$
Cross Cov.	$\mathbf{P}_{xz,k}^- = \mathbf{P}_{xx,k}^-\mathbf{H}_x^T(\mathbf{m}_{x,k}^-)$
Innov. Cov.	$\mathbf{P}_{zz,k}^- = \mathbf{H}_x(\mathbf{m}_{x,k}^-)\mathbf{P}_{xx,k}^-\mathbf{H}_x^T(\mathbf{m}_{x,k}^-) + \mathbf{H}_v(\mathbf{m}_{x,k}^-)\mathbf{P}_{vv,k}\mathbf{H}_v^T(\mathbf{m}_{x,k}^-)$
Kalman Gain	$\mathbf{K}_k = \mathbf{P}_{xz,k}^-(\mathbf{P}_{zz,k}^-)^{-1}$
Mean Upd.	$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k[\mathbf{z}_k - \mathbf{m}_{z,k}^-]$
Cov. Upd.	$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^-\mathbf{K}_k^T - \mathbf{K}_k(\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k\mathbf{P}_{zz,k}^-\mathbf{K}_k^T$

3.2.1 Example: Falling Body

Here we will look at a classic example of the EKF as presented by Gelb (1974).

Consider the problem of tracking a body falling freely through the atmosphere. The motion is modeled in one dimension by assuming the body falls in a straight line, directly toward a tracking radar. A radar return is received every 0.1 [sec].

The state is defined as

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x \\ \dot{x} \\ \beta \end{bmatrix}$$

where x is the height of the falling body above the earth and β is the ballistic coefficient of the object. The equations of motion for the body are given by

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ d - g \\ 0 \end{bmatrix} = \mathbf{f}(\mathbf{x})$$

with

$$d = \frac{\rho x_2^2}{2x_3} \quad \text{and} \quad \rho = \rho_0 \exp\left\{-\frac{x_1}{k_\rho}\right\}$$

where d is drag acceleration, g is the gravitational acceleration, ρ is atmospheric density (with ρ_0 as the atmospheric density at sea level), and k_ρ is a decay constant. The differential equation governing x_2 (velocity) is nonlinear through the dependence of drag on velocity, air density, and ballistic coefficient.

The range sensor is modeled to generate measurements of the form

$$z_k = x_{1,k} + v_k$$

where v_k is zero mean with constant (co)variance P_{vv} . For simulation purposes, we take the measurement noise to be Gaussian distributed, such that $v_k \sim p_g(0, P_{vv})$.

The initial truth for the simulation is drawn according to the Gaussian distributions

$$x_0 = p_g(10^5 \text{ ft}, 500 \text{ ft}^2)$$

$$\dot{x}_0 = p_g(-6000 \text{ ft/sec}, 2 \times 10^4 \text{ ft}^2/\text{sec}^2)$$

$$\beta = p_g(2000 \text{ lb/ft}^2, 2.5 \times 10^5 \text{ lb}^2/\text{ft}^4)$$

such that the initial mean and covariance are taken to be

$$\mathbf{m}_{x,0} = \begin{bmatrix} 10^5 \text{ ft} \\ -6000 \text{ ft/sec} \\ 2000 \text{ lb/ft}^2 \end{bmatrix}$$

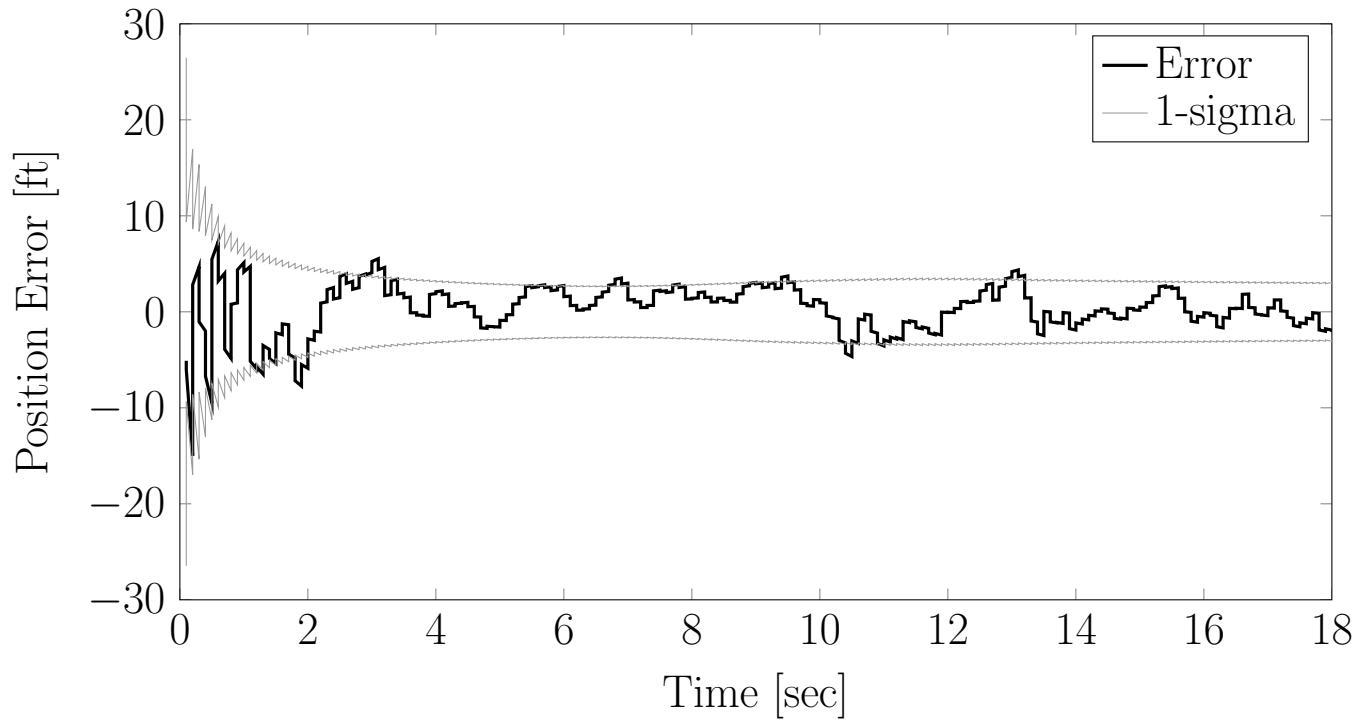
$$\mathbf{P}_{xx,0} = \begin{bmatrix} 500 \text{ ft}^2 & 0 & 0 \\ 0 & 2 \times 10^4 \text{ ft}^2/\text{sec}^2 & 0 \\ 0 & 0 & 2.5 \times 10^5 \text{ lb}^2/\text{ft}^4 \end{bmatrix}$$

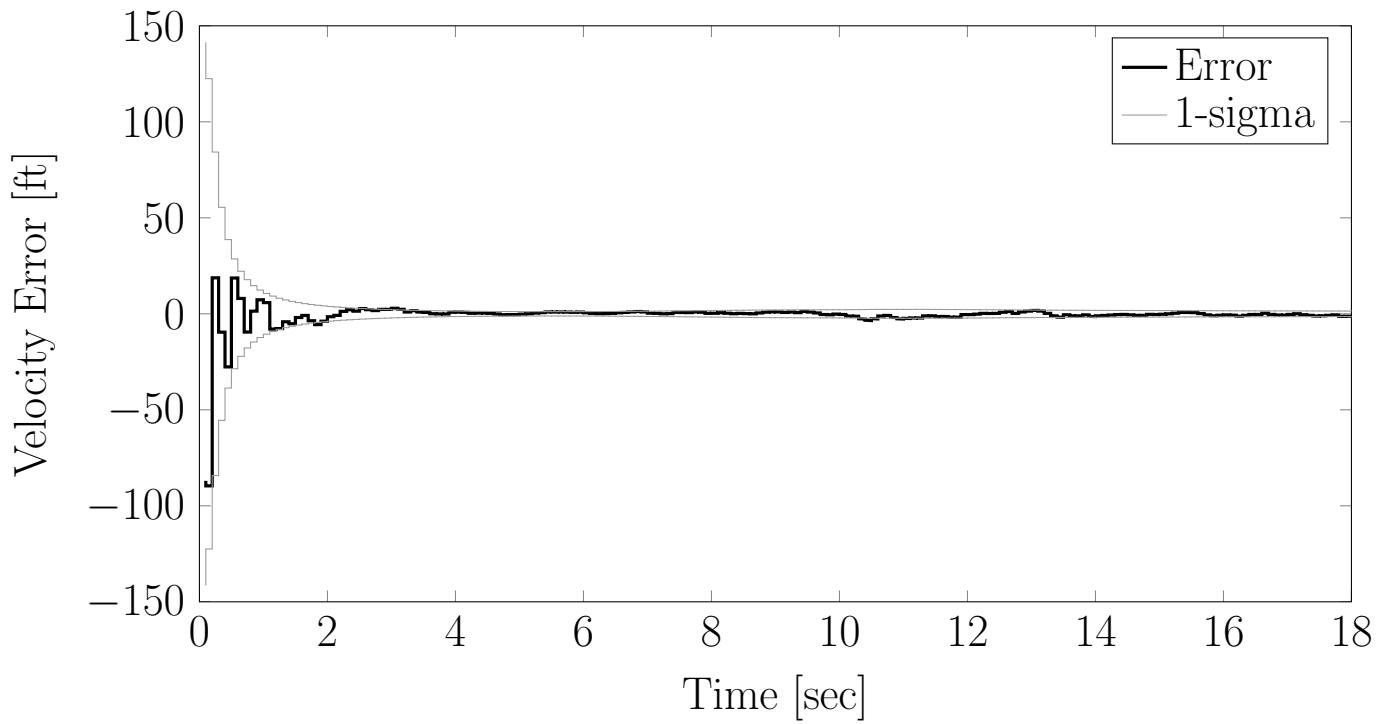
The system parameters are taken to be

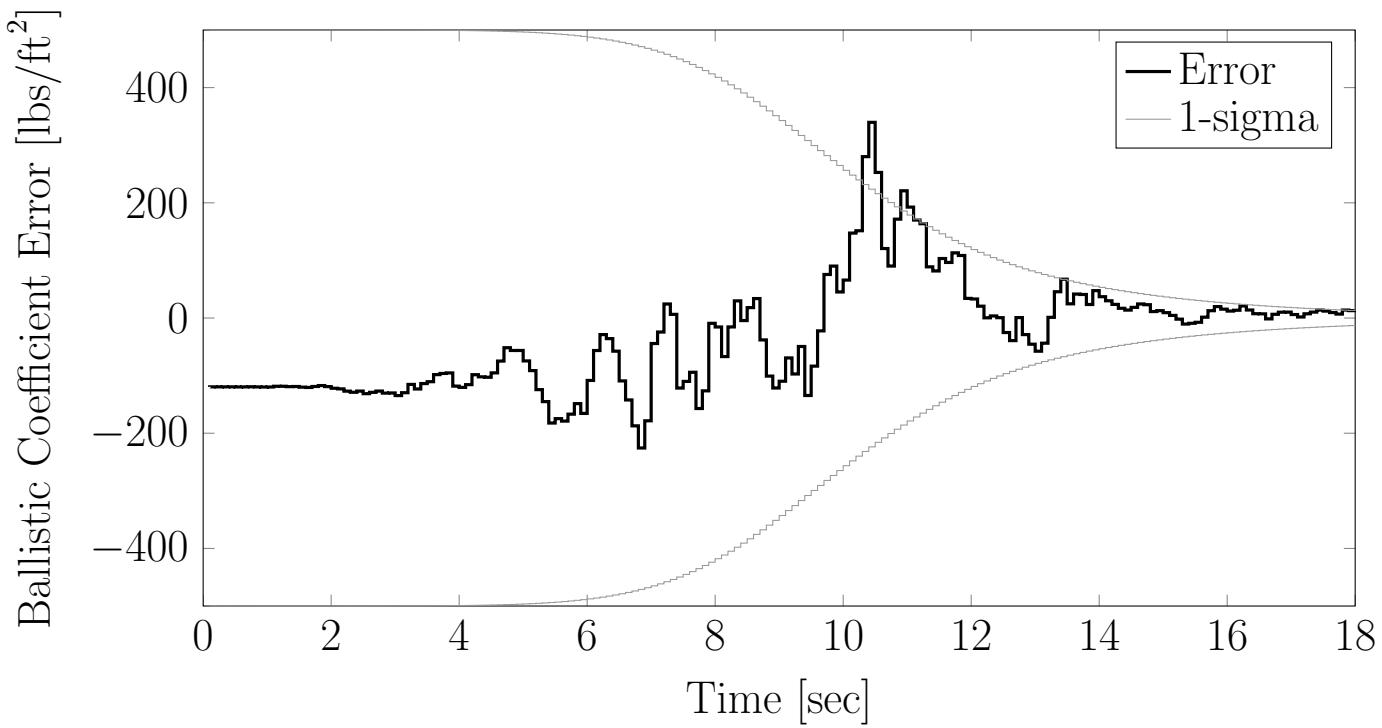
$$\rho_0 = 3.4 \times 10^{-3} \text{ lb sec}^2/\text{ft}^4 \quad g = 32.2 \text{ ft/sec}^2$$

$$k_\rho = 22000 \text{ ft} \quad P_{vv} = 100 \text{ ft}^2.$$

The resulting error for each state variable and their associated 1σ intervals (from the square root of the corresponding entry in the covariance matrix) can be seen below.







For the most part, we see good estimation error performance, in comparison to the 1σ intervals. As expected, there are brief periods of time when the estimation error of any particular state goes beyond the $\pm 1\sigma$ interval.

For both the position and velocity states, we notice fairly rapid “convergence” of the uncertainty interval as measurements are processed. This is fairly intuitive, as we are processing linear measurements of the position state. As we process a time sequence of these measurements, we also are able to estimate the velocity of the object.

On the other hand, if we direct our attention to the errors in estimating the ballistic coefficient, we note that the EKF does not track β accurately in the early stages of measurement processing. Physically, this is due to the fact that the thin atmosphere at high altitude produces a small drag force on the body. The thicker atmosphere, creating an increased drag force, enables the EKF to achieve lower estimation error for β .

In this problem, we can only estimate the ballistic coefficient by correlating it to the states that we can observe, i.e., there need to be position-dependent effects in order to estimate the ballistic coefficient using measurements of the position. This accumulation of correlation occurs through the atmospheric density. It takes some time for the

correlation to build, and the correlation builds more rapidly when the effects are stronger.

We will now walk through the steps involved in coding this extended Kalman filter in MATLAB. There are a few things to note before we do so. First of all, the general structure that we have employed in coding the EKF mirrors to a great deal the structure we used in our coding example of the Kalman filter. Secondly, since the measurements are linear for this problem, we use the linear relationships for the measurement update quantities. This would need to be adjusted for nonlinear measurements.

Define a random number seed, integrator options, and timing parameters:

```
% Set the random number seed
rng(100);

% Integrator options
opts = odeset('AbsTol',1e-6,'RelTol',1e-6);

% Simulation timing information
dt = 0.1;
tv = 0.0:dt:10.0;
```

Specify initial conditions for the filter, and generate a random true initial state:

```
% Initial estimate and random truth
mx0 = [1e4; -2000.0; 2000.0];
Pxx0 = diag([500.0, 2e4, 2.5e5]);
x0 = mx0 + chol(Pxx0)'*randn(3,1);
```

The dynamics and measurements models get specified next:

```
% Dynamics model parameters
rho0 = 3.4e-3;
g     = 32.2;
krho = 22000.0;

% Measurement model
Hx   = [1.0, 0.0, 0.0];
Hv   = 1.0;
Pvv = 100.0;
```

We set up storage space and counters to help us store results for analysis:

```
% Initialize storage space for analysis
xcount = 0;
txstore = zeros(1,2*length(tv)-1);
exstore = zeros(3,2*length(tv)-1);
sxstore = zeros(3,2*length(tv)-1);
zcount = 0;
tzstore = zeros(1,length(tv)-1);
ezstore = zeros(1,length(tv)-1);
szstore = zeros(1,length(tv)-1);
```

The first thing we store is information about the initial truth and estimate:

```
% Store initial data
xcount           = xcount + 1;
txstore(:,xcount) = tv(1);
exstore(:,xcount) = x0 - mx0;
sxstore(:,xcount) = sqrt(diag(Pxx0));
```

We now initialize variables to start running the filter, and loop over time:

```
% Initialize variables for filter
xkm1 = x0;
mxkm1 = mx0;
Pxxkm1 = Pxx0;

% Loop over time vector
for i = 2:length(tv)
```

Propagate the truth from t_{k-1} to t_k :

```
% Propagate truth
[~,X] = ode45(@one_dim_eoms,[tv(i-1),tv(i)],xkm1,opts,g,rho0,krho);
xk = X(end,1:3);
```

Generate a noisy measurement at t_k :

```
% Generate measurement
zk = Hx*xk + chol(Pvv) '*randn(1,1);
```

Predict according to the extended Kalman filter equations:

```
% Propagate step of extended Kalman filter
[~,X] = ode45(@one_dim_eoms,[tv(i-1),tv(i)], [mxkm1;Pxxkm1(:)],opts,g,rho0,krho);
mxkm = X(end,1:3)';
Pxxkm = reshape(X(end,4:end)',3,3);
```

Store the *a priori* mean and standard deviations to look at later:

```
% Store a priori mean and covariance
xcount = xcount + 1;
txstore(:,xcount) = tv(i);
exstore(:,xcount) = xk - mxkm;
sxstore(:,xcount) = sqrt(diag(Pxxkm));
```

Update according to the Kalman filter equations:

```
% Update step of extended Kalman filter (linear meas.)  
mzkm = Hx*mxkm;  
Pxzkm = Pxxkm*Hx';  
Pzzkm = Hx*Pxxkm*Hx' + Hv*Pvv*Hv';  
Kk = Pxzkm/Pzzkm;  
mxkp = mxkm + Kk*(zk - mzkm);  
Pxxkp = Pxxkm - Pxzkm*Kk' - Kk*Pxzkm' + Kk*Pzzkm*Kk';
```

Store the *a posteriori* mean and standard deviations to look at later:

```
% Store a posteriori mean and stdevs  
xcount = xcount + 1;  
txstore(:,xcount) = tv(i);  
exstore(:,xcount) = xk - mxkp;  
sxstore(:,xcount) = sqrt(diag(Pxxkp));
```

Also store predicted measurement and standard deviations to look at later:

```
% Store data, estimate, and stdev
zcount = zcount + 1;
tzstore(:,zcount) = tv(i);
ezstore(:,zcount) = zk - mzkm;
szstore(:,zcount) = sqrt(diag(Pzzkm));
```

Set up the recursion for the next time step (and don't forget to close the loop):

```
% Cycle for the next time step
xkm1 = xk;
mxkm1 = mxkp;
Pxxkm1 = Pxxkp;
end
```

At this point, we have completed the loop of propagating between measurements and processing the acquired measurement data. In doing so, we have stored the true state of the system, our estimated mean and covariance (via standard deviations). We have also stored the true data that we simulated, our predictions of that data, and our innovation covariance (via standard deviations). This information gives us the ability to analyze

several aspects of the filter that we've applied to this problem.

The filter also relies on a file containing the equations of motion. For this problem, that file is:

```
function [dxdt] = one_dim_eoms(t,x,g,rho0,krho)

% Density
rho = rho0*exp(-x(1)/krho);

% Dynamics of each state
x1dot = x(2);
x2dot = rho*x(2)^2/(2.0*x(3)) - g;
x3dot = 0.0;

% Dynamics of all states
dxdt = [x1dot; x2dot; x3dot];
```

If we are only interested in propagating the truth or the estimated state, this is sufficient; however, we can also include the propagation of the covariance as

```

% If also propagating the covariance, continue
if length(x) > 3
    % Evaluate the dynamics Jacobian
    Fx = [0.0, 1.0, 0.0;
           -(rho*x(2)^2)/(2.0*krho*x(3)), ...
           (rho*x(2))/x(3), ...
           -(rho*x(2)^2)/(2.0*x(3)^2);
           0.0, 0.0, 0.0];

    % Extract covariance from the integration state
    Pxx = reshape(x(4:end), 3, 3);

    % Covariance dynamics (no process noise)
    Pxxdot = Fx*Pxx + Pxx*Fx';

    % Append output with covariance time derivative
    dxdt = [dxdt; Pxxdot(:)];
end

```

It is also possible (and oftentimes even desirable) to write these as two separate equations of motion files. For the case here, however, it is simple enough and easy enough to combine

them into a single function.

3.2.2 A Few Important Points

There is a very important and often overlooked difference between the Kalman filter and the extended Kalman filter.

The gain, \mathbf{K}_k , employed in the EKF is actually a random variable, whereas the gain in the Kalman filter is not. This stems from the fact that we have chosen to linearize our dynamics and our measurement about the current conditional mean. Once we update the mean and covariance with a single measurement, which has some random noise included in it, the mean becomes a random variable.

In the Kalman filter, this is equally true, but there is no linearization performed. In the EKF, linearizing about the mean yields Jacobian matrices that are functions of the mean. Since the mean is random, the Jacobians become random. Because of this, the Kalman gain becomes random.

It is important to note that we assumed that the Jacobians were deterministic, and

we did this several times:

- computing the covariance evolution (not explicitly shown)
- computing the expected measurement
- computing the cross-covariance
- computing the residual covariance
- computing the posterior covariance (not explicitly shown)
- computing the Kalman gain

Additionally, the covariance becomes random as well. This implies that the EKF is trajectory-dependent, meaning that its inherent accuracy and precision are functionally dependent upon the trajectory and the sequence of measurements employed. The Kalman filter, by contrast, is not trajectory dependent.

The Kalman gain can be computed off-line, and the covariance is not a function of the trajectory. The estimated state in the Kalman filter will still change based on the

processed data, but these effects do not make their way into the covariance matrix.

For the EKF, all calculations must be performed on-line.

3.3 The Discrete Kalman Filter

We have now developed the Kalman filter and the extended Kalman filter for a system model where the dynamics are in continuous-time (ordinary differential equations) but the measurements occur at discrete points in time. Not all systems fit into this set of models; while measurement data are very commonly taken at discrete points in times, there are systems that also have discrete-time dynamics. As such, we now turn our attention to a system that is modeled accorded to discrete-time dynamics and measurements, and we will developed the discrete Kalman filter to estimate the states of these types of system.

Consider a linear, discrete-time dynamical system of the form

$$\mathbf{x}_k = \mathbf{F}_{x,k-1} \mathbf{x}_{k-1} + \mathbf{F}_{w,k-1} \mathbf{w}_{k-1}$$

that is accompanied by linear, discrete-time measurements of the form

$$\mathbf{z}_k = \mathbf{H}_{x,k} \mathbf{x}_k + \mathbf{H}_{v,k} \mathbf{v}_k$$

Rather than a differential equation governing the evolution of the state of the system, we now have a difference equation. The measurement equation, on the other hand, is as what we have previously handled in the Kalman filter developments.

The state, \mathbf{x}_{k-1} , only occurs at discrete instances of time, and the initial mean and covariance of the state are given by $\mathbf{m}_{x,0}$ and $\mathbf{P}_{xx,0}$. The process noise, \mathbf{w}_{k-1} , is taken to be a zero-mean, white-noise sequence with covariance $\mathbf{P}_{ww,k-1}$. Similarly, the measurement noise, \mathbf{v}_k , is taken to be a zero-mean, white-noise sequence with covariance $\mathbf{P}_{vv,k}$.

The discrete Kalman filter, like other Kalman filters, is comprised of a propagation and update stages.

We will first consider the propagation stage, where we want to propagate the mean and covariance from t_{k-1} to t_k . To begin, we consider the propagation of the mean by

taking the expected value of our difference equation as

$$\begin{aligned}\mathrm{E}\{\boldsymbol{x}_k\} &= \mathrm{E}\{\boldsymbol{F}_{x,k-1}\boldsymbol{x}_{k-1} + \boldsymbol{F}_{w,k-1}\boldsymbol{w}_{k-1}\} \\ &= \mathrm{E}\{\boldsymbol{F}_{x,k-1}\boldsymbol{x}_{k-1}\} + \mathrm{E}\{\boldsymbol{F}_{w,k-1}\boldsymbol{w}_{k-1}\} \\ &= \boldsymbol{F}_{x,k-1}\mathrm{E}\{\boldsymbol{x}_{k-1}\} + \boldsymbol{F}_{w,k-1}\mathrm{E}\{\boldsymbol{w}_{k-1}\}\end{aligned}$$

In arriving at this result, we have assumed that $\boldsymbol{F}_{x,k-1}$ and $\boldsymbol{F}_{w,k-1}$ are deterministic. We can now make use of the fact that the process noise is zero mean, and it follows that the mean is propagated according to

$$\boldsymbol{m}_{x,k} = \boldsymbol{F}_{x,k-1}\boldsymbol{m}_{x,k-1}$$

To obtain a covariance propagation method, we start by defining the estimation errors at steps $k - 1$ and k to be

$$\boldsymbol{e}_{x,k-1} = \boldsymbol{x}_{k-1} - \boldsymbol{m}_{x,k-1} \quad \text{and} \quad \boldsymbol{e}_{x,k} = \boldsymbol{x}_k - \boldsymbol{m}_{x,k}$$

Starting from the estimation error at time k , we can find a difference equation for the estimation error at k as

$$\begin{aligned}\boldsymbol{e}_{x,k} &= \boldsymbol{x}_k - \boldsymbol{m}_{x,k} \\ &= [\boldsymbol{F}_{x,k-1}\boldsymbol{x}_{k-1} + \boldsymbol{F}_{w,k-1}\boldsymbol{w}_{k-1}] - [\boldsymbol{F}_{x,k-1}\boldsymbol{m}_{x,k-1}] \\ &= \boldsymbol{F}_{x,k-1}\boldsymbol{e}_{x,k-1} + \boldsymbol{F}_{w,k-1}\boldsymbol{w}_{k-1}\end{aligned}$$

The state estimation error covariance at time k is defined as

$$\boldsymbol{P}_{xx,k} = \text{E}\{\boldsymbol{e}_{x,k}\boldsymbol{e}_{x,k}^T\}$$

We can apply our error propagation equation to this definition to find

$$\begin{aligned}\boldsymbol{P}_{xx,k} &= \text{E}\left\{[\boldsymbol{F}_{x,k-1}\boldsymbol{e}_{x,k-1} + \boldsymbol{F}_{w,k-1}\boldsymbol{w}_{k-1}][\boldsymbol{F}_{x,k-1}\boldsymbol{e}_{x,k-1} + \boldsymbol{F}_{w,k-1}\boldsymbol{w}_{k-1}]^T\right\} \\ &= \boldsymbol{F}_{x,k-1}\text{E}\{\boldsymbol{e}_{x,k-1}\boldsymbol{e}_{x,k-1}^T\}\boldsymbol{F}_{x,k-1}^T + \boldsymbol{F}_{w,k-1}\text{E}\{\boldsymbol{w}_{k-1}\boldsymbol{e}_{x,k-1}^T\}\boldsymbol{F}_{x,k-1}^T \\ &\quad + \boldsymbol{F}_{x,k-1}\text{E}\{\boldsymbol{e}_{x,k-1}\boldsymbol{w}_{k-1}^T\}\boldsymbol{F}_{w,k-1}^T + \boldsymbol{F}_{w,k-1}\text{E}\{\boldsymbol{w}_{k-1}\boldsymbol{w}_{k-1}^T\}\boldsymbol{F}_{w,k-1}^T\end{aligned}$$

where we have used the fact that $\boldsymbol{F}_{x,k-1}$ and $\boldsymbol{F}_{w,k-1}$ are deterministic to pull them outside of the expectations.

The first expectation is the covariance at t_{k-1} , the final expectation is the process noise covariance, and the middle two expectations are the covariance of the state with the process noise. Taking the state to be uncorrelated with the process noise, it follows that

$$\mathbf{P}_{xx,k} = \mathbf{F}_{x,k-1} \mathbf{P}_{xx,k-1} \mathbf{F}_{x,k-1}^T + \mathbf{F}_{w,k-1} \mathbf{P}_{ww,k-1} \mathbf{F}_{w,k-1}^T$$

It is worth noting that this shares a remarkable similarity to the covariance propagation equation that makes use of the state transition matrix, which occurs because the state transition matrix effectively converts our continuous-time system into a discrete-time system. The inclusion of process noise, however, occurs in a slightly different manner. β

What about the measurement update? After we complete the propagation step, we have the prior mean and covariance, $\mathbf{m}_{x,k}^-$ and $\mathbf{P}_{xx,k}^-$, and we want to use the new measurement information to update our estimated state and our confidence in that estimate. For the discrete Kalman filter, which applies to linear measurements, this is a problem that we have already solved. This is exactly the problem considered in the update stage of the Kalman filter.

Since there's no point in reinventing the wheel, let's just use those results. Specifically,

our mean and covariance updates are given by

$$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{m}_{z,k}^-)$$

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^-\mathbf{K}_k^T - \mathbf{K}_k(\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k\mathbf{P}_{zz,k}^-\mathbf{K}_k^T$$

where $\mathbf{m}_{z,k}^-$ is the predicted measurement, $\mathbf{P}_{xz,k}^-$ is the cross-covariance (of the state with the measurement), $\mathbf{P}_{zz,k}^-$ is the innovation covariance, and \mathbf{K}_k is a linear gain, such as the Kalman gain.

From our treatment of linear measurements for the Kalman filter, we know that

$$\mathbf{m}_{z,k}^- = \mathbf{H}_{x,k}\mathbf{m}_{x,k}^-$$

$$\mathbf{P}_{xz,k}^- = \mathbf{P}_{xx,k}^-\mathbf{H}_{x,k}^T$$

$$\mathbf{P}_{zz,k}^- = \mathbf{H}_{x,k}\mathbf{P}_{xx,k}^-\mathbf{H}_{x,k}^T + \mathbf{H}_{v,k}\mathbf{P}_{vv,k}\mathbf{H}_{v,k}^T$$

and that the Kalman gain is found to minimize the mean square *a posteriori* error, which leads to

$$\mathbf{K}_k = \mathbf{P}_{xz,k}^-(\mathbf{P}_{zz,k}^-)^{-1}$$

To summarize, we put everything together in a single table

System Model	$\mathbf{x}_k = \mathbf{F}_{x,k-1}\mathbf{x}_{k-1} + \mathbf{F}_{w,k-1}\mathbf{w}_{k-1}$
Meas. Model	$\mathbf{z}_k = \mathbf{H}_{x,k}\mathbf{x}_k + \mathbf{H}_{v,k}\mathbf{v}_k$
Init. Cond.	$\mathbf{m}_{x,0} = \text{E}\{\mathbf{x}(t_0)\}$ $\mathbf{P}_{xx,0} = \text{E}\{(\mathbf{x}(t_0) - \mathbf{m}_{x,0})(\mathbf{x}(t_0) - \mathbf{m}_{x,0})^T\}$
Mean Prop.	$\mathbf{m}_{x,k} = \mathbf{F}_{x,k-1}\mathbf{m}_{x,k-1}$
Cov. Prop.	$\mathbf{P}_{xx,k} = \mathbf{F}_{x,k-1}\mathbf{P}_{xx,k-1}\mathbf{F}_{x,k-1}^T + \mathbf{F}_{w,k-1}\mathbf{P}_{ww,k-1}\mathbf{F}_{w,k-1}^T$
Exp. Meas.	$\mathbf{m}_{z,k}^- = \mathbf{H}_{x,k}\mathbf{m}_{x,k}^-$
Cross Cov.	$\mathbf{P}_{xz,k}^- = \mathbf{P}_{xx,k}^-\mathbf{H}_{x,k}^T$
Innov. Cov.	$\mathbf{P}_{zz,k}^- = \mathbf{H}_{x,k}\mathbf{P}_{xx,k}^-\mathbf{H}_{x,k}^T + \mathbf{H}_{v,k}\mathbf{P}_{vv,k}\mathbf{H}_{v,k}^T$
Kalman Gain	$\mathbf{K}_k = \mathbf{P}_{xz,k}^-(\mathbf{P}_{zz,k}^-)^{-1}$
Mean Upd.	$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k[\mathbf{z}_k - \mathbf{m}_{z,k}^-]$
Cov. Upd.	$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^-\mathbf{K}_k^T - \mathbf{K}_k(\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k\mathbf{P}_{zz,k}^-\mathbf{K}_k^T$

3.4 The Discrete Extended Kalman Filter

Now we turn to the task of extending our discrete Kalman filter to the case where we have nonlinear dynamics and nonlinear observations.

In this case, our discrete dynamical system is assumed to have the form

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{w}_{k-1})$$

where $\mathbf{f}(\cdot, \cdot)$ is the nonlinear dynamics governing the transition of the state from \mathbf{x}_{k-1} to \mathbf{x}_k , the initial state is taken to have mean $\mathbf{m}_{x,0}$ and covariance $\mathbf{P}_{xx,0}$, and the process noise, \mathbf{w}_{k-1} , is taken to be a zero-mean, white-noise sequence, with covariance $\mathbf{P}_{ww,k-1}$.

First, we take the expected value of the difference equation

$$\begin{aligned}\mathbf{m}_{x,k} &= \text{E}\{\mathbf{x}_k\} \\ &= \text{E}\{\mathbf{f}(\mathbf{x}_{k-1}, \mathbf{w}_{k-1})\}\end{aligned}$$

Since computing the expected value of an arbitrary nonlinear function is a non-trivial task, we will proceed by expanding the nonlinear dynamics by using a first-order Taylor

series expansion (FOTSE) about the current mean of the state and of the process noise; that is

$$\mathbf{f}(\mathbf{x}_{k-1}, \mathbf{w}_{k-1}) \approx \mathbf{f}(\mathbf{m}_{x,k-1}, \mathbf{0}_w) + \mathbf{F}_x(\mathbf{m}_{x,k-1})(\mathbf{x}_{k-1} - \mathbf{m}_{x,k-1}) + \mathbf{F}_w(\mathbf{m}_{x,k-1})\mathbf{w}_{k-1}$$

where

$$\mathbf{F}_a(\mathbf{m}_{k-1}) = \left[\frac{\partial \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{w}_{k-1})}{\partial \mathbf{a}_{k-1}} \middle|_{\substack{\mathbf{x}_{k-1}=\mathbf{m}_{x,k-1} \\ \mathbf{w}_{k-1}=\mathbf{0}_w}} \right]$$

We can now substitute this FOTSE into the evolution equation for the mean to find

$$\begin{aligned} \mathbf{m}_{x,k} &= \text{E}\{\mathbf{f}(\mathbf{m}_{x,k-1}, \mathbf{0}_w) + \mathbf{F}_x(\mathbf{m}_{x,k-1})\mathbf{e}_{x,k-1} + \mathbf{F}_w(\mathbf{m}_{x,k-1})\mathbf{w}_{k-1}\} \\ &= \text{E}\{\mathbf{f}(\mathbf{m}_{x,k-1}, \mathbf{0}_w)\} + \text{E}\{\mathbf{F}_x(\mathbf{m}_{x,k-1})\mathbf{e}_{x,k-1}\} + \text{E}\{\mathbf{F}_w(\mathbf{m}_{x,k-1})\mathbf{w}_{k-1}\} \end{aligned}$$

where $\mathbf{e}_{x,k-1} = \mathbf{x}_{k-1} - \mathbf{m}_{x,k-1}$ is the estimation error. If we assume that the mean at t_{k-1} is deterministic, then $\mathbf{f}(\mathbf{m}_{x,k-1}, \mathbf{0}_w)$ is deterministic and the Jacobians are deterministic. If we apply these assumptions and if our estimator is unbiased, then it follows that the propagation equation for the mean is

$$\mathbf{m}_{x,k} = \mathbf{f}(\mathbf{m}_{x,k-1}, \mathbf{0}_w)$$

To develop the covariance propagation equation, we begin by developing a propagation equation for the estimation error. As such, let the estimation error at step k be

$$\mathbf{e}_{x,k} = \mathbf{x}_k - \mathbf{m}_{x,k}$$

We can apply the dynamics equations for the true and estimated states to yield

$$\mathbf{e}_{x,k} = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{w}_{k-1}) - \mathbf{f}(\mathbf{m}_{x,k-1}, \mathbf{0}_w)$$

Recalling the FOTSE of the nonlinear dynamics, it follows that the dynamics of the estimation error are

$$\mathbf{e}_k = \mathbf{F}_x(\mathbf{m}_{x,k-1})\mathbf{e}_{x,k-1} + \mathbf{F}_{w,k-1}(\mathbf{m}_{x,k-1})\mathbf{w}_{k-1}$$

From this difference equation, we can develop our covariance propagation equation. It is, however, worth noting that we have already solved this problem in the development of the discrete Kalman filter. If we replace $\mathbf{F}_{x,k-1}$ with $\mathbf{F}_x(\mathbf{m}_{x,k-1})$ and $\mathbf{F}_{w,k-1}$ with $\mathbf{F}_w(\mathbf{m}_{x,k-1})$, it follows that the covariance evolves according to

$$\mathbf{P}_{xx,k} = \mathbf{F}_x(\mathbf{m}_{x,k-1})\mathbf{P}_{xx,k-1}\mathbf{F}_x^T(\mathbf{m}_{x,k-1}) + \mathbf{F}_w(\mathbf{m}_{x,k-1})\mathbf{P}_{ww,k-1}\mathbf{F}_w^T(\mathbf{m}_{x,k-1})$$

It is important to remind ourselves of a few assumptions that are necessary for us to get to this result:

- the matrices $\mathbf{F}_x(\mathbf{m}_{x,k-1})$ and $\mathbf{F}_w \mathbf{m}_{k-1}$ are deterministic.
- the process noise and the state estimation error are uncorrelated

$$\mathrm{E}\{\mathbf{w}_{k-1} \mathbf{e}_{x,k-1}^T\} = \mathbf{0} \quad \text{and} \quad \mathrm{E}\{\mathbf{e}_{x,k-1} \mathbf{w}_{k-1}^T\} = \mathbf{0}$$

When we acquire a new measurement at step k , we want to fuse this information with the outputs from our propagation stage. The new measurement is, in general, a nonlinear function of the state and of the measurement noise that is taken to be of the form

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k)$$

where

$$\mathrm{E}\{\mathbf{v}_k\} = \mathbf{0}_v \quad \text{and} \quad \mathrm{E}\{\mathbf{v}_k \mathbf{v}_\ell^T\} = \mathbf{P}_{vv,k} \delta_{k\ell}$$

In words, the measurement noise is zero-mean and white.

This is another problem that we've already solved when we developed our original extended Kalman filter, so we will leverage our previously developed results. As such, the

update for the mean and covariance are

$$\mathbf{m}_k^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{m}_{z,k}^-)$$

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- \mathbf{K}_k^T - \mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

where

$$\mathbf{m}_{z,k}^- = \text{E}\{\mathbf{z}_k\}$$

$$\mathbf{P}_{xz,k}^- = \text{E}\left\{(\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T\right\}$$

$$\mathbf{P}_{zz,k}^- = \text{E}\left\{(\mathbf{z}_k - \mathbf{m}_{z,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T\right\}$$

and \mathbf{K}_k is any linear gain.

In the special situation where we want to intelligently choose the gain, we can select the Kalman gain as

$$\mathbf{K}_k = \mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1}$$

which we have shown to minimize the posterior mean-square estimation error. This minimization is exact for the case of linear measurements, but it becomes approximate when we have nonlinear measurements.

From our treatment of nonlinear measurements for the extended Kalman filter, we know that linearization leads us to compute the required expected values as

$$\mathbf{m}_{z,k}^- = \mathbf{h}(\mathbf{m}_{x,k}^-, \mathbf{0}_v)$$

$$\mathbf{P}_{xz,k}^- = \mathbf{P}_{xx,k}^- \mathbf{H}_x^T(\mathbf{m}_{x,k}^-)$$

$$\mathbf{P}_{zz,k}^- = \mathbf{H}_x(\mathbf{m}_{x,k}^-) \mathbf{P}_{xx,k}^- \mathbf{H}_x^T(\mathbf{m}_{x,k}^-) + \mathbf{H}_v(\mathbf{m}_{x,k}^-) \mathbf{P}_{vv,k}^- \mathbf{H}_v^T(\mathbf{m}_{x,k}^-)$$

This completes the development of the discrete extended Kalman filter. As with the EKF, the use of linearization makes the filter outputs trajectory (or path) dependent.

To summarize, we put everything together in a single table

System Model	$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{w}_{k-1})$
Meas. Model	$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k)$
Init. Cond.	$\mathbf{m}_{x,0} = \text{E}\{\mathbf{x}(t_0)\}$ $\mathbf{P}_{xx,0} = \text{E}\{(\mathbf{x}(t_0) - \mathbf{m}_{x,0})(\mathbf{x}(t_0) - \mathbf{m}_{x,0})^T\}$
Mean Prop.	$\mathbf{m}_{x,k} = \mathbf{f}(\mathbf{m}_{x,k-1}, \mathbf{0}_w)$
Cov. Prop.	$\mathbf{P}_{xx,k} = \mathbf{F}_x(\mathbf{m}_{x,k-1})\mathbf{P}_{xx,k-1}\mathbf{F}_x^T(\mathbf{m}_{x,k-1}) + \mathbf{F}_w(\mathbf{m}_{x,k-1})\mathbf{P}_{ww,k-1}\mathbf{F}_w^T(\mathbf{m}_{x,k-1})$
Exp. Meas.	$\mathbf{m}_{z,k}^- = \mathbf{h}(\mathbf{m}_{x,k}^-, \mathbf{0}_v)$
Cross Cov.	$\mathbf{P}_{xz,k}^- = \mathbf{P}_{xx,k}^-\mathbf{H}_x^T(\mathbf{m}_{x,k}^-)$
Innov. Cov.	$\mathbf{P}_{zz,k}^- = \mathbf{H}_x(\mathbf{m}_{x,k}^-)\mathbf{P}_{xx,k}^-\mathbf{H}_x^T(\mathbf{m}_{x,k}^-) + \mathbf{H}_v(\mathbf{m}_{x,k}^-)\mathbf{P}_{vv,k}\mathbf{H}_v^T(\mathbf{m}_{x,k}^-)$
Kalman Gain	$\mathbf{K}_k = \mathbf{P}_{xz,k}^-(\mathbf{P}_{zz,k}^-)^{-1}$
Mean Upd.	$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k[\mathbf{z}_k - \mathbf{m}_{z,k}^-]$
Cov. Upd.	$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^-\mathbf{K}_k^T - \mathbf{K}_k(\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k\mathbf{P}_{zz,k}^-\mathbf{K}_k^T$

3.4.1 Example: Clohessy-Wiltshire Problem

Let's take a look at a problem with linear, discrete-time dynamics and nonlinear, discrete-time measurements. For this problem, we will consider an object that is moving in orbit nearby the International Space Station (ISS).

We are going to use the Clohessy-Wiltshire (CW) equations to model the motion of a particle that is in close proximity to the ISS, which we will assume is in a circular orbit about Earth. The CW equations can be formulated in continuous-time or discrete-time, so we will use the discrete-time model to perform a discrete prediction step in our EKF.

General, three-dimensional motion can be simulated with the CW equations; however, this makes visualization and analysis more challenging. As such, we will restricting this example to only in-plane motion. To represent the planar motion, we define the state as

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

where x and y are the relative positions of the object and \dot{x} and \dot{y} are its relative velocities, all with respect to the ISS.

The CW equations allow us to form $\mathbf{F}_{x,k-1}$ for a linear propagation as

$$\mathbf{F}_{x,k-1} = \begin{bmatrix} 4 - 3c & 0 & s/n & 2(1 - c)/n \\ 6(s - n\Delta t) & 1 & 2(c - 1)/n & (4s - 3n\Delta t)/n \\ 3ns & 0 & c & 2s \\ 6n(c - 1) & 0 & -2s & 4c - 3 \end{bmatrix}$$

where $s = \sin(n\Delta t)$, $c = \cos(n\Delta t)$, n is the mean motion of the ISS, and $\Delta t = t_k - t_{k-1}$. For our simulation, we will take the ISS to be in a circular orbit with a semimajor axis of 6775 km, which is used to determine the mean motion of the ISS.

To complete the dynamics, we will take the process noise is directly added into the state dynamics, such that $\mathbf{F}_{w,k-1} = \mathbf{I}_4$. The covariance of the process noise is taken to be constant, with values of

$$\mathbf{P}_{ww} = \text{diag}([1 \times 10^{-6} \quad 1 \times 10^{-6} \quad 1 \times 10^{-9} \quad 1 \times 10^{-9}])$$

in units of m² and (m/s)². Samples of the process noise are drawn from a zero-mean Gaussian distribution to simulate noise effects on the evolution of the true state.

For the measurements, we will consider range measurements of the form

$$\mathbf{h}(\mathbf{x}_k, v_k) = \sqrt{x_k^2 + y_k^2} + v_k$$

where v_k is zero-mean with covariance $P_{vv} = 2 \text{ m}^2$. Samples of the measurement noise are drawn from a Gaussian distribution to corrupt the range measurements. This range model assumes that the ISS is a point mass and that the sensor is located at the center of the ISS. Given the range measurement model, we can also formulate the Jacobians required for processing data in the discrete EKF as

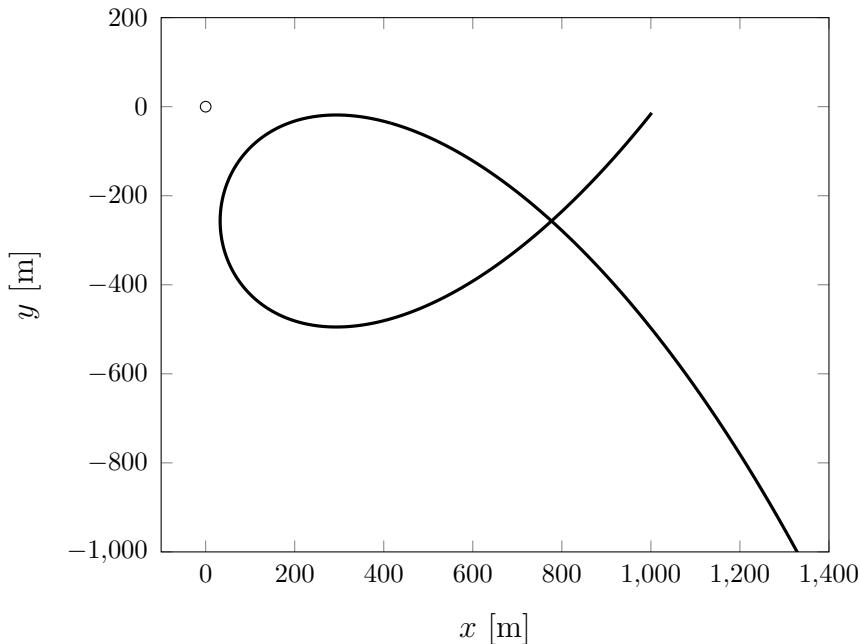
$$\mathbf{H}_{x,k} = \left[\begin{array}{cc} \frac{x_k}{\sqrt{x_k^2+y_k^2}} & \frac{y_k}{\sqrt{x_k^2+y_k^2}} \\ 0 & 0 \end{array} \right] \Bigg|_{\mathbf{x}_k=\mathbf{m}_{x,k}^-} \quad \text{and} \quad \mathbf{H}_{v,k} = 1$$

Finally, to kick everything off, we need initial estimates for the mean and covariance of the object. These are taken to be

$$\mathbf{m}_{x,0} = [1000 \ 0 \ -1.23 \ -1.73]^T \quad \text{and} \quad \mathbf{P}_{xx,0} = \text{diag}([20^2 \ 20^2 \ 0.2^2 \ 0.2^2])$$

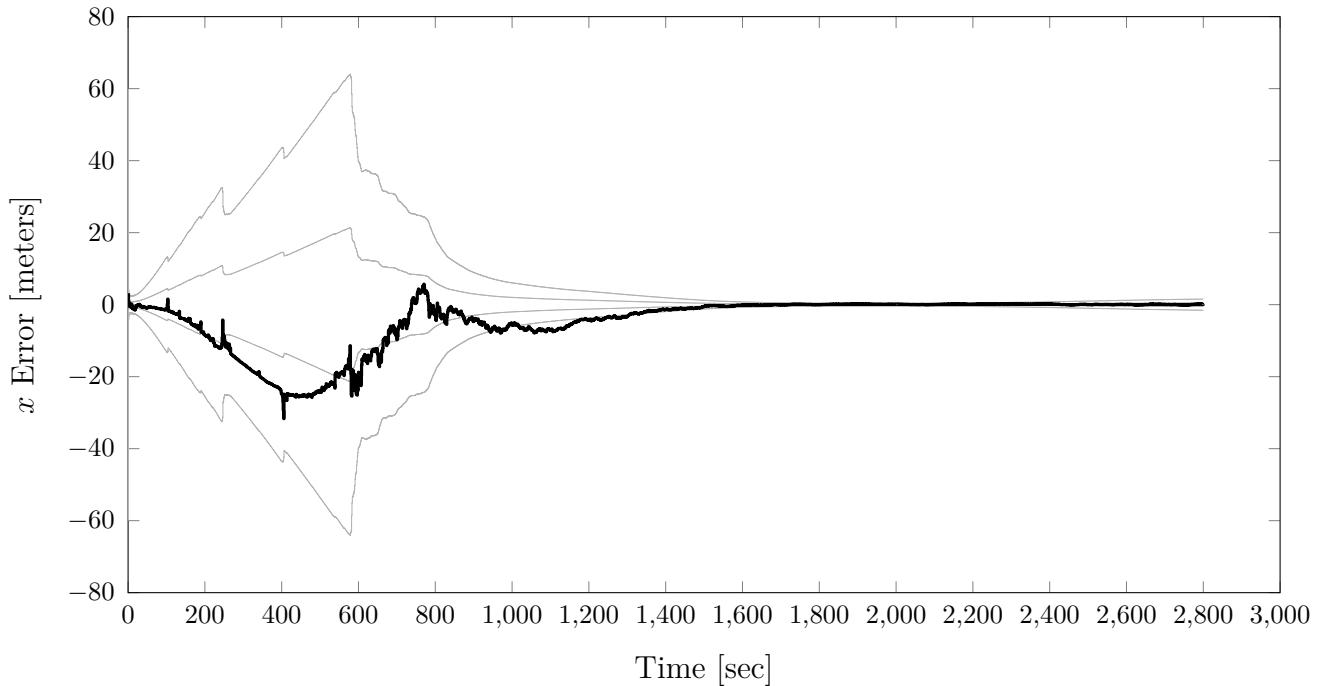
with units for positions of m and velocities of m/s. We assume that the initial true state is drawn from a Gaussian distribution with this mean and covariance.

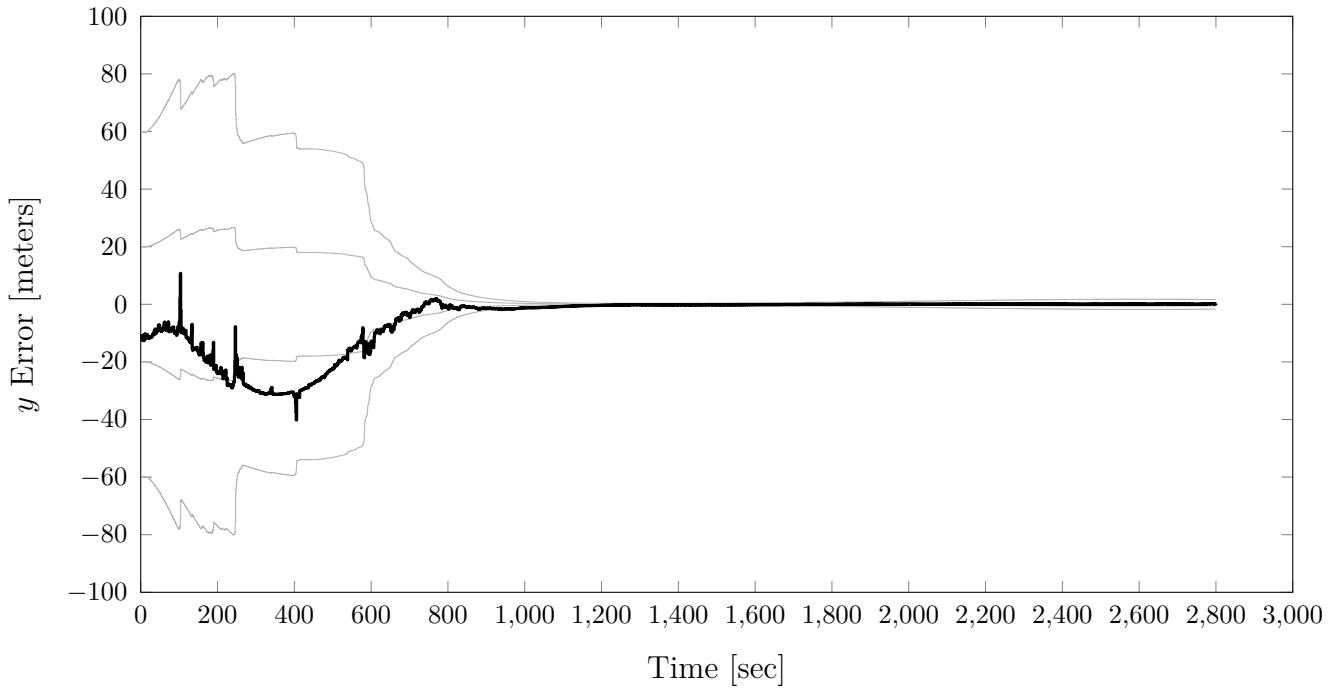
We use all of this information to implement the discrete extended Kalman filter (with a truly linear propagation step). The true trajectory of the object can be seen in the following figure (with the ISS at the origin as a circle).



We see here that the object makes a pretty close approach to the ISS before moving away again. This trajectory is used to generate measurement data, the measurement data is subjected to noise, and the noisy measurement data is used to produce a tracking solution of the object.

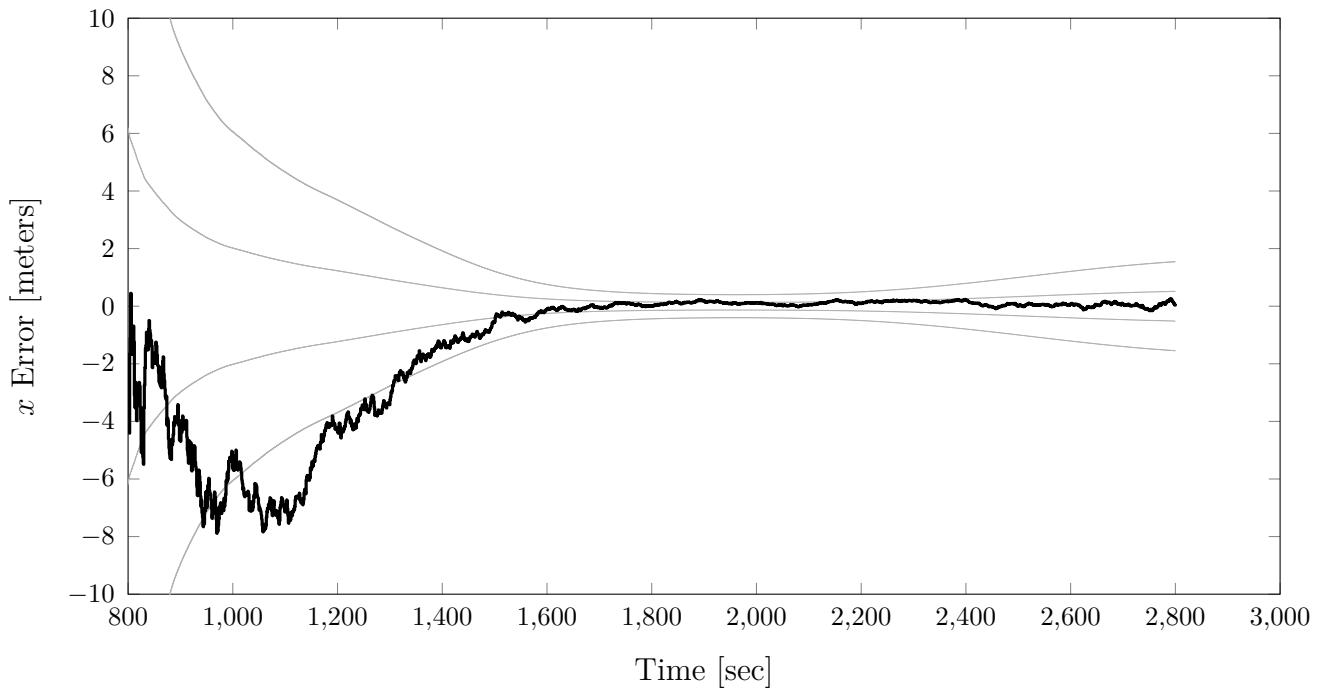
The error in the position states and the corresponding $1\sigma/3\sigma$ intervals are given in the following figures.

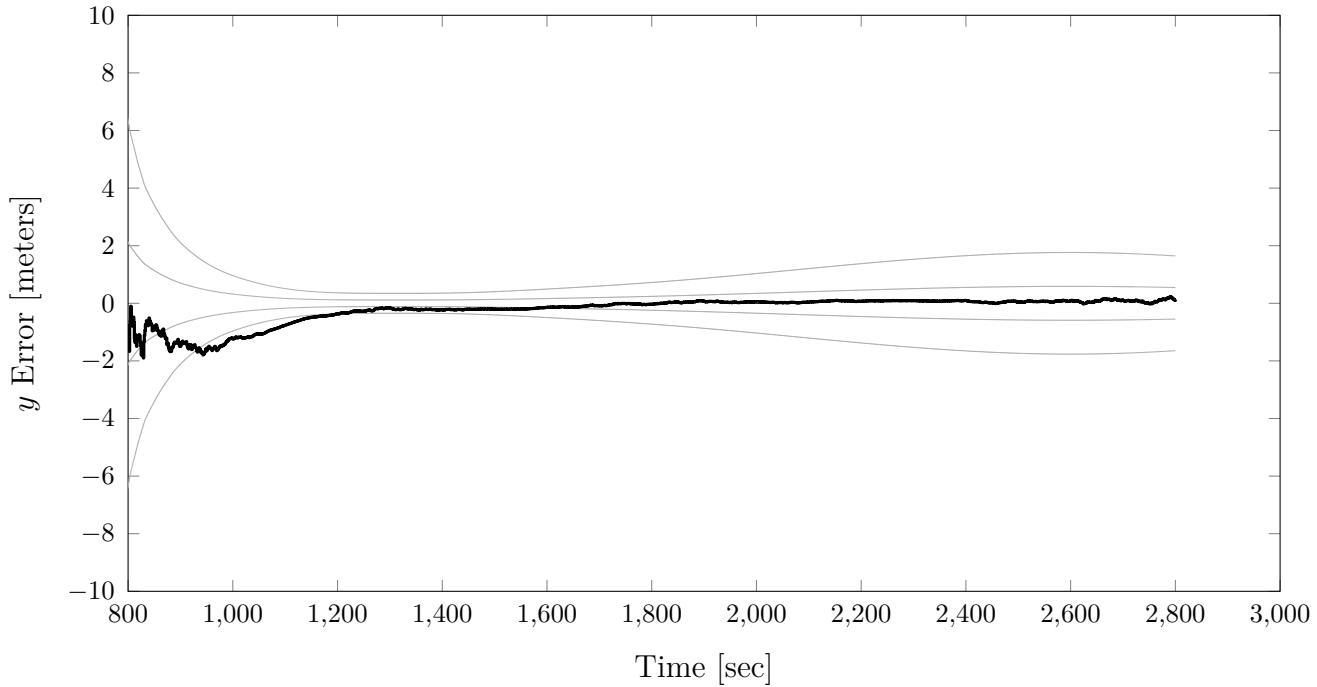




As expected, we start out with relative large uncertainties, but those get smaller as we process measurement data.

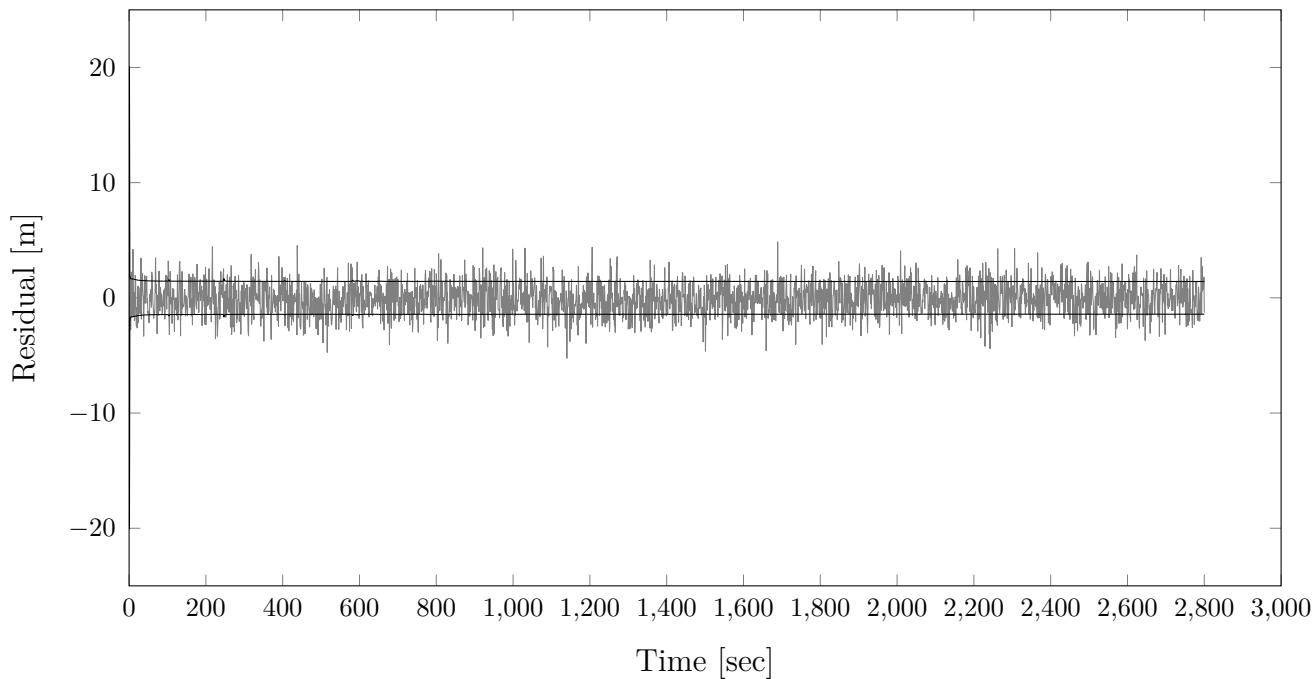
We can also look at a zoomed-in perspective to see a bit more detail at the end of the simulation.





By the end of the simulation, we have a pretty good idea where the object is, although that knowledge appears to be degrading towards the very end.

Additionally, we can look at the measurement residuals and its 1σ interval as defined by the measurement covariance.



4

Alterations to the Kalman Filter

4.1 Care and Feeding of Your Kalman Filter

There are many areas where it is important to pay attention to the inputs and outputs of a Kalman filter, especially when we are considering extended Kalman filters in which approximations are made.

For instance, from the modeling side, the EKF relies heavily on Jacobians to properly model the linearization-based approach that we leverage for approximating the nonlinear functions involved.

There are other areas that also require some attention. We assume that our filter models properly describe the system under consideration, and this is not guaranteed to be the case, especially in light of some of the approximations that we have made. We have touched on some elements of filter tuning, but this is primarily a dynamics-driven process. It is then natural to wonder what happens when our measurement models are wrong to some extent? What are the influences of bad data?

Another question that often arises is should we use the Kalman gain? Are there situations in which this gain is too aggressive? And if so, what other types of linear gains might we consider?

4.1.1 Validation of Filter Implementations

We begin by considering a few validation steps that we can take to ensure our models are properly configured in the EKF. In doing so, we consider both the propagation stage of the EKF and the update stage of the EKF. We will focus on the continuous-discrete form of the EKF, but the techniques that we will develop can be applied to other forms of the EKF, as well.

From the EKF propagation equations, it is seen that several things need to be validated in order to validate the propagation stage of the EKF:

1. the dynamics function, $\mathbf{f}(\mathbf{m}_x(t), \mathbf{0}_w)$
2. the dynamics Jacobian, $\mathbf{F}_x(\mathbf{m}_x(t))$
3. the state transition matrix, $\Phi(t, t_0)$

It should be noted that the validation steps are taken with respect to the filtering equations and not those representing the true state evolution. That is, we want to verify that the filter is consistent with the (assumed) true dynamics of the system and that subsequent filtering operations are self-consistent. While the equations representing the true state are utilized in validating the equations representing the estimated state, the true state equations are taken to be the reference.

Validation of the dynamics function is broken down into two steps:

- (a) point-wise validation
- (b) integration-based validation

Point-wise validation is performed by evaluating the true dynamics at a reference state $\mathbf{x}^*(t)$ with zero process noise and evaluating the estimated dynamics at the same reference state.

- If these two evaluations are numerically equivalent, then point-wise validation is complete.
- If these two evaluations differ, then a discrepancy exists between the dynamics evaluations.

Integration-based validation is performed by integrating the true dynamics and estimated dynamics over a time interval and comparing the true state to the estimated state. We need to ensure that consistency is maintained between these solutions for meaningful results to be obtained, including:

- The method of integration when implementing the true and estimated dynamics should be the same.
- Process noise should not be applied when implementing the true dynamics.
- Any exogenous inputs should be uncorrupted.

- The initial conditions must be set equal to a reference state, i.e.

$$\mathbf{x}_0 = \mathbf{x}_0^* \quad \text{and} \quad \mathbf{m}_{x,0} = \mathbf{x}_0^*$$

If the true and estimated states are identical, that is if

$$\mathbf{e}_x(t) = \mathbf{x}(t) - \mathbf{m}_x(t) = \mathbf{0}_n \quad \forall t$$

to within some allowable tolerance, then the estimated dynamics are consistent with the true dynamics.

This process can also be used to investigate alternative integration methods or different types of approximations to the true dynamics of the problem. That is, if the true states are integrated according to a high-precision integrator and the filter uses simpler integration methods or simpler dynamics, we can use this approach to assess the numerical error introduced into our filter by the simplifications. It should be noted, however, that this process is highly dependent upon the problem under consideration.

Now, we turn to validating the elements required for propagating the covariance matrix. Either method that we have discussed for propagating the covariance matrix requires

the dynamics Jacobian, $\mathbf{F}_x(\mathbf{m}_x(t))$. The Jacobian is validated using finite differences approximations for the derivative. As such, numerical methods for calculation of the derivatives can be considered. For some function $\mathbf{g}(\mathbf{x})$, finite difference approximations include

- forward differences

$$\frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} \approx \frac{\mathbf{g}(\mathbf{x} + \Delta \mathbf{x}) - \mathbf{g}(\mathbf{x})}{\Delta \mathbf{x}}$$

- backward differences

$$\frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} \approx \frac{\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x} - \Delta \mathbf{x})}{\Delta \mathbf{x}}$$

- central differences

$$\frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} \approx \frac{\mathbf{g}(\mathbf{x} + \Delta \mathbf{x}) - \mathbf{g}(\mathbf{x} - \Delta \mathbf{x})}{2\Delta \mathbf{x}}$$

The central differences method will be used exclusively from here on out; however, the following discussions may be viewed equivalently under any of the above numerical differentiation techniques.

In the preceding expressions for approximate derivatives, an abuse of notation is utilized by writing the approximation as a quotient with a vector appearing in the denominator. This is intended as shorthand notation for

$$\left[\frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} \right]_i \approx \frac{\mathbf{g}(\mathbf{x} + \Delta x_i \mathbf{e}_i) - \mathbf{g}(\mathbf{x} - \Delta x_i \mathbf{e}_i)}{2\Delta x_i}$$

where

- $[\cdot]_i$ denotes the i^{th} column of the matrix
- Δx_i is the i^{th} element of $\Delta \mathbf{x}$
- \mathbf{e}_i is a vector of all zeros except for the i^{th} element, which is one

Now we proceed to the validation steps for the EKF, utilizing the approximation methods for derivatives.

Validation of the dynamics Jacobian is performed by comparing the analytic compu-

tation of $\mathbf{F}_x(\mathbf{m}_x(t))$ to the numerical approximation given by

$$\mathbf{F}_x(\mathbf{m}_x(t)) \approx \frac{\mathbf{f}(\mathbf{m}_x(t) + \Delta\mathbf{x}(t), \mathbf{0}_w) - \mathbf{f}(\mathbf{m}_x(t) - \Delta\mathbf{x}(t), \mathbf{0}_w)}{2\Delta\mathbf{x}(t)}$$

where t is fixed at some time and $\mathbf{m}_x(t)$ is chosen as some reference state. It is good practice to choose the reference state such that no element of the state is identically equal to zero.

If the analytic computation is numerically equivalent (to within some tolerance) of the numerical approximation then the dynamics Jacobian is consistent with the dynamics function. This comparison is general done on an element-wise based, and the magnitudes of the elements should be considered when performing the comparison.

The numerical computation of the Jacobian depends upon the choice of Δx_i . Selecting these values is, again, problem-dependent, but usually considering the quantities involved (position, velocity, etc.) can lead to reasonable selections of the step sizes. Too large of values will result in inaccurate results for the finite differences. Too small of steps will also result in inaccurate results for the finite differences.

A newer technique is to use complex-step derivatives, which are far less sensitive to small step sizes.¹ This method requires that all of the functions and computations in the nonlinear function support complex numbers, which is not always trivial. As such, we will not consider this method here.

We can also use this method to validate the Jacobian of the dynamics with respect to the process noise, $\mathbf{F}_w(\mathbf{m}_x(t))$. Validation of this Jacobian is carried out in the same manner as is done for $\mathbf{F}_x(\mathbf{m}_x(t))$, except that perturbations are applied to the process noise instead of to the state. Once again, it is best practice to ensure that no element of the state is identically equal to zero when carrying out this comparison.

One of the methods of propagating the covariance matrix requires the state transition matrix. Provided that our dynamics Jacobian is accurate, the state transition matrix integration should also be accurate. However, it never hurts to be sure.

The state transition matrix $\Phi(t, t_0)$ describes the linear mapping of perturbations from

¹Squire, W. and Trapp, G., “Using Complex Variables to Estimate Derivatives of Real Functions,” *SIAM Review*, Vol. 40, No. 1, March 1998, pp. 110–112.

time t_0 to time t , and can be conceptually represented by

$$\Phi(t, t_0) = \frac{\partial \mathbf{m}_x(t)}{\partial \mathbf{m}_x(t_0)}$$

Numerically, this may be evaluated by perturbing the state at time t_0 ,

$$\mathbf{m}_{(+)}(t_0) = \mathbf{m}(t_0) + \Delta \mathbf{x}(t_0) \quad \text{and} \quad \mathbf{m}_{(-)}(t_0) = \mathbf{m}(t_0) - \Delta \mathbf{x}(t_0),$$

integrating forward to time t ,

$$\mathbf{m}_{(+)}(t_0) \xrightarrow{\mathbf{f}(\cdot)} \mathbf{m}_{(+)}(t) \quad \text{and} \quad \mathbf{m}_{(-)}(t_0) \xrightarrow{\mathbf{f}(\cdot)} \mathbf{m}_{(-)}(t),$$

and computing a finite differences approximation of the derivative,

$$\Phi(t, t_0) = \frac{\mathbf{m}_{(+)}(t) - \mathbf{m}_{(-)}(t)}{2\Delta \mathbf{x}(t_0)}$$

Comparison of the numerical approximation of the state transition matrix to the analytic computation then may be used to validate the analytic computation. As with other

comparisons, the two computations should match (element-wise) to within some desired tolerance.

Now, we move to validation of the EKF update equations.

From the EKF update equations, it is seen that two things need to be validated in order to validate the update stage of the EKF:

1. the measurement function, $\mathbf{h}(\mathbf{m}_{x,k}^-, \mathbf{0}_v)$
2. the measurement Jacobian with respect to the state, $\mathbf{H}_x(\mathbf{m}_{x,k}^-)$
3. the measurement Jacobian with respect to the measurement noise, $\mathbf{H}_v(\mathbf{m}_{x,k}^-)$

Once again, when validating these functions, we are assuming that there is a “truth side” evaluation and a “filter side” evaluation. We assume that the “truth side” is correct, and we want to validate the “filter side” against this.

Validation of the measurement function is performed via a point-wise comparison, which is done by evaluating the true measurement at a reference state \mathbf{x}_k^* with zero measurement noise and evaluating the estimated measurement at the same reference state.

- If these two evaluations are numerically equivalent, then point-wise validation is complete.
- If these two evaluations differ, then a discrepancy exists between the measurement evaluations.

This process should be performed for each available sensor; that is, altimeter, velocimeter, star camera, etc. validations should be done separately of one another.

As with the dynamics validation, this method can also be used to assess acceptability of reduced-complexity modeling, such as using a topography-based altimeter model for the truth generation and an ellipsoidal altimeter model for the filter evaluation. Again, this is a highly problem-dependent element that will not be treated further here.

Now, we turn to validating the measurement Jacobians. Regardless of which Jacobian we are evaluating, the process is the same. As such, we will focus on $\mathbf{H}_x(\mathbf{m}_{x,k}^-)$, with the understanding that the validation of $\mathbf{H}_v(\mathbf{m}_{x,k}^-)$ is done in a similar way. Validation of the measurement Jacobian is performed by comparing the analytic computation of

$\mathbf{H}_x(\mathbf{m}_{x,k}^-)$ to the numerical approximation given by

$$\mathbf{H}_x(\mathbf{m}_{x,k}^-) \approx \frac{\mathbf{h}(\mathbf{m}_{x,k}^- + \Delta\mathbf{x}_k, \mathbf{0}_v) - \mathbf{h}(\mathbf{m}_{x,k}^- - \Delta\mathbf{x}_k, \mathbf{0}_v)}{2\Delta\mathbf{x}_k}$$

where $\mathbf{m}_{x,k}^-$ is chosen as a reference state. It is important to choose the reference state as generally as possible; that is, *do not* use a state with zero elements.

If the analytic computation is numerically equivalent (to within some tolerance) of the numerical approximation then the measurement Jacobian is consistent with the measurement function. This process should be performed for each available sensor, i.e. altimeter, velocimeter, and star camera validations should be done separately of one another.

The same comments from the dynamics Jacobian process apply. Comparisons should be done element-wise, and magnitudes of the elements should be considered. The evaluation depends on the choice of Δx_i , which is problem-dependent, but the type of and units of the state elements lend insight. Complex-step derivatives, which are far less sensitive to small step sizes, can also be used, but, as a reminder, this method requires that all of the functions and computations in the nonlinear function support complex numbers, which is not always trivial.

4.1.2 Measurement Editing

In practical use of the extended Kalman filter, care must be taken by the user in order to avoid potential pitfalls of accepting unrealistic sensor measurements. It is inevitable that, due to the use of real sensors, not all data will be good. Spurious returns can easily occur in many sensor systems.

One such case is a laser ranging system used during terminal descent and landing. Dust that is kicked up during landing can cause the laser ranging system to return measurements of the range to the dust instead of the range to the ground. This situation can be avoided by not using the ranging system during the final landing phase, but other such sensor “glitches” or spurious measurements can occur.

As of now, the EKF has no way of knowing if a spurious measurement is received, unless the sensor system provides additional information. We do not want to rely solely on the sensor system giving us the green light for processing data; we want our filter to be robust to spurious data.

A “bad” measurement, if processed using statistics of good data, will result in a de-

crease to the filter covariance but can also result in a large update to the estimated state. This is the situation we wish to avoid by checking for statistical consistency of the measurement data with our model and editing out the bad data. This process is what we will call **measurement editing**.

Measurement editing is the act of eliminating measurements (or innovations) that do not agree with the associated innovation covariance, thereby eliminating the inclusion of statistically inconsistent measurements. Monitoring of the measurement editing process can also be used to detect sensor failures; measurements that consistently disagree with the innovation covariance are typically attributed to a failed sensor. This could also indicate a flaw in our model, but if we are processing onboard and do not have access to a different model, we should still edit the data so that our filter does not process bad data.

The presented method

1. reviews the calculation of the residual covariance
2. develops a statistical check to determine the acceptability of measurements for inclusion into the filter

In the extended Kalman filter, it is assumed that at time t_k a measurement is made available of the form

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k)$$

where \mathbf{v}_k is a zero-mean, time-wise uncorrelated sequence with covariance $\mathbf{P}_{vv,k}$. The expected measurement computed in the filter is taken to be

$$\mathbf{m}_{z,k}^- = \mathbf{h}(\mathbf{m}_{x,k}^-, \mathbf{0}_v)$$

With the measurement and the estimate of the measurement, the innovation is computed as

$$\mathbf{e}_{z,k}^- = \mathbf{z}_k - \mathbf{m}_{z,k}^-$$

We have previously used a first-order Taylor series expansion to express the innovation as a linear combination of the state estimation error and the measurement noise, such that under this approximation, the innovation is zero mean and has covariance

$$\mathbf{P}_{zz,k}^- = \mathbf{H}_x(\mathbf{m}_{x,k}^-) \mathbf{P}_{xx,k}^- \mathbf{H}_x^T(\mathbf{m}_{x,k}^-) + \mathbf{H}_v(\mathbf{m}_{x,k}^-) \mathbf{P}_{vv,k}^- \mathbf{H}_v^T(\mathbf{m}_{x,k}^-)$$

where it is important to keep in mind all of the assumptions that go into building the EKF.

Now, we describe a scalar measurement editing method.

Given q vector measurements at time t_k of the form $\mathbf{z}_k^{(i)}$ along with associated measurement noise covariances $\mathbf{P}_{vv,k}^{(i)}$, first compute the associated estimated measurements, $\mathbf{m}_{z,k}^{(i)-}$. The i^{th} measurement innovation is given by

$$\mathbf{e}_{z,k}^{(i)-} = \mathbf{z}_k^{(i)} - \mathbf{m}_{z,k}^{(i)-}$$

The total measurement, expected measurement, and innovation are assembled by concatenating the corresponding terms from each of the sensors to yield

$$\mathbf{z}_k = \begin{bmatrix} \mathbf{z}_k^{(1)} \\ \mathbf{z}_k^{(2)} \\ \vdots \\ \mathbf{z}_k^{(q)} \end{bmatrix}, \quad \mathbf{m}_{z,k}^- = \begin{bmatrix} \mathbf{m}_{z,k}^{(1)-} \\ \mathbf{m}_{z,k}^{(2)-} \\ \vdots \\ \mathbf{m}_{z,k}^{(q)-} \end{bmatrix}, \quad \text{and} \quad \mathbf{e}_{z,k}^- = \begin{bmatrix} \mathbf{e}_{z,k}^{(1)-} \\ \mathbf{e}_{z,k}^{(2)-} \\ \vdots \\ \mathbf{e}_{z,k}^{(q)-} \end{bmatrix}$$

where the total dimension of \mathbf{z}_k is m . Next, compute the associated measurement Jacobians, $\mathbf{H}_x^{(i)}(\mathbf{m}_{x,k}^-)$ and $\mathbf{H}_v^{(i)}(\mathbf{m}_{x,k}^-)$, and concatenate them as

$$\mathbf{H}_x(\mathbf{m}_{x,k}^-) = \begin{bmatrix} \mathbf{H}_x^{(1)}(\mathbf{m}_{x,k}^-) \\ \mathbf{H}_x^{(2)}(\mathbf{m}_{x,k}^-) \\ \vdots \\ \mathbf{H}_x^{(q)}(\mathbf{m}_k^-) \end{bmatrix} \quad \text{and} \quad \mathbf{H}_v(\mathbf{m}_{x,k}^-) = \begin{bmatrix} \mathbf{H}_v^{(1)}(\mathbf{m}_{x,k}^-) \\ \mathbf{H}_v^{(2)}(\mathbf{m}_{x,k}^-) \\ \vdots \\ \mathbf{H}_v^{(q)}(\mathbf{m}_{x,k}^-) \end{bmatrix}$$

Determine the innovation covariance as

$$\mathbf{P}_{zz,k}^- = \mathbf{H}_x(\mathbf{m}_{x,k}^-) \mathbf{P}_{xx,k}^- \mathbf{H}_x^T(\mathbf{m}_{x,k}^-) + \mathbf{H}_v(\mathbf{m}_{x,k}^-) \mathbf{P}_{vv,k}^- \mathbf{H}_v^T(\mathbf{m}_{x,k}^-)$$

where $\mathbf{P}_{vv,k}$ is generally formed as a block diagonal matrix with entries $\mathbf{P}_{vv,k}^{(i)}$. This is true if the different measurement noises are uncorrelated. If the measurement noises are correlated, however, there will be off-block-diagonal elements in $\mathbf{P}_{vv,k}$.

Let n_j be a user-specified editing tolerance, $[\mathbf{e}_{z,k}]_{(j)}$ be the j^{th} element of the innovation vector, and $[\mathbf{P}_{zz,k}^-]_{(j,j)}$ be the j^{th} diagonal element of $\mathbf{P}_{zz,k}^-$.

If

$$|[\boldsymbol{e}_{z,k}^-]_{(j)}| > n_j \sqrt{[\boldsymbol{P}_{zz,k}^-]_{(j,j)}}$$

the j^{th} element of the measurement is edited by removing the corresponding entry in $\boldsymbol{e}_{z,k}^-$, the corresponding column of $\boldsymbol{P}_{xz,k}^-$, and the corresponding row and column of $\boldsymbol{P}_{zz,k}^-$.

For some ways of performing the update, such as using the classical Joseph form of the covariance update, it may be preferable to remove the corresponding row of $\boldsymbol{H}_x(\boldsymbol{m}_{x,k}^-)$, the corresponding row of $\boldsymbol{H}_v(\boldsymbol{m}_{x,k}^-)$, and the corresponding row and column of $\boldsymbol{P}_{vv,k}$.

Once the edit has been made, the Kalman gain is computed, and the update is performed.

The user-specified editing tolerance, n_j , acts as an $n_j\sigma$ interval test for the innovation. Effectively, the test checks if the j^{th} element of the innovation falls within the $n_j\sigma$ interval described by the innovation covariance. If it does, that scalar part of the measurement is processed; if it does not, that scalar part of the measurement is completely excluded from consideration. Common values of n_j range anywhere from 3 to 9.

If the editing tolerance is exceeded, the measurement is assumed to have originated from a faulty measurement. Continued observation of the measurement editing test can serve as an indication of a possible failed sensor. This should not be confused with a simple disagreement due to the noise falling outside of the tolerance band. Single-point edits are not uncommon and are not even bad to see.

The scalar measurement editing method provides a good test for determining the statistical agreement of measurements (innovations) with their associated innovation covariance. However, if vector measurements are considered (magnetometer, velocimeter, star camera, etc.), then the scalar method ignores all cross-correlations in the innovation covariance and is capable of rejecting only a single axis of a vector measurement.

If a sensor failure exists, the entire vector measurement should be considered faulty and not simply one channel of the measurement. Moreover, to detect failure of a vector-measurement sensor, the correlation structure needs to be considered. To account for the cross-correlations, a vector method is developed that rejects the entire measurement vector if it is found to be in disagreement with the innovation covariance.

Given a random vector \mathbf{z} with mean \mathbf{m}_z and covariance \mathbf{P}_{zz} , the squared Mahalanobis distance is defined as

$$d^2 = (\mathbf{z} - \mathbf{m}_z)^T \mathbf{P}_{zz}^{-1} (\mathbf{z} - \mathbf{m}_z)$$

If \mathbf{z} is Gaussian-distributed, it can be shown that the squared Mahalanobis distance satisfies a χ^2 distribution with p degrees of freedom, where p is the dimension of \mathbf{z} .

Specification of a probability gate (P_G) along with the number of degrees of freedom can be used to look up a χ^2 value from a table, such as (degrees of freedom on the left and probabilities along the top)

		Probability Gate, P_G								
		75%	90%	95%	99%	99.5%	99.75%	99.9%	99.95%	99.99%
Dof, p	1	1.32	2.71	3.84	6.63	7.88	9.14	10.83	12.12	15.14
	2	2.77	4.61	5.99	9.21	10.60	11.98	13.82	15.20	18.42
	3	4.11	6.25	7.81	11.34	12.84	14.32	16.27	17.73	21.11
	4	5.39	7.78	9.49	13.28	14.86	16.42	18.47	20.00	23.51

The agreement test is therefore given as: if

$$d^2 \leq \gamma(p, P_G)$$

then \mathbf{z} is in agreement with \mathbf{m}_z and \mathbf{P}_{zz} , where $\gamma(p, P_G)$ represents the χ^2 table lookup.

If the dimension of the test vector is one and if the test vector is zero-mean, then the Mahalanobis distance test is given more succinctly by

$$d^2 = \frac{z^2}{\sigma_z^2} \leq \gamma \quad \text{or} \quad |z| \leq \sqrt{\gamma} \sigma_z$$

This resembles the scalar editing test that we previously developed, where we expressed the test as

$$|z| \leq n\sigma_z$$

and so, in this case, n can be associated with $\sqrt{\gamma}$. Referencing the χ^2 lookup table for one degree of freedom and a 99.75% probability, it is seen that $\gamma = 9.14$, such that $\sqrt{\gamma} = 3.02$, which is in agreement with the 3σ interval Gaussian probability of 99.73%.

Returning to the issue of measurement editing, it was previously shown that the innovation is zero mean, and therefore the Mahalanobis distance for residuals is found as

$$d^2 = (\mathbf{e}_{z,k}^-)^T (\mathbf{P}_{zz,k}^-)^{-1} \mathbf{e}_{z,k}^-$$

The editing process is performed by specifying a probability gate, looking up the associated χ^2 value for γ , and if

$$(\mathbf{e}_{z,k}^-)^T (\mathbf{P}_{zz,k}^-)^{-1} \mathbf{e}_{z,k}^- > \gamma(m, P_G)$$

then the measurement is rejected for inclusion into the filter. This works for the entire measurement, but it might be more useful if we have a method that works on a sensor-by-sensor basis.

Given q sensor returns at time t_k of the form $\mathbf{z}_k^{(i)}$ and associated measurement noise covariances $\mathbf{P}_{vv,k}^{(i)}$, first compute the associated estimated measurements,

$$\mathbf{m}_{z,k}^{(i)-} = \mathbf{h}^{(i)}(\mathbf{m}_{x,k}^-, \mathbf{0}_v)$$

and then form the measurement innovations as

$$\boldsymbol{e}_{z,k}^{(i)-} = \boldsymbol{z}_k^{(i)} - \boldsymbol{m}_{z,k}^{(i)-}$$

Compute the associated measurement Jacobians, $\boldsymbol{H}_x^{(i)}(\boldsymbol{m}_{x,k}^-)$ and $\boldsymbol{H}_v^{(i)}(\boldsymbol{m}_{x,k}^-)$, and determine the innovation covariances as

$$\boldsymbol{P}_{zz,k}^{(i)-} = \boldsymbol{H}_x^{(i)}(\boldsymbol{m}_{x,k}^-) \boldsymbol{P}_{xx,k}^- (\boldsymbol{H}_x^{(i)}(\boldsymbol{m}_{x,k}^-))^T + \boldsymbol{H}_v^{(i)}(\boldsymbol{m}_{x,k}^-) \boldsymbol{P}_{vv,k}^{(i)} (\boldsymbol{H}_v^{(i)}(\boldsymbol{m}_{x,k}^-))^T$$

Choose $P_{G,i}$ to be a probability gate for the activation of residual editing, and look up the associated χ^2 value ($\gamma_i(q_i, P_{G,i})$) from the table.

If

$$(\boldsymbol{e}_{z,k}^{(i)-})^T (\boldsymbol{P}_{zz,k}^{(i)-})^{-1} \boldsymbol{e}_{z,k}^{(i)-} > \gamma_i$$

then eliminate the i^{th} vector measurement and associated cross and innovation covariance elements from being included in the estimation process. This can also be accomplished through removing the corresponding elements of the measurement Jacobians.

This test means that the innovation has exceeded the allotted probability gate and is assumed to have originated from a faulty measurement. Continued observation of the innovations and their failing of the above test serves as an indication of a possible failed sensor. As with the scalar editing method, this should not be confused with a simple disagreement due to the noise falling outside of the tolerance band.

4.1.3 Underweighting

In some cases, we observe divergent filter behaviors in the extended Kalman filter. Divergent behavior in this sense refers to a filter whose estimation error is inconsistent with its confidence in the estimation error. Typically, the estimation error covariance becomes small, and the estimation error becomes large.

One case in which this commonly occurs is when a precise measurement is combined with large prior uncertainties. When the measurement is processed, a “large” update occurs. The estimation error covariance “snaps down,” even though the estimation error might not similarly decrease. One of the main mechanisms behind this occurring is the use of linearization to formulate the EKF update.

Underweighting is the act of adjusting the Kalman gain to “soften” the update in an effort to avoid over-convergence, which often results in filter divergence, of the extended Kalman filter. The resulting gain still belongs to the class of linear gains, but it is no longer the optimal (or approximately optimal) Kalman gain. The idea is to provide the filter more time to progressively converge.

The general update equations used to incorporate measurements in the EKF are

$$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k [\mathbf{z}_k - \mathbf{m}_{z,k}^-]$$

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- \mathbf{K}_k^T - \mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

We have previously developed alternate covariance update relationships. If we apply the Kalman gain to the covariance update, we can show that

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

From this form of the covariance update, it is seen that lowering the Kalman gain leads to a smaller update in the state estimation error covariance. We do have to be somewhat careful here, however. Once we move away from the Kalman gain, the preceding relationship no longer holds. This relationship is simply meant to guide our intuition that

lowering the Kalman gain should lead to a slow down the convergence of the filter and help prevent over-convergence of the filter.

When we look at the update equation for the state estimate, we see that \mathbf{K}_k (whether it is the Kalman gain or some other linear gain) controls how much new information is injected into the state estimate. Therefore, again, we see that a “smaller” gain should incorporate less information and should lead to slower convergence of the estimator.

To make the gain smaller, we start from the Kalman gain and try to increase the “denominator.” For linear or linearized measurements, the Kalman gain is given by

$$\mathbf{K}_k = \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T \left[\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T \right]^{-1}$$

Note that we have suppressed the dependence of the measurement Jacobians on the mean for ease of notation; that is, $\mathbf{H}_{x,k}$ is shorthand for $\mathbf{H}_{x,k}(\mathbf{m}_{x,k}^-)$ and similarly for $\mathbf{H}_{v,k}$.

To decrease the gain, let the Kalman gain be augmented with an underweighting factor

$\mathbf{P}_{\text{uw},k}$ as

$$\mathbf{K}_k = \mathbf{P}_{xx,k}^{-} \mathbf{H}_{x,k}^T \left[\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{-} \mathbf{H}_{x,k}^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T + \mathbf{P}_{\text{uw},k} \right]^{-1}$$

If $\mathbf{P}_{\text{uw},k}$ is symmetric and positive-definite, the “denominator” of the Kalman gain is increased, thereby lowering the applied gain. It is important to note that we no longer have the Kalman gain as soon as we include this modification, which means that we must use one of the Joseph formula versions of the covariance update once we modify the gain.

The underweighting factor could be computed by considering second-order effects in the innovation covariance. If the state distribution is assumed to be Gaussian, then the second-order effects can be computed analytically and the second-order Kalman filter is (partially) born.

We can also make an ad-hoc selection for the underweighting term. The ad-hoc selection of the underweighting factor is taken to be a scaled form of the state uncertainty mapped into the measurement uncertainty, such that, for $0 < p < 1$,

$$\mathbf{P}_{\text{uw},k} = \frac{1-p}{p} \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{-} \mathbf{H}_{x,k}^T$$

If this underweighting factor is substituted into our modified Kalman gain, we find that

$$\begin{aligned}\mathbf{K}_k &= \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T \left[\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T + \frac{1-p}{p} \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T \right]^{-1} \\ &= \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T \left[\frac{1}{p} \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T \right]^{-1}\end{aligned}$$

Note that as $p \rightarrow 1$, the underweighted gain approaches the usual Kalman gain and that as $p \rightarrow 0$, $\mathbf{K}_k \rightarrow \mathbf{0}$. These two cases are the cases when the system is such that it can accommodate almost the full update ($p \rightarrow 1$) and when the system is such that it can accommodate almost no update ($p \rightarrow 0$).

With the underweighted gain available, it is desired to determine when to implement an underweighted Kalman gain instead of the optimal Kalman gain. We know that we want to have a test to detect when the projection of the state uncertainty into the measurement domain is large compared to the measurement noise uncertainty. Therefore, an ad-hoc, but useful condition is that underweighting should be applied whenever

$$\|\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T\| > \frac{p}{1-p} \|\mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T\|$$

How should we interpret the parameter p ? This parameter is interpreted as the acceptable percentage decrease in the state-to-measurement conversion of uncertainty across a single update. Consider, for example, the case when the acceptable percentage decrease in $\mathbf{H}_{x,k} \mathbf{P}_{xx,k} \mathbf{H}_{x,k}^T$ is selected as $p = 5/6$. In this case, underweighting is applied if

$$\|\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T\| \geq 5 \|\mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T\|$$

and the underweighted gain is

$$\mathbf{K}_k = \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T \left[1.2 \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T \right]^{-1}$$

The factor 1.2 is also the underweighting factor used in the Shuttle navigation system².

It was previously shown that if the norm condition is met for $0 < p < 1$, then the gain applied in the update is altered, with respect to the Kalman gain. Alternatively,

²Kriegsman, B.A. and Tao, Y., "Shuttle Navigation System for Entry and Landing Mission Phases," *Journal of Spacecraft and Rockets*, Vol. 12, No. 4, April 1975, pp. 213–219.

the underweighted gain can be written as

$$\mathbf{K}_k = \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T \left[\left[\sqrt{(1/p)} \mathbf{H}_{x,k} \right] \mathbf{P}_{xx,k}^- \left[\sqrt{(1/p)} \mathbf{H}_{x,k} \right]^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T \right]^{-1}$$

which has the interpretation that underweighting increases the measurement sensitivity matrix by a factor $\sqrt{(1/p)}$. Since $(1/p) > 1$, it is guaranteed that when underweighting is applied, it will increase the elements of the measurement sensitivity matrix. As such, the mapping of state estimation error covariance to the measurement domain is inflated to account for the increase in sensitivity.

In the situation in which multiple measurements from different sensors are being processed, it may be that different values of p are selected for each sensor.

To demonstrate how underweighting works in the multi-sensor case, consider the simultaneous processing of two measurements, $\mathbf{z}_k^{(1)}$ and $\mathbf{z}_k^{(2)}$, which are combined together as

$$\mathbf{z}_k = \begin{bmatrix} \mathbf{z}_k^{(1)} \\ \mathbf{z}_k^{(2)} \end{bmatrix}$$

In this situation, the measurement Jacobian matrices and the measurement noise covariance can also be broken into their constituent elements, yielding

$$\mathbf{H}_{x,k} = \begin{bmatrix} \mathbf{H}_{x,k}^{(1)} \\ \mathbf{H}_{x,k}^{(2)} \end{bmatrix}, \quad \mathbf{H}_{v,k} = \begin{bmatrix} \mathbf{H}_{v,k}^{(1)} \\ \mathbf{H}_{v,k}^{(2)} \end{bmatrix}, \quad \text{and} \quad \mathbf{P}_{vv,k} = \begin{bmatrix} \mathbf{P}_{vv,k}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_{vv,k}^{(2)} \end{bmatrix}$$

where we have assumed that the two sensors' measurements noises are uncorrelated.

Letting p_1 be the (user-specified) acceptable percentage decrease in $\mathbf{H}_{x,k}^{(1)} \mathbf{P}_k (\mathbf{H}_{x,k}^{(1)})^T$ and similarly for p_2 , the underweighting condition becomes a set of two condition tests as

$$\|\mathbf{H}_{x,k}^{(1)} \mathbf{P}_{xx,k}^- (\mathbf{H}_{x,k}^{(1)})^T\| \geq \frac{p_1}{1 - p_1} \|\mathbf{H}_{v,k}^{(1)} \mathbf{P}_{vv,k}^{(1)} (\mathbf{H}_{v,k}^{(1)})^T\|$$

$$\|\mathbf{H}_{x,k}^{(2)} \mathbf{P}_{xx,k}^- (\mathbf{H}_{x,k}^{(2)})^T\| \geq \frac{p_2}{1 - p_2} \|\mathbf{H}_{v,k}^{(2)} \mathbf{P}_{vv,k}^{(2)} (\mathbf{H}_{v,k}^{(2)})^T\|$$

Each test is done independently and may have different outcomes. When the condition is met for the i^{th} sensor, underweighting is applied to only the i^{th} sensor. Using the sensitivity scaling interpretation, this is implemented using the scaled measurement Jacobian

as

$$\bar{\mathbf{H}}_{x,k}^{(i)} = \begin{cases} \sqrt{(1/p_i)} \mathbf{H}_{x,k}^{(i)} & \text{if underweighting is applied} \\ \mathbf{H}_{x,k}^{(i)} & \text{otherwise} \end{cases}$$

leads to an underweighted gain of the form

$$\mathbf{K}_k = \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T \left[\bar{\mathbf{H}}_{x,k} \mathbf{P}_{xx,k}^- \bar{\mathbf{H}}_{x,k}^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T \right]^{-1}$$

where $\bar{\mathbf{H}}_{x,k}$ is concatenated using $\bar{\mathbf{H}}_{x,k}^{(i)}$ in the same way that $\mathbf{H}_{x,k}$ is concatenated using $\mathbf{H}_{x,k}^{(i)}$. It is also worth noting that the alterations to the measurement Jacobians (with respect to the state) only occur in the “denominator” of the gain. The other appearance of $\mathbf{H}_{x,k}$ is left unchanged.

This process is easily extended to a set of q vector measurements received and processed at the same time. Each measurement is tested to determine if underweighting is applied, the Jacobians are altered if it is, and the rest of the formulation follows as presented.

Both measurement editing and underweighting affect the computation of the state estimate and state estimation error covariance update by testing conditions on the innovation covariance. We therefore need to take a look at how these two processes are used together. The net effect is that underweighting is applied first, and then the measurement editing process is applied. This is done because the underweighting process has the ability to inflate the innovation covariance when we believe that there may be additional effects that need to be captured. This is then the innovation covariance that we use in processing the data, and so it is the innovation covariance that we use in testing the data for inclusion or rejection.

For simplicity's sake, consider a single measurement, \mathbf{z}_k , that is of dimension q . Let the user-defined values for underweighting and measurement editing be p and P_G , respectively. The combined underweighting and measurement editing processes are included in the update as

1. expected measurement: $\mathbf{m}_{z,k}^- = \mathbf{h}(\mathbf{m}_{x,k}^-, \mathbf{0}_v)$

2. innovation: $\mathbf{e}_{z,k}^- = \mathbf{z}_k - \mathbf{m}_{z,k}^-$

3. cross-covariance: $\mathbf{P}_{xz,k}^- = \mathbf{P}_{xx,k}^- \mathbf{H}_x^T(\mathbf{m}_{x,k}^-)$
4. state-mapped covariance: $\mathbf{P}_{zz,k}^{(x)-} = \mathbf{H}_x(\mathbf{m}_{x,k}^-) \mathbf{P}_{xx,k}^- \mathbf{H}_x^T(\mathbf{m}_{x,k}^-)$
5. noise-mapped covariance: $\mathbf{P}_{zz,k}^{(v)-} = \mathbf{H}_v(\mathbf{m}_{x,k}^-) \mathbf{P}_{vv,k} \mathbf{H}_v^T(\mathbf{m}_{x,k}^-)$
6. check underweighting: if $(1 - p) \|\mathbf{P}_{zz,k}^{(x)-}\| > p \|\mathbf{P}_{zz,k}^{(v)-}\|$, choose $s = 1/p$
otherwise, choose $s = 1$
7. underweighted innovation covariance: $\mathbf{P}_{zz,k}^- = s \mathbf{P}_{zz,k}^{(x)-} + \mathbf{P}_{zz,k}^{(v)-}$
8. check residual editing: if $(\mathbf{e}_{z,k}^-)^T (\mathbf{P}_{zz,k}^-)^{-1} \mathbf{e}_{z,k}^- > \gamma(q, P_G)$, stop
9. compute the gain: $\mathbf{K}_k = \mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1}$

10. update the state estimate: $\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{m}_{z,k}^-)$

11. update the covariance: $\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^-\mathbf{K}_k^T - \mathbf{K}_k(\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k\mathbf{P}_{zz,k}^-\mathbf{K}_k^T$

It is important to keep in mind that one of the Joseph forms of the covariance update must be used when underweighting is applied.

Example: Altimeter Measurements

Let's take a look at a problem where we apply both underweighting and measurement editing.

This problem is going to be a case of an object in orbit that is able to take altitude measurements. As such, both the dynamics and the measurements are nonlinear.

We'll consider the dynamical system

$$\dot{\mathbf{r}} = \mathbf{v}$$

$$\dot{\mathbf{v}} = -(\mu/\|\mathbf{r}\|^3)\mathbf{r}$$

which is a continuous-time nonlinear dynamical system with no process noise. This does not prevent us from implementing a $\mathbf{Q}_{ww}(t)$ term in our filter, should we deem it necessary. We will use a state that is given by

$$\mathbf{x}^T = [\mathbf{r}^T \ \mathbf{v}^T]$$

and our dynamics give us the form for $\mathbf{f}(\mathbf{x}(t), \mathbf{w}(t))$, from which we can develop the dynamics Jacobians required in the EKF.

Given R_e to be the radius of the Earth, the initial mean position and velocity of the object are taken such that the object is in a circulate orbit of altitude h_0 , or

$$\mathbf{m}_{x,0} = \begin{bmatrix} R_e + h_0 \\ 0 \\ 0 \\ 0 \\ \sqrt{(\mu/(R_e + h_0))} \\ 0 \end{bmatrix} \quad \text{where} \quad h_0 = 1 \times 10^6 \text{ m}$$

Additionally, we take the initial position uncertainty (1σ) to be 100 m and 1 m/s in position and velocity, respectively, which gives

$$\mathbf{P}_{xx,0} = \text{diag}\left\{\begin{bmatrix}100^2 & 100^2 & 100^2 & 1^2 & 1^2 & 1^2\end{bmatrix}\right\}$$

For the measurements, we assume that measurements of altitude are generated according to the model

$$z_k = \|\mathbf{r}_k\| - R_e + v_k$$

where v_k is zero-mean and white with constant covariance $P_{vv} = (0.1 \text{ m})^2$. These measurements in conjunction with the defined states allow us to define $\mathbf{h}(\mathbf{x}_k, \mathbf{v}_k)$, from which we can develop the measurement Jacobians required in the EKF.

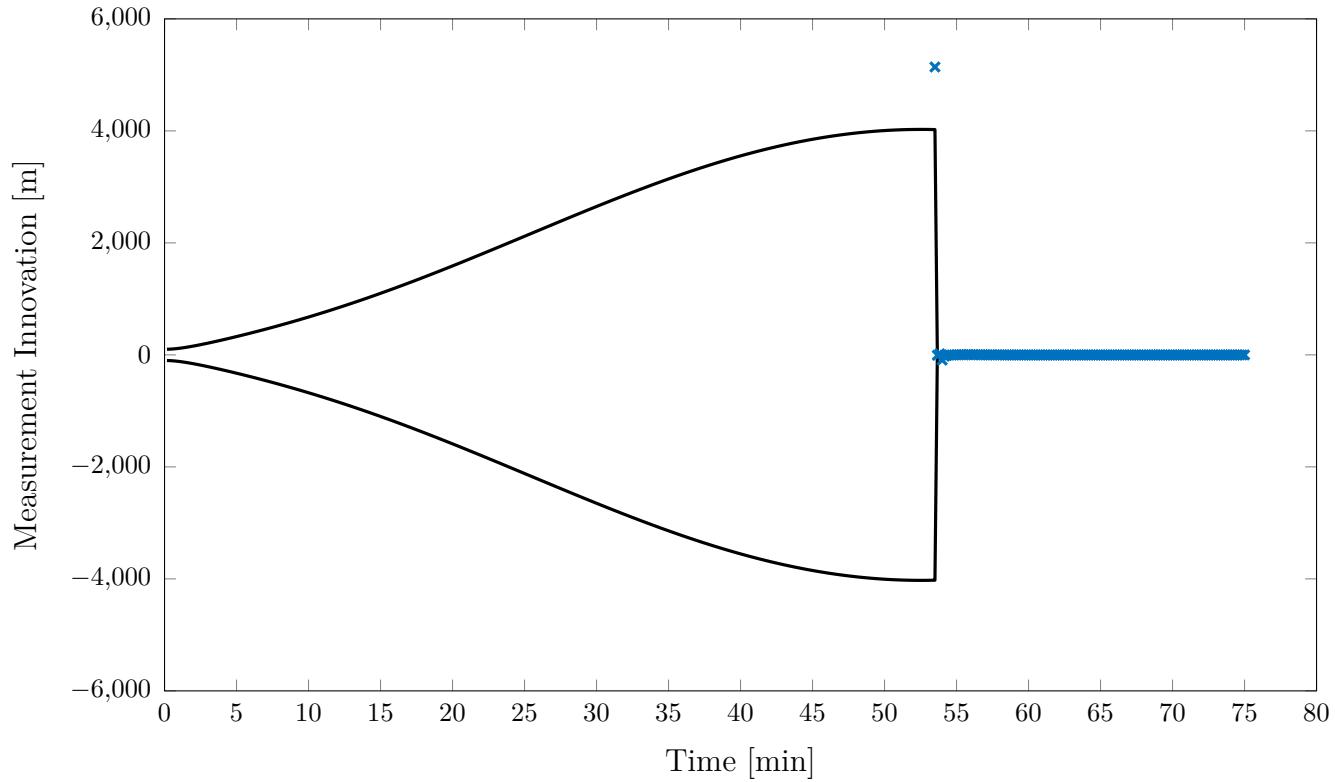
We will make a few adjustments to add some challenges to the problem:

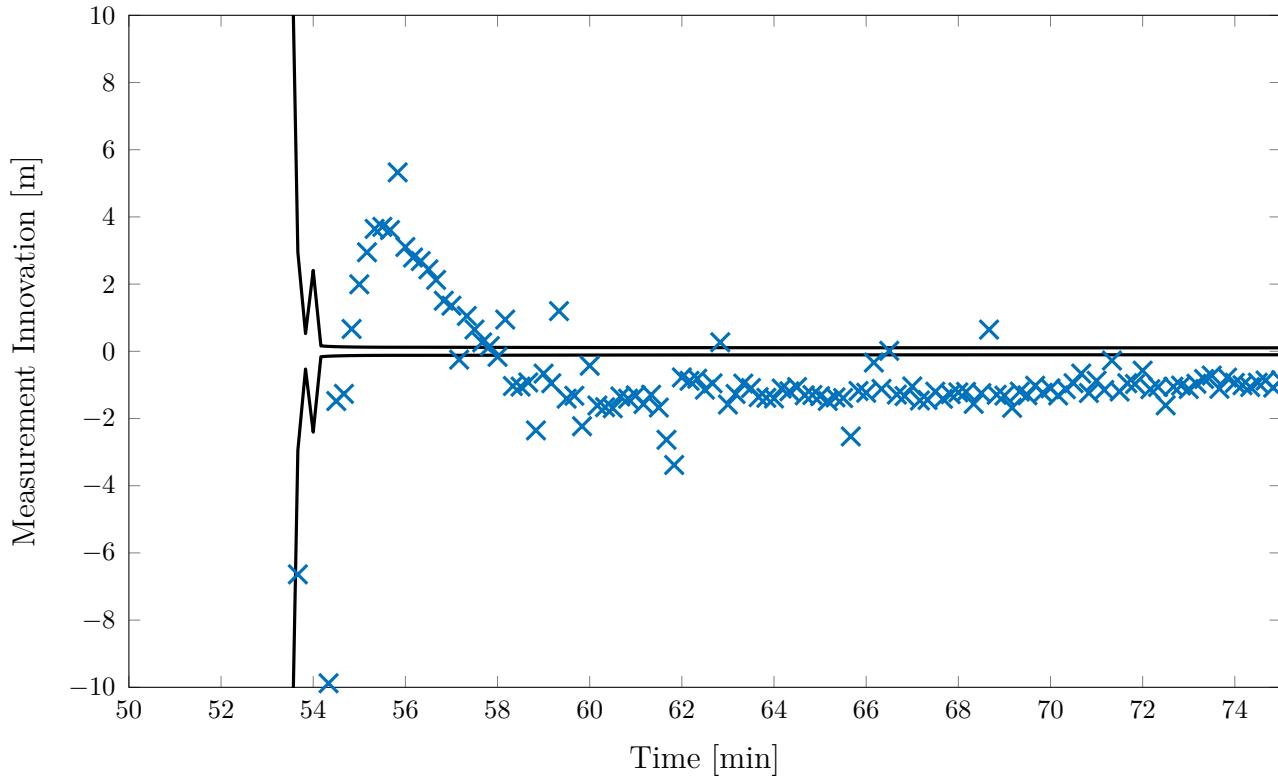
- We only generate measurements after some time has elapsed, which allows our state uncertainties to develop correlations and to grow somewhat. Therefore, measurements are generated once every 10 seconds after 3200 seconds have elapsed.
- We also want to throw in some bad data to see if we can effectively edit out the bad data. To do this, we generate data in accordance with the assumed model 80% of

the time. The other 20% of the time, we use the same model, but we use 10 times the measurement noise standard deviation to generate the data. The filter is not made aware of this change to the measurement noise statistics.

We use all of this information to develop and implement an EKF.

Let's first take a look at some results if underweighting and measurement editing are not applied. We will focus on the results from the perspective of the data processing.





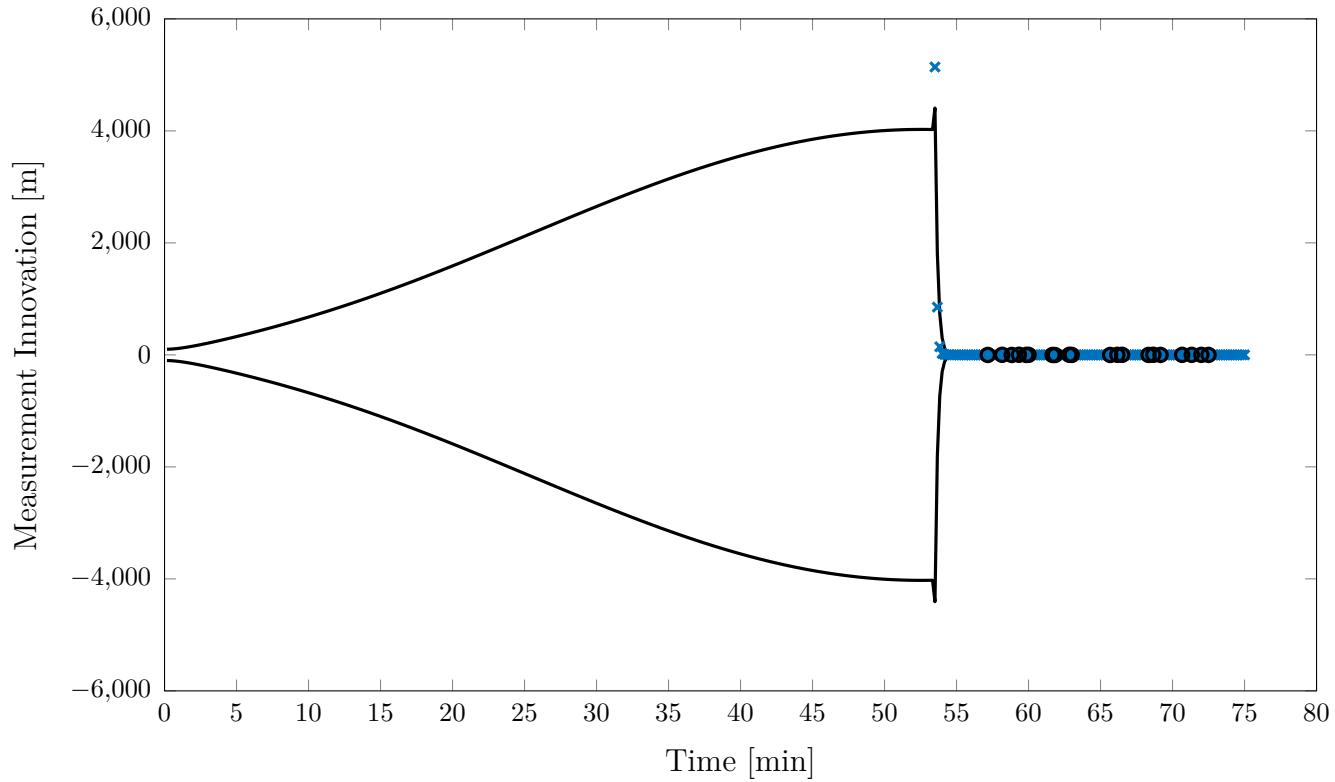
The first of these two figures shows the innovations and innovations covariance (as a 3σ interval) for the entirety of the simulation.

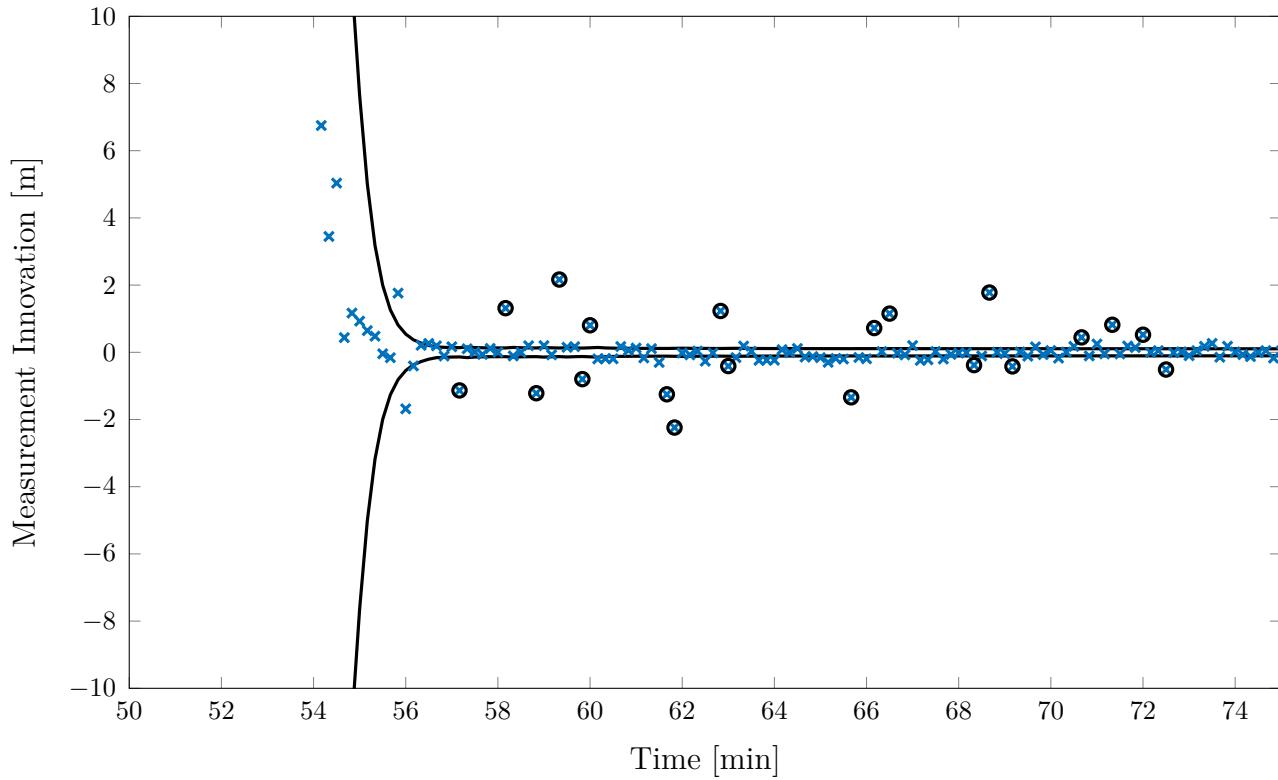
It is worth noting that even though measurement data is not processed until $t > 3200$ sec, we can still compute the innovation covariance. This is what is being shown in the early period of the first figure when there are no innovations to overlay.

The second figure focuses on the time period when measurement data is received and processed by the filter. We see a “snap down” of the innovation covariance when the first measurement is processed. This is the filter attempting to extract all of the information about the altitude of this vehicle at once.

We also see that the innovations fail to agree with the filter’s prediction of the uncertainty in the innovations. We see a bit of transient behavior before the innovations settle into a mostly biased solution.

Let’s see if we can improve this with underweighting ($p = 5/6$) and measurement editing ($p = 1$ and $P_G = 0.9995$).



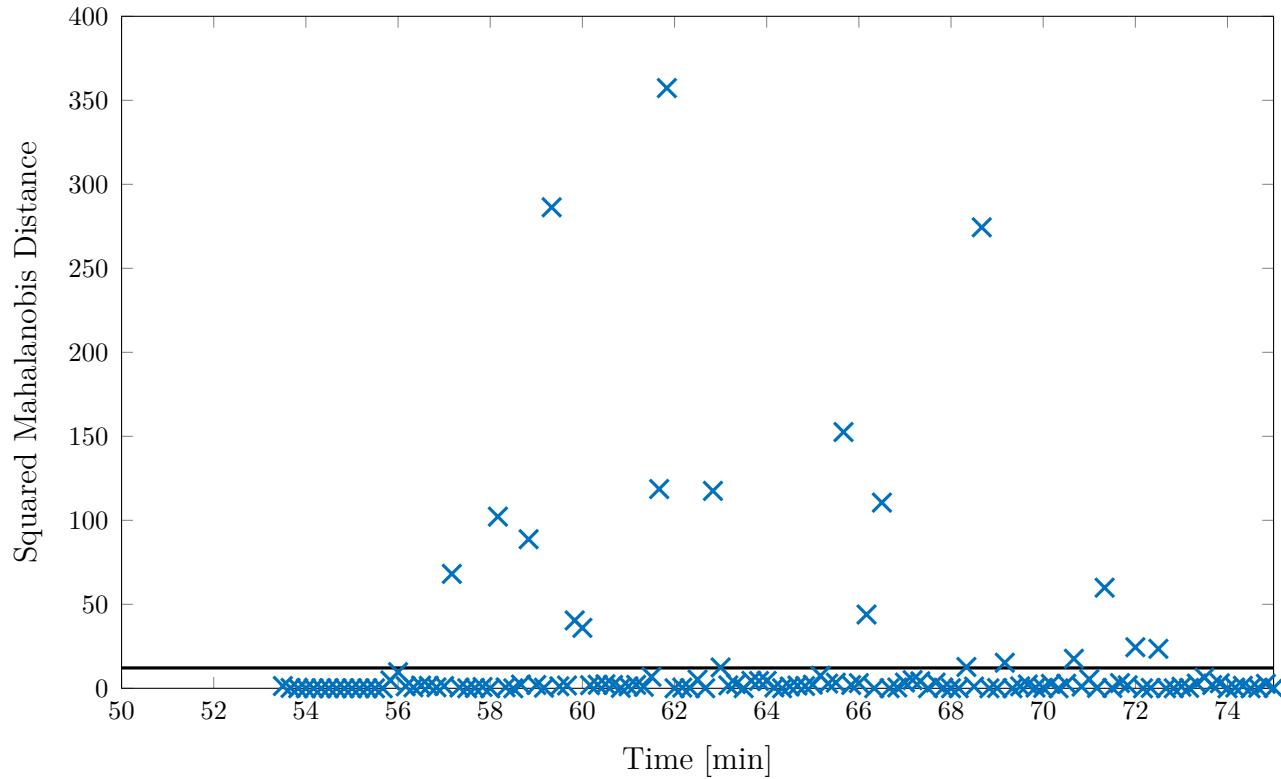


This set of two figures replicates the first set of two figures, except with underweighting and measurement underweight operating within the filter. Measurements that are edited out are still illustrated, but they are circled so that we can tell what data was not accepted by the filter for processing.

Looking at the full time interval, we see a difference in the innovation covariance matrix from the non-underweighting/editing case when measurements are first acquired. The innovation covariance has a spike upwards; this is the effect of underweighting kicking in.

Focusing on the interval during which data is processed, we notice that the innovation covariance exhibits a smoother and slower convergence compared to when underweighting was not employed. This gradual convergence allows the filter to properly process the data, and we notice that the data is much more in agreement with the 3σ interval now.

As expected, there are some outliers in the data, and these are successfully detected by the measurement editing process; as such, these data are not included in the filtering solution. Another way to look at this is via the squared Mahalanobis distance, which is illustrated in the next figure. Here, we see the editing gate and the measurements that fall outside of region allowed for inclusion into the filter.



4.1.4 Monte Carlo Analysis

When we have a truly linear system, we know that the Kalman filter produces a minimum mean-square error estimate of the state of the system. When we have nonlinear systems, we know that there are some approximations and assumptions made in developing the extended Kalman filter. We also know that the outputs of the filter become trajectory or path dependent.

We have developed some validation techniques to ensure that our functions and our linearizations are operating correctly. We have also developed some alterations to ensure that we prevent over-convergence and that we prevent bad data processing. With all of this going on, how can we tell if our filter is really working?

Fundamentally, we have two quantities available to us for assessment/analysis: mean (or the estimated state/measurement) and covariance. When we are operating a filter on real data, we generally get a single sequence of data and we have no access to the real true states. On the other hand, when we are operating a filter in simulation, we can generate as many sequences of data for which we have patience, and we have access to the exact truth that we are simulating.

In simulation, we can perform as many “trials” of the simulation as we want to. Each time we run a trial, a new truth is generated. New measurements are generated. New filter results are generated. For each trial, we can use the truth and filter outputs to compute and record the estimation error and the measurement innovation. We can also record the estimation error covariance and the innovation covariance. When the system is nonlinear, the filter’s mean and covariance (for the state or for the innovation) will change from one simulation to the next.

We aim to develop a filter that produces unbiased estimates, both of the states and of the measurements. Therefore, we can collect the estimation error from each of these trials, and we can compute the average error. If the average error is zero (or at least near-zero), we say that the filter is unbiased. Otherwise, we say that the filter is biased. This holds for analysis of estimation error and innovations.

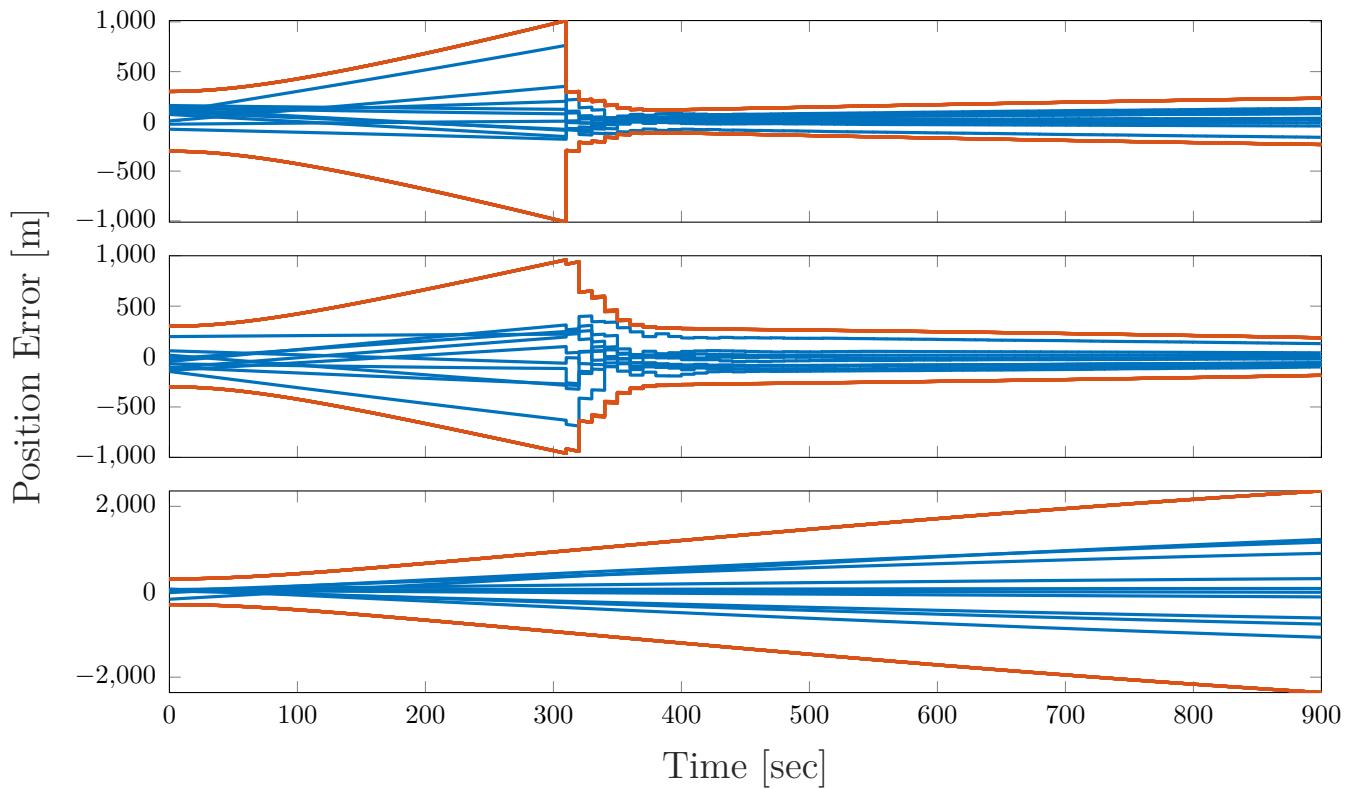
Since our estimation error covariance is supposed to described the covariance of the estimation error, we can compute the sample covariance from the collection of estimation errors. If the filter is operating correctly, the sample covariance should align with the average of the filters’ covariances. If the sample covariance is larger than the average

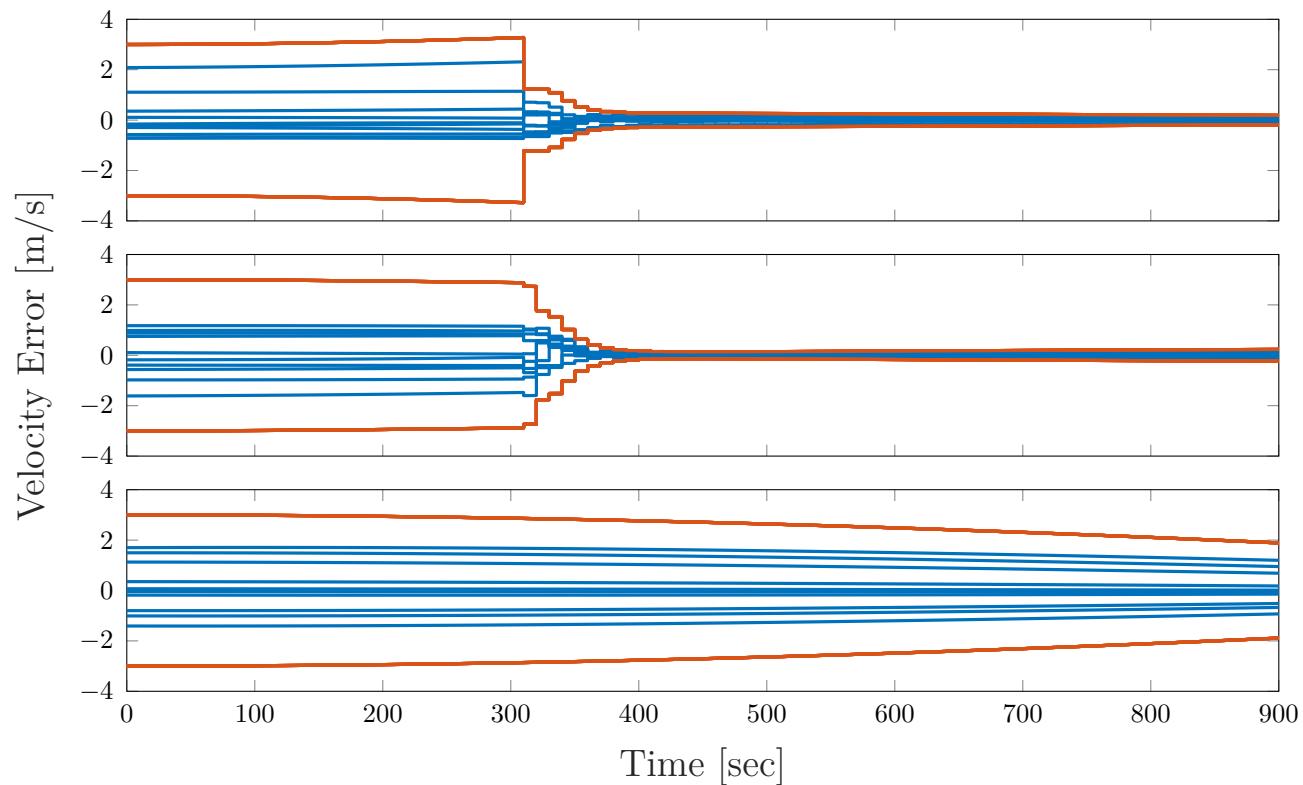
covariance, we say that the filter is smug or overconfident. If the sample covariance is smaller than the average covariance, we say that the filter is conservative. If the sample covariance agrees with the average covariance, we say that the filter is statistically consistent. This analysis also works for both the state and the innovation.

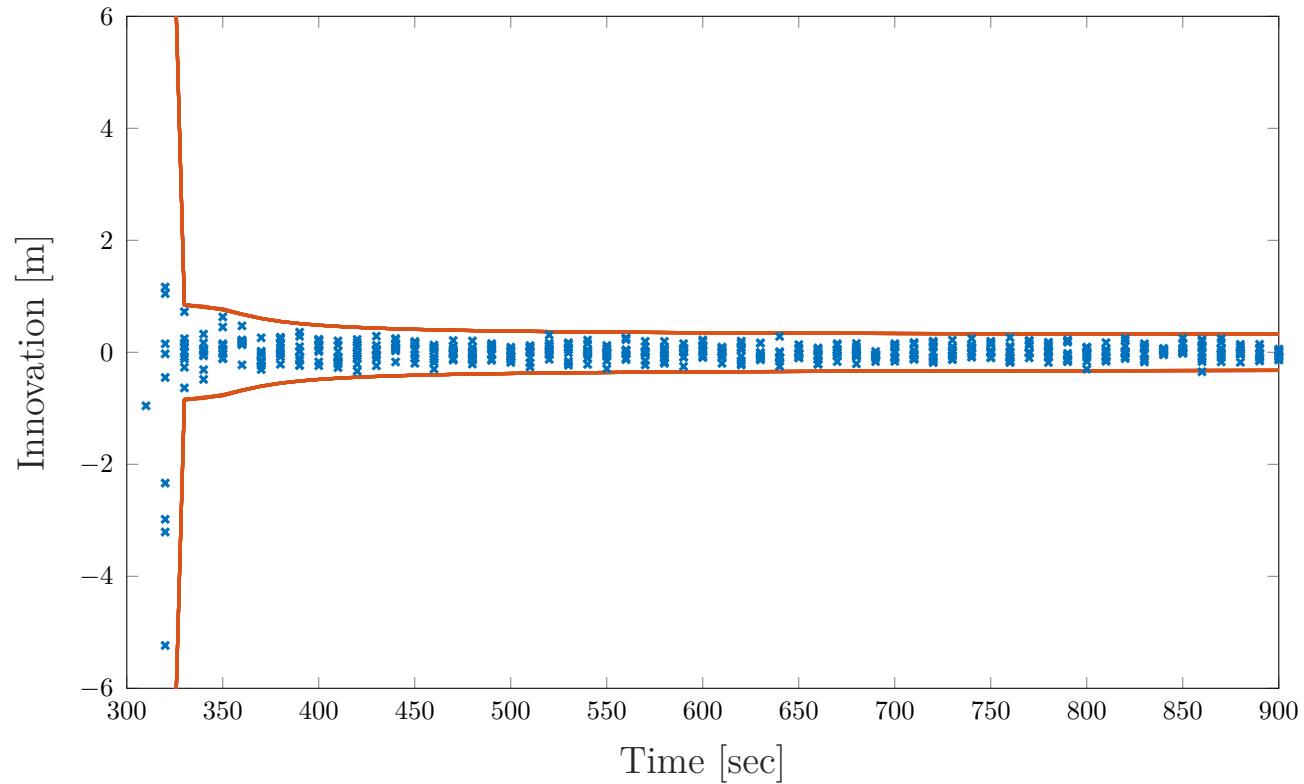
This process is referred to as Monte Carlo analysis.

Let's look at what we might expect from Monte Carlo analysis using a slightly simpler version of the altimeter problem that we investigated previously. This time, we won't wait as long to start taking measurements, and we won't implement underweighting or editing. These simplifications aren't required; they're just here to make the results easier to discuss. Otherwise, the configuration is the same.

We perform a set of 100 trials. For the first 10 trials, the estimation error and 3σ interval for the position and velocity states are provided. Additionally, the innovations and 3σ interval are also provided.





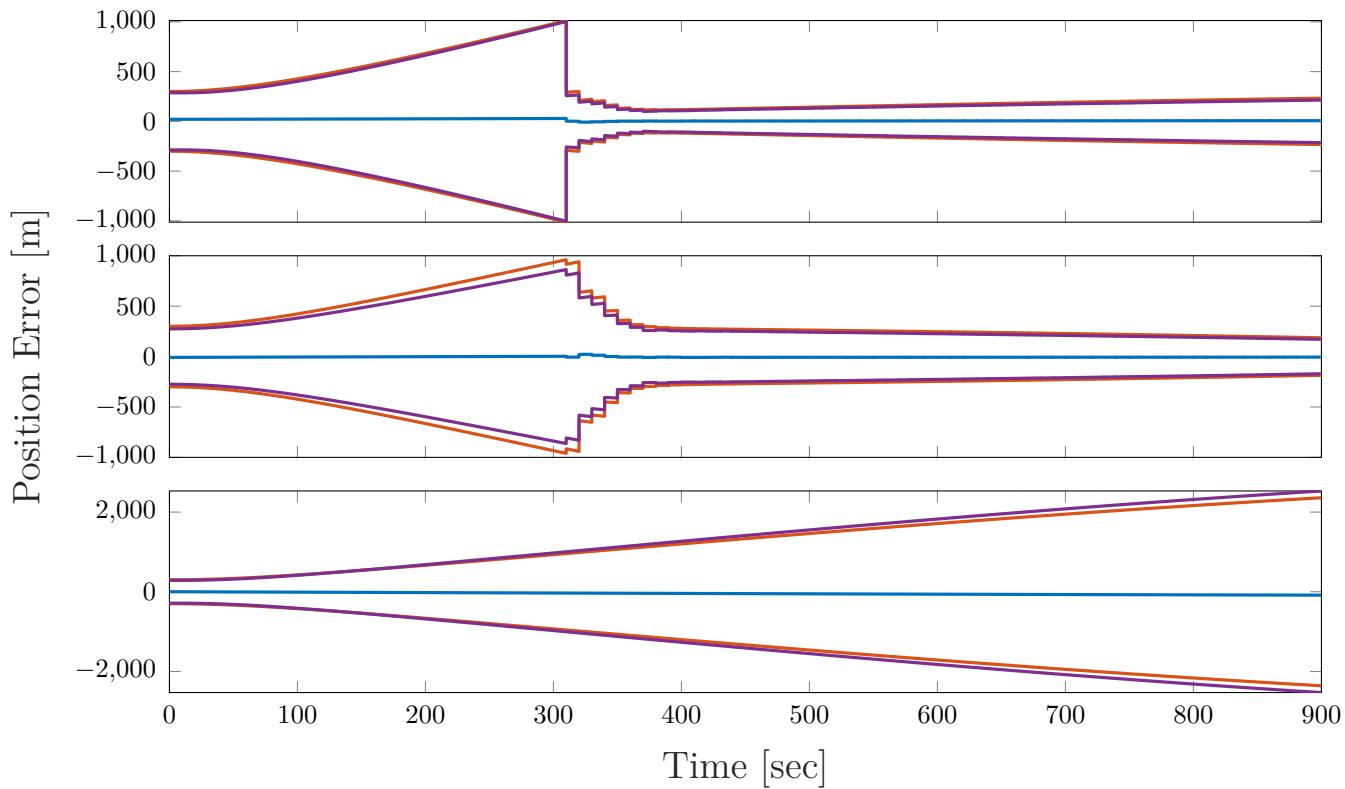


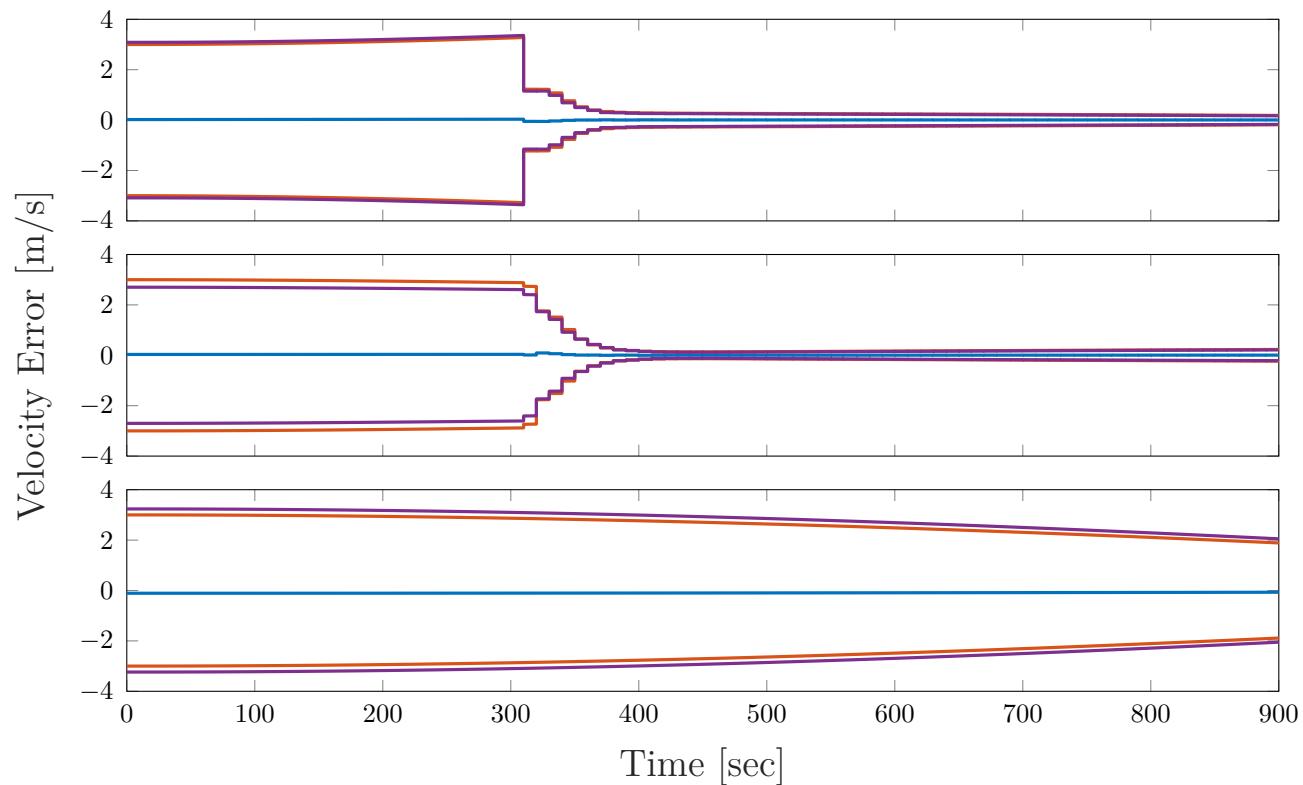
Even with just 10 trials plotted, the figures are starting to get crowded.

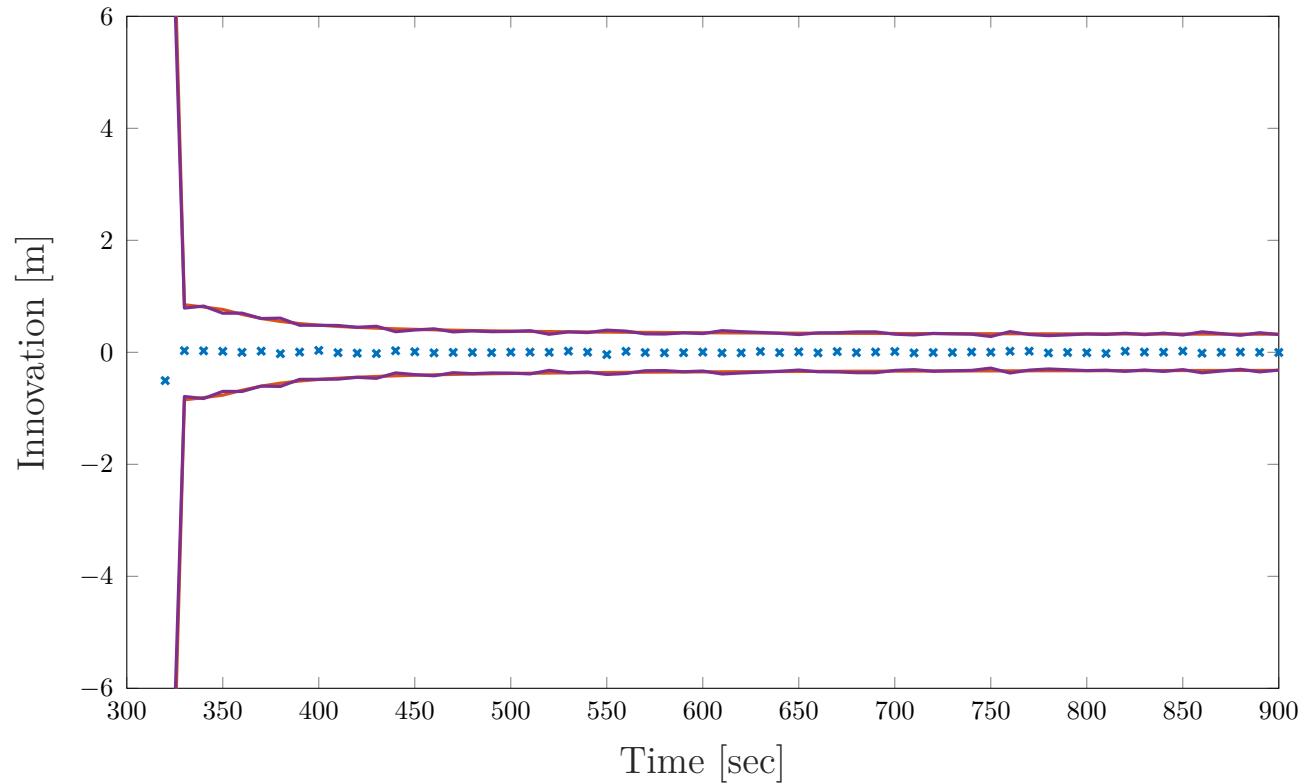
We notice that each filter appears to behave relatively well, as there are no trials that are observed to diverge. It is also clear that each trial has a very different estimation error profile and very different innovation profile. Visibly, 10 trials is not enough to properly sample the position and velocity space, but this is to be expected.

While the 3σ intervals may appear to be single curves, there are in fact 10 distinct curves here. Each filter exhibits very similar behavior in terms of the covariance, both for the estimation error covariance and the innovation covariance. This is a trend that is often, but not always, observed, and it is a good sign that things are working as expected.

The next figures shows the average error, average filter 3σ , and Monte Carlo 3σ . The average filter 3σ is computed by averaging the covariance matrices from all of the trials. The Monte Carlo 3σ is computed by finding the sample covariance of the estimation errors from all of the trials.







From these results, we see that the filter is unbiased and statistically consistent. This analysis only has 100 trials, so there is still some discrepancy between the 3σ intervals, but the major trends are captured. These discrepancies can be mitigated by using a larger number of trials, and more is pretty much always better, up to the point of waiting for results and requiring availability of computational resources.

4.2 The Information Filter

Thus far, we have really only relied upon mean and covariance for our Kalman filters, but we used some terms that we called λ and Λ when we worked with LUMVE. By working in these variables, we were able to consider cases where we had no prior information. A natural question is then: Does the Kalman filter also facilitate a similar approach? After all, we've seen quite a few parallels between the methods so far.

These parameters are often referred to as the “information state” (λ) and the “information matrix” (Λ), and when we reformulate the Kalman filter to work with these quantities, we have what is known as the information form of the Kalman filter, or simply the information filter. The principle idea is to replace the mean and covariance of

the Kalman filter with the information state and matrix. These two sets of parameters are related by

$$\boldsymbol{\lambda}_x = \mathbf{P}_{xx}^{-1} \mathbf{m}_x$$

$$\boldsymbol{\Lambda}_{xx} = \mathbf{P}_{xx}^{-1}$$

where \mathbf{m}_x and \mathbf{P}_{xx} are the mean and covariance that we are used to working with in various versions of the Kalman filter. These conversions can be applied at any instance in time and also prior to and after a measurement update.

To understand a use case for making this change, consider the scenario where we let the confidence in our mean (or estimate) go to zero. In terms of the covariance, this means that $\mathbf{P}_{xx} \rightarrow \infty$. While this is rather informal notation, the meaning is hopefully clear. When our confidence is very low, our uncertainty (covariance) is very large. Quantities that go to infinity are usually pretty difficult to work with. In this same scenario, the information matrix tends to a null matrix, i.e., $\boldsymbol{\Lambda}_{xx} \rightarrow \mathbf{0}$. While infinity might be difficult to work with computationally, zero is pretty easy to work with.

The overall idea is therefore to replace mean and covariance with the information state

and the information matrix. In the following, we will make this transformation for the discrete form of the Kalman filter, with the understanding that the conversion can be applied to other forms of the Kalman filter, as well. We will take the system to have the form

$$\boldsymbol{x}_k = \boldsymbol{F}_{x,k-1} \boldsymbol{x}_{k-1} + \boldsymbol{w}_{k-1}$$

$$\boldsymbol{z}_k = \boldsymbol{H}_{x,k} \boldsymbol{x}_k + \boldsymbol{v}_k$$

That is, the process and measurement noises directly add in with identity mapping matrices. Accommodating the more general case is possible, and the necessary alterations can be made by following the subsequent developments with those terms in place.

For the discrete Kalman filter, the propagation step is given by

$$\boldsymbol{m}_{x,k} = \boldsymbol{F}_{x,k-1} \boldsymbol{m}_{x,k-1}$$

$$\boldsymbol{P}_{xx,k} = \boldsymbol{F}_{x,k-1} \boldsymbol{P}_{xx,k-1} \boldsymbol{F}_{x,k-1}^T + \boldsymbol{P}_{ww,k-1}$$

Let's first take a look at the covariance propagation and convert it to and information matrix propagation. While we have set $\boldsymbol{F}_{w,k-1} = \boldsymbol{I}$, it is possible to make adjustments

for the more general case.

At the highest level, the propagated information matrix is given by the inverse of the propagated covariance matrix, i.e.,

$$\boldsymbol{\Lambda}_{xx,k} = [\mathbf{F}_{x,k-1} \mathbf{P}_{xx,k-1} \mathbf{F}_{x,k-1}^T + \mathbf{P}_{ww,k-1}]^{-1}$$

We can also make the substitution for the covariance matrix on the right-hand side in terms of the information matrix to find

$$\boldsymbol{\Lambda}_{xx,k} = [\mathbf{F}_{x,k-1} \boldsymbol{\Lambda}_{xx,k-1}^{-1} \mathbf{F}_{x,k-1}^T + \mathbf{P}_{ww,k-1}]^{-1}$$

While this is a valid result for the propagation, it is effectively requiring us to form a covariance matrix, propagate the covariance form, and then convert back to information form. This is not a desirable sequence, and we can avoid the conversions by applying the matrix inversion lemma.

First, define $\boldsymbol{\Xi}_{k-1}$ to be

$$\boldsymbol{\Xi}_{k-1} = [\mathbf{F}_{x,k-1} \boldsymbol{\Lambda}_{xx,k-1}^{-1} \mathbf{F}_{x,k-1}^T]^{-1}$$

and distribute the inverse to yield

$$\boldsymbol{\Xi}_{k-1} = \mathbf{F}_{x,k-1}^{-T} \boldsymbol{\Lambda}_{xx,k-1} \mathbf{F}_{x,k-1}^{-1}$$

This involves inverting the dynamics matrix, which is perhaps not always easy; however, in some cases, this inverse can be found *a priori*, which means that we do not have to compute the inverse online. Additionally, if the dynamics matrix is interpreted as a state transition matrix with two time arguments, then it is straightforward to compute the inverse via the fundamental properties of the state transition matrix.

With the definition of $\boldsymbol{\Xi}_{k-1}$, the propagated information matrix is

$$\boldsymbol{\Lambda}_{xx,k} = [\boldsymbol{\Xi}_{k-1}^{-1} + \mathbf{P}_{ww,k-1}]^{-1}$$

Now, we can apply the matrix inversion lemma to the preceding relationship to find that the propagated information matrix is

$$\boldsymbol{\Lambda}_{xx,k} = \boldsymbol{\Xi}_{k-1} - \boldsymbol{\Xi}_{k-1} [\boldsymbol{\Xi}_{k-1} + \mathbf{P}_{ww,k-1}^{-1}]^{-1} \boldsymbol{\Xi}_{k-1}$$

To provide some insight, let's consider two extreme cases:

No Noise If we have no process noise, i.e., $\mathbf{P}_{ww,k-1} = \mathbf{0}$, then the inverse of the process noise covariance becomes infinite and dominates $\boldsymbol{\Xi}_{k-1}$, such that the combined inverse is zero and the propagated information matrix is simply $\boldsymbol{\Xi}_{k-1}$.

All Noise If we have process noise with “infinite” covariance, then the inverse of the process noise covariance is $\mathbf{P}_{ww,k-1}^{-1} = \mathbf{0}$, and we find that the propagated information matrix is zero.

These are of course two extreme cases, but they give a little insight into the behavior of the information matrix, which is not always as intuitive as the covariance matrix.

We now have the propagation equation for the information matrix, but what about the information state? By definition, the information state at t_k is given by

$$\boldsymbol{\lambda}_{x,k} = \mathbf{P}_{xx,k}^{-1} \mathbf{m}_{x,k} = \boldsymbol{\Lambda}_{xx,k} \mathbf{m}_{x,k}$$

We already know the relationships for the two terms on the right-hand side; namely,

$$\boldsymbol{\Lambda}_{xx,k} = \boldsymbol{\Xi}_{k-1} - \boldsymbol{\Xi}_{k-1} [\boldsymbol{\Xi}_{k-1} + \mathbf{P}_{ww,k-1}^{-1}]^{-1} \boldsymbol{\Xi}_{k-1}$$

$$\mathbf{m}_{x,k} = \mathbf{F}_{x,k-1} \mathbf{m}_{x,k-1}$$

If we substitute these into our equation for the information state propagation, it follows that

$$\boldsymbol{\lambda}_{x,k} = \left[\mathbf{I} - \boldsymbol{\Xi}_{k-1} \left[\boldsymbol{\Xi}_{k-1} + \mathbf{P}_{ww,k-1}^{-1} \right]^{-1} \right] \boldsymbol{\Xi}_{k-1} \mathbf{F}_{x,k-1} \boldsymbol{\Lambda}_{xx,k-1}^{-1} \boldsymbol{\lambda}_{x,k-1}$$

Noting that the product $\boldsymbol{\Xi}_{k-1} \mathbf{F}_{x,k-1} \boldsymbol{\Lambda}_{xx,k-1}^{-1}$ is given by

$$\boldsymbol{\Xi}_{k-1} \mathbf{F}_{x,k-1} \boldsymbol{\Lambda}_{xx,k-1}^{-1} = \mathbf{F}_{x,k-1}^{-T} \boldsymbol{\Lambda}_{xx,k-1} \mathbf{F}_{x,k-1}^{-1} \mathbf{F}_{x,k-1} \boldsymbol{\Lambda}_{xx,k-1}^{-1} = \mathbf{F}_{x,k-1}^{-T}$$

it directly follows that the information state propagation is

$$\boldsymbol{\lambda}_{x,k} = \mathbf{F}_{x,k-1}^{-T} \boldsymbol{\lambda}_{x,k-1} - \boldsymbol{\Xi}_{k-1} \left[\boldsymbol{\Xi}_{k-1} + \mathbf{P}_{ww,k-1}^{-1} \right]^{-1} \mathbf{F}_{x,k-1}^{-T} \boldsymbol{\lambda}_{x,k-1}$$

We now have propagation equations for both the information matrix and the information state. As opposed to the mean and covariance propagation equations that depend upon $\mathbf{F}_{x,k-1}$ and $\mathbf{P}_{ww,k-1}$, the propagation equations for the information state and the information matrix depend upon $\mathbf{F}_{x,k-1}^{-1}$ and $\mathbf{P}_{ww,k-1}^{-1}$. In some cases, these terms can be computed offline, meaning that their inverses should be viewed as the inputs to the propagation stage, as opposed to interpreting the inverses as operations that occur in the operation of the propagation stage.

With the propagation stage sorted out, we now turn our attention to the update stage. As a reminder, the mean and covariance update equations for the Kalman filter can be written as

$$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k [\mathbf{z}_k - \mathbf{m}_{z,k}^-]$$

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{K}_k \mathbf{H}_k \mathbf{P}_{xx,k}^-$$

where we are making use of the Kalman gain to formulate the covariance update. If we substitute for the Kalman gain in the covariance update, it follows that

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T [\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k}]^{-1} \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^-$$

In our LUMVE developments, we applied the matrix inversion lemma to sequentialize the batch processor. In applying the matrix inversion lemma, we arrived at this preceding form for the covariance update. If we reverse the process, we can show that the covariance update is equivalent to

$$(\mathbf{P}_{xx,k}^+)^{-1} = (\mathbf{P}_{xx,k}^-)^{-1} + \mathbf{H}_{x,k}^T \mathbf{P}_{vv,k}^{-1} \mathbf{H}_{x,k}$$

or, in terms of information matrices,

$$\boldsymbol{\Lambda}_{xx,k}^+ = \boldsymbol{\Lambda}_{xx,k}^- + \mathbf{H}_{x,k}^T \mathbf{P}_{vv,k}^{-1} \mathbf{H}_{x,k}$$

This gives us a simple method for updating the information matrix with new information, but what about the mean? We know that the update equation for the state estimate holds for any linear gain, which includes the Kalman gain

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T [\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{P}_{vv,k}]^{-1}$$

We have previously shown that the Kalman gain can also be written in terms of the posterior estimation error covariance matrix as

$$\mathbf{K}_k = \mathbf{P}_{xx,k}^+ \mathbf{H}_{x,k}^T \mathbf{P}_{vv,k}^{-1}$$

which can also be expressed in terms of the posterior information matrix as

$$\mathbf{K}_k = (\boldsymbol{\Lambda}_{xx,k}^+)^{-1} \mathbf{H}_{x,k}^T \mathbf{P}_{vv,k}^{-1}$$

Let's take this expression for the Kalman gain to be the linear gain that we use in our update. Leveraging the state estimate update in conjunction with linear measurements

and the Kalman gain, we have

$$\boldsymbol{m}_{x,k}^+ = \boldsymbol{m}_{x,k}^- + (\Lambda_{xx,k}^+)^{-1} \mathbf{H}_{x,k}^T \mathbf{P}_{vv,k}^{-1} \boldsymbol{z}_k - (\Lambda_{xx,k}^+)^{-1} \mathbf{H}_{x,k}^T \mathbf{P}_{vv,k}^{-1} \mathbf{H}_{x,k} \boldsymbol{m}_{x,k}^-$$

We can now pre-multiply both sides by the posterior information matrix and collect like terms to find

$$\Lambda_{xx,k}^+ \boldsymbol{m}_k^+ = [\Lambda_{xx,k}^+ - \mathbf{H}_{x,k}^T \mathbf{P}_{vv,k}^{-1} \mathbf{H}_{x,k}] \boldsymbol{m}_{x,k}^- + \mathbf{H}_{x,k}^T \mathbf{P}_{vv,k}^{-1} \boldsymbol{z}_k$$

From the information matrix update, the matrix in square brackets is simply the prior information matrix, such that the update for linear measurements using the Kalman gain is

$$\Lambda_{xx,k}^+ \boldsymbol{m}_{x,k}^+ = \Lambda_{xx,k}^- \boldsymbol{m}_{x,k}^- + \mathbf{H}_{x,k}^T \mathbf{P}_{vv,k}^{-1} \boldsymbol{z}_k$$

or, making use of the information state,

$$\boldsymbol{\lambda}_{x,k}^+ = \boldsymbol{\lambda}_{x,k}^- + \mathbf{H}_{x,k}^T \mathbf{P}_{vv,k}^{-1} \boldsymbol{z}_k$$

Note that the measurement update depends upon $\mathbf{P}_{vv,k}^{-1}$, much as the propagation depends upon $\mathbf{P}_{ww,k-1}^{-1}$. In this sense, you can think of the inverse of the measurement noise

covariance matrix as the input to the update.

It is also interesting to note the similarities to the LUMVE data accumulation equations. In fact, with the information form of the update, if we have m sensors that are uncorrelated with one another, the information state/matrix update are given by

$$\boldsymbol{\lambda}_{x,k}^+ = \boldsymbol{\lambda}_{x,k}^- + \sum_{i=1}^m (\mathbf{H}_{x,k}^{(i)})^T (\mathbf{P}_{vv,k}^{(i)})^{-1} \mathbf{z}_k^{(i)}$$

$$\boldsymbol{\Lambda}_{xx,k}^+ = \boldsymbol{\Lambda}_{xx,k}^- + \sum_{i=1}^m (\mathbf{H}_{x,k}^{(i)})^T (\mathbf{P}_{vv,k}^{(i)})^{-1} \mathbf{H}_{x,k}^{(i)}$$

which has the same form as the data accumulation process we developed for batch processing. In this form, we clearly see that the “information” from each sensor is simply added into the prior “information.” This is beautifully simple.

In general, the Kalman filter propagation equations are simple and the update equations are complicated; the reverse is true for the information filter: the propagation equations are complicated and the update equations are simple.

To summarize, we put everything together in a single table

System Model	$\boldsymbol{x}_k = \boldsymbol{F}_{x,k-1}\boldsymbol{x}_{k-1} + \boldsymbol{F}_{w,k-1}\boldsymbol{w}_{k-1}$
Meas. Model	$\boldsymbol{z}_k = \boldsymbol{H}_{x,k}\boldsymbol{x}_k + \boldsymbol{H}_{v,k}\boldsymbol{v}_k$
Init. Cond.	$\boldsymbol{m}_{x,0} = \text{E}\{\boldsymbol{x}(t_0)\}$ $\boldsymbol{P}_{xx,0} = \text{E}\{(\boldsymbol{x}(t_0) - \boldsymbol{m}_{x,0})(\boldsymbol{x}(t_0) - \boldsymbol{m}_{x,0})^T\}$ $\boldsymbol{\lambda}_{x,0} = \boldsymbol{P}_{xx,0}^{-1}\boldsymbol{m}_{x,0}$ $\boldsymbol{\Lambda}_{xx,0} = \boldsymbol{P}_{xx,0}^{-1}$
Inf. State Prop.	$\boldsymbol{\lambda}_{x,k} = \boldsymbol{F}_{x,k-1}^{-T}\boldsymbol{\lambda}_{x,k-1} - \boldsymbol{\Xi}_{k-1}[\boldsymbol{\Xi}_{k-1} + \boldsymbol{\Lambda}_{ww,k-1}]^{-1}\boldsymbol{F}_{x,k-1}^{-T}\boldsymbol{\lambda}_{x,k-1}$
Inf. Matrix Prop.	$\boldsymbol{\Lambda}_{xx,k} = \boldsymbol{\Xi}_{k-1} - \boldsymbol{\Xi}_{k-1}[\boldsymbol{\Xi}_{k-1} + \boldsymbol{\Lambda}_{ww,k-1}]^{-1}\boldsymbol{\Xi}_{k-1}$
Def.	$\boldsymbol{\Xi}_{k-1} = \boldsymbol{F}_{x,k-1}^{-T}\boldsymbol{\Lambda}_{xx,k-1}\boldsymbol{F}_{x,k-1}^{-1}$
Inf. State Upd.	$\boldsymbol{\lambda}_{x,k}^+ = \boldsymbol{\lambda}_{x,k}^- + \boldsymbol{H}_{x,k}^T\boldsymbol{\Lambda}_{vv,k}\boldsymbol{z}_k$
Inf. Matrix Upd.	$\boldsymbol{\Lambda}_{xx,k}^+ = \boldsymbol{\Lambda}_{xx,k}^- + \boldsymbol{H}_{x,k}^T\boldsymbol{\Lambda}_{vv,k}\boldsymbol{H}_{x,k}$

where we have used $\boldsymbol{\Lambda}_{ww,k-1} = \boldsymbol{P}_{ww,k-1}^{-1}$ and $\boldsymbol{\Lambda}_{vv,k} = \boldsymbol{P}_{vv,k}^{-1}$ for the information matrices associated with the covariance matrices of the process noise and measurement noise, respectively.

It is possible to convert between covariance and information forms of the Kalman filters. This simply requires that the inverses exist. In practice, it is common to see the information form leveraged when data processing starts in the presence of no prior information. Once enough information has been acquired to make the information matrix invertible, we switch over the covariance form. This is not required, however. We could choose to stay in the information form, if desired.

4.3 Square Root Filters

By now, we have seen the way we expect a Kalman filter to behave in an estimation problem. It should come as no surprise, however, that there is sometimes a difference between *theory* and *practice* that demands development of new techniques to produce the desired results.

Soon after the Kalman filter began to be implemented on computers (think 1960s NASA, MIT Instrumentation Laboratory, the Apollo program), some unexpected issues appeared. The mean-square estimation errors resulting from the Kalman filter implemen-

tations were larger than the covariance matrix would seem to support as statistically appropriate, even on simulated data. Sometimes, the variances in the estimation errors were observed to take *negative values*, which is a theoretical impossibility. It turns out that computer roundoff errors seriously degrade the performance of Kalman filters.

Sometimes we find that the state estimation covariance matrix, something which is rigidly defined as being symmetric, becomes asymmetric due to numerical errors, and we can lose positive-definiteness of our covariance matrices as a result. This causes degradation of the algorithms and a large amount of numerical instability.

These numerical issues inspired the development of methods which employ square root factors of covariance. Square root methods afford two huge benefits:

1. guaranteed symmetry and easy-to-check positive-definiteness of covariance matrices
2. all numerical values are closer to unity, making square root methods more resistant to computational overflow/underflow issues

What do we mean by “roundoff error?” In binary representation, all rational numbers are represented as transformed sums of twos

$$1 = 2^0$$

$$3 = 2^0 + 2^1$$

$$\frac{1}{3} = \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{256} + \dots$$

This series of fractions has an equivalent binary representation, but this representation is limited to 24 bits of mantissa. We can round the true binary value to the 24-bit approximation

$$\frac{1}{3} = 0_b0101010101010101010101010\dots$$

$$\approx 0_b0101010101010101010101011$$

$$= \frac{11184811}{33554432}$$

$$= \frac{1}{3} - \frac{1}{100663296}$$

which yields an approximation error on the order of 10^{-8} . The difference between the true value and the value approximated by the machine is what we call *roundoff error*.

Say we have knowledge of the unit roundoff error and we call it $\epsilon_{\text{roundoff}}$. It turns out that, to the machine, both of the following are true:

$$1 + \epsilon_{\text{roundoff}} \equiv 1$$

$$1 + \epsilon_{\text{roundoff}}/2 \equiv 1$$

In MATLAB, the unit roundoff error can be obtained at any time and is given by the pre-loaded global variable `eps`. Note, however, that this value changes depending on the approximated number of interest, and MATLAB make this easy for us, too. So, if to the machine the following are true:

$$n + \epsilon_{\text{roundoff}} \equiv n$$

$$n + \epsilon_{\text{roundoff}}/2 \equiv n$$

in MATLAB, the unit roundoff error can be obtained at any time and is given by the command `eps(n)`.

Many of the roundoff problems discussed in the early literature were with respect to computers with a much lower bit resolution than is available today. However, roundoff can still be a problem in running a Kalman filter today. Even if the problem is well-conditioned, it can be made ill-conditioned by the implementation (in say, MATLAB).

Consider the measurement update at some un-indexed time that has Jacobians

$$\mathbf{H}_x = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 + \delta \end{bmatrix} \quad \text{and} \quad \mathbf{H}_v = \mathbf{I}_2$$

and covariance matrices

$$\mathbf{P}_{xx}^- = \mathbf{I}_3 \quad \text{and} \quad \mathbf{P}_{vv} = \delta^2 \mathbf{I}_2$$

where $\delta^2 < \epsilon_{\text{roundoff}}$ but $\delta > \epsilon_{\text{roundoff}}$.

In this case, although \mathbf{H}_x clearly has a rank of 2, the product of $\mathbf{H}_x \mathbf{P}_{xx}^- \mathbf{H}_x^T$, taking into consideration roundoff, equals

$$\begin{bmatrix} 3 & 3 + \delta \\ 3 + \delta & 3 + 2\delta \end{bmatrix}$$

which is a singular matrix (even if we add $\mathbf{H}_v \mathbf{P}_{vv} \mathbf{H}_v^T$). This will cause the measurement update to fail since \mathbf{P}_{zz}^- is not invertible.

By using square root factors, these effects are greatly alleviated, with the added benefit of easily checking if our covariance matrices remain positive definite. In some cases, we can also guarantee that they remain positive definite. In the following discussions on square-root filters, we will make use of three extremely useful techniques: Cholesky factorization, QR decomposition, and Cholesky factor updating. We will discuss the three methods here and illuminate their usefulness in filtering applications.

Let's begin by introducing square-root factors. Broadly speaking, a square-root factor (SRF) is an extension of scalar square roots into the space of matrices. We consider \mathbf{B} to be an SRF of \mathbf{A} if

$$\mathbf{B}\mathbf{B}^T = \mathbf{A}$$

There are other definitions, such as $\mathbf{B}^2 = \mathbf{A}$ and $\mathbf{B}^T\mathbf{B} = \mathbf{A}$, but we will adhere to the first one. In some cases, we might use the shorthand notation $\mathbf{B} = \sqrt{\mathbf{A}}$ with the understanding that this matrix satisfies the preceding relationship. There are several methods for computing these SRFs, and one of the more common ones is to use a

Cholesky factorization or a Cholesky decomposition. Another popular approach is to use an eigenvalue/eigenvector factorization. We will focus here on the Cholesky factorization.

Cholesky factors can be thought of as a generalization of scalar square roots to symmetric, positive definite matrices. MThe Cholesky decomposition process for a matrix requires that the target matrix be Hermitian (square and self-adjoint) and positive-definite. This means that the Cholesky decomposition can be applied to matrices with complex entries, but we will only concern ourselves with real matrices here (all this does is replace conjugate transposes with transposes). In fact, it turns out at that if \mathbf{A} is real and positive definite, its Cholesky factor is *unique*. This is not true for SRFs in general.

We will not discuss the actual algorithm here, but we will discuss how we can easily obtain Cholesky factors, which we will often just call SRFs. MATLAB makes the process very easy with the function `chol`, but we need to be a little bit careful with the form of the matrix it returns. As we have defined them, SRFs are of the form $\mathbf{B}\mathbf{B}^T = \mathbf{A}$, but `chol` returns a matrix such that $\mathbf{B}^T\mathbf{B} = \mathbf{A}$, which does not fit our expectation. We can circumvent this issue by using `chol` in one of two ways:

1. $\mathbf{B} = \text{chol}(\mathbf{A}, \text{'lower'})$

2. $\mathbf{B} = \text{chol}(\mathbf{A})'$

Both of these will return a lower triangular SRF of \mathbf{A} , and both of these produce the matrix \mathbf{B} such that $\mathbf{B}\mathbf{B}^T = \mathbf{A}$.

One last note on Cholesky factors before we move on: Cholesky factorization only works for symmetric, positive definite matrices. This requirement does not limit us, though. For the square root methods that we are developing, we are computing the SRFs of covariance matrices, which are symmetric and positive definite.

To begin to see the benefit of square-root methods, consider the discrete-time dynamical system

$$\mathbf{x}_k = \mathbf{F}_x \mathbf{x}_{k-1} + \mathbf{w}_{k-1}$$

where \mathbf{w}_{k-1} is a zero-mean, white-noise sequence with constant covariance \mathbf{P}_{ww} . Note also that $\mathbf{F}_{x,k-1} = \mathbf{F}_x$ is a constant matrix. For ease of exposition, we have taken \mathbf{F}_x and \mathbf{P}_{ww} to be constant matrices, but this is absolutely not a requirement. If we propagate the state estimation error covariance matrix for this system starting from $\mathbf{P}_{xx,0}$ at $k = 0$,

it follows that the propagated covariance at $k = 1$ is

$$\mathbf{P}_{xx,1} = \mathbf{F}_x \mathbf{P}_{xx,0} \mathbf{F}_x^T + \mathbf{P}_{ww}$$

Let's introduce square-root factors for $\mathbf{P}_{xx,k}$ to be

$$\mathbf{P}_{xx,k} = \mathbf{S}_{xx,k} \mathbf{S}_{xx,k}^T$$

where $k = 0, 1, 2, \dots$. We can use the square-root factors in the covariance propagation to find that

$$\mathbf{S}_{xx,1} \mathbf{S}_{xx,1}^T = \mathbf{F}_x \mathbf{S}_{xx,0} \mathbf{S}_{xx,0}^T \mathbf{F}_x^T + \mathbf{S}_{ww} \mathbf{S}_{ww}^T$$

where \mathbf{S}_{ww} is the SRF of \mathbf{P}_{ww} , i.e., $\mathbf{S}_{ww} \mathbf{S}_{ww}^T = \mathbf{P}_{ww}$. We can bow factor the covariance propagation equation according to

$$\mathbf{S}_{xx,1} \mathbf{S}_{xx,1}^T = [\mathbf{F}_x \mathbf{S}_{xx,0} \mid \mathbf{S}_{ww}] [\mathbf{F}_x \mathbf{S}_{xx,0} \mid \mathbf{S}_{ww}]^T$$

which tells us that

$$\mathbf{S}_{xx,1} = [\mathbf{F}_x \mathbf{S}_{xx,0} \mid \mathbf{S}_{ww}]$$

This is great, because this means that we have acquired the SRF of the covariance matrix at $k = 1$, and we have done so by operating only on SRFs. That is, we didn't need to compute a Cholesky factorization of $\mathbf{P}_{xx,1}$ to find $\mathbf{S}_{xx,1}$.

What if we want to move forward another step in time? Applying the same process, we find that

$$\begin{aligned}\mathbf{S}_{xx,2}\mathbf{S}_{xx,2}^T &= \mathbf{F}_x\mathbf{S}_{xx,1}\mathbf{S}_{xx,1}^T\mathbf{F}_x^T + \mathbf{S}_{ww}\mathbf{S}_{ww}^T \\ &= [\mathbf{F}_x\mathbf{S}_{xx,1} \mid \mathbf{S}_{ww}][\mathbf{F}_x\mathbf{S}_{xx,1} \mid \mathbf{S}_{ww}]^T\end{aligned}$$

meaning that

$$\begin{aligned}\mathbf{S}_{xx,2} &= [\mathbf{F}_x\mathbf{S}_{xx,1} \mid \mathbf{S}_{ww}] \\ &= [\mathbf{F}_x^2\mathbf{S}_{xx,0} \mid \mathbf{F}_x\mathbf{S}_{ww} \mid \mathbf{S}_{ww}]\end{aligned}$$

If we continue propagating through more steps in time, we end up with

$$\begin{aligned}\mathbf{S}_{xx,1} &= [\mathbf{F}_x\mathbf{S}_{xx,0} \mid \mathbf{S}_{ww}] \\ \mathbf{S}_{xx,2} &= [\mathbf{F}_x^2\mathbf{S}_{xx,0} \mid \mathbf{F}_x\mathbf{S}_{ww} \mid \mathbf{S}_{ww}]\end{aligned}$$

⋮

$$\mathbf{S}_{xx,k} = [\mathbf{F}_x^k \mathbf{S}_{xx,0} \mid \mathbf{F}_x^{k-1} \mathbf{S}_{ww} \mid \mathbf{F}_x^{k-2} \mathbf{S}_{ww} \mid \dots \mid \mathbf{S}_{ww}]$$

At each point in the sequence of propagation steps, we have the SRF of the covariance matrix at that time, and we have these only in terms of other SRFs (and the dynamics matrix, \mathbf{F}_x). With each step, however, $\mathbf{S}_{xx,k}$ grows larger along the second dimension (columnwise), demanding more memory than we are used to. Even though our SRF started off as a square matrix, it becomes rectangular. In fact at step k , the matrix has size $n \times (k + 1)n$. The fact that it is rectangular does not mean that it is not a valid SRF. In fact, it is relatively easy to check that $\mathbf{S}_{xx,k} \mathbf{S}_{xx,k}^T = \mathbf{P}_{xx,k}$, which is our definition of SRFs. The rectangular shape of the matrix just means that we have some complexities (in terms of memory allocation and management) that we might not want to take on.

Fortunately, this problem is pretty easy to handle using a QR decomposition. QR decomposition is a method (or a collection of techniques that includes Gram–Schmidt orthonormalization, Householder transformations, and Givens rotations) that we will use to triangularize matrices. The QR algorithm allows any rectangular matrix, call it

$\mathbf{A} \in \mathbb{R}^{\ell \times m}$, to be decomposed as

$$\mathbf{A} = \mathbf{Q}\mathbf{R}$$

where $\mathbf{Q} \in \mathbb{R}^{\ell \times \ell}$ is orthogonal, $\mathbf{R} \in \mathbb{R}^{\ell \times m}$ is upper triangular, and $\ell \geq m$. Note that the dimensions and specification of \mathbf{R} means that there is, in general, a block structure where the upper $\ell \times \ell$ square block of \mathbf{R} is upper triangular and the lower (generally non-square) block of \mathbf{R} is the null matrix. We will use the shorthand notation $\text{qr}\{\cdot\}$ to indicate a QR decomposition, specifically one in which only the upper square (triangular) block of \mathbf{R} is returned.

The QR decomposition is effectively triangularizing a matrix, but how can we use this in our filtering problem? Let's consider a QR decomposition of the form

$$\mathbf{A}^T = \mathbf{Q}\mathbf{R}^T$$

from which it follows that

$$\mathbf{A}\mathbf{A}^T = \mathbf{R}\mathbf{Q}^T\mathbf{Q}\mathbf{R}^T = \mathbf{R}\mathbf{R}^T$$

The last step in the preceding expression follows from the fact that \mathbf{Q} is orthogonal. This shows us that \mathbf{R} is a valid SRF of \mathbf{A} , which means that we can perform a QR

decomposition to transform a rectangular matrix into a valid, triangular, SRF. This is the property we want to exploit to avoid our SRFs from continually expanding in size and changing our memory requirements in filtering.

How do we utilize the QR decomposition for our problem? Thinking back to our SRF propagation problem, we have and $n \times (k + 1)n$ dimensional representation of $\mathbf{S}_{xx,k}$, and we want to find an square matrix equivalent with n rows and columns. As stated, QR operates on “tall” matrices, so we perform a QR decomposition on $\mathbf{S}_{xx,k}^T$. This returns the upper triangular factor \mathbf{R}^T , so we transpose the output to get our desired triangularization. This is also referred to as the LQ decomposition, but it makes use of MATLAB’s qr algorithm. In general, take the “wide” matrix, transpose it to get a “tall” matrix, perform the QR decomposition to get the triangularization, and transpose the output.

The last tool we will discuss here is the Cholesky update (or downdate). It sometimes occurs that we have the sum or difference of matrices, for example $\mathbf{A} \pm \mathbf{V}\mathbf{V}^T$, where \mathbf{V} is some matrix of appropriate dimension. When we are dealing with Cholesky factors, we are interested in how the SRF associated with \mathbf{A} changes through this operation. The Cholesky update allows us to obtain the transformed SRF of \mathbf{A} . If the operation is a sum, this is called a *Cholesky update*, and if the operation is a difference, this is called a

Cholesky downdate. The update/downdate is said to be rank- k , where \mathbf{V} has k columns. A rank-1 update/downdate occurs when $\mathbf{V} = \mathbf{v}$ is a vector. As usual, MATLAB makes this very easy with the function calls

1. `cholupdate(B,v,'+')` (the rank-1 update)
2. `cholupdate(B,v,'-')` (the rank-1 downdate)

where $\mathbf{B}\mathbf{B}^T = \mathbf{A}$.

These commands input and output upper triangular SRFs, so you have to be careful if you're working with lower-triangular factors. You always need to be aware of the assumed form that is being input/output in the algorithms that you're making use of.

4.3.1 The Potter Square Root Update

Square root filters were introduced, soon after the introduction of the Kalman filter, by James E. Potter to overcome the problems present in an ill-conditioned navigation problem for the Apollo program. The mission used an onboard sextant to measure angles between stars and the limb of the earth or moon. To avoid the numerical instability

caused by the ill-conditioned nature of the problem, Potter thought to define Cholesky factors of the covariance matrix $\mathbf{P} = \mathbf{S}\mathbf{S}^T$, and what resulted is what is now called the Potter square root filter.

We will develop the Potter update for the SRF of the covariance matrix by starting with the updated covariance matrix in the form

$$\begin{aligned}\mathbf{P}_{xx,k}^+ &= \mathbf{P}_{xx,k}^- - \mathbf{K}_k \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \\ &= \mathbf{P}_{xx,k}^- - \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T [\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k}]^{-1} \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^-\end{aligned}$$

which makes use of the Kalman gain, linear (or linearized) measurement models, and additive white noise with covariance $\mathbf{P}_{vv,k}$. This last specification is not required, and generalizations to this are straightforward.

We can replace the prior and posterior covariance matrices by appropriate SRFs to find

$$\mathbf{S}_{xx,k}^+ (\mathbf{S}_{xx,k}^+)^T = \mathbf{S}_{xx,k}^- \left\{ \mathbf{I}_n - \mathbf{M}_k [\mathbf{M}_k^T \mathbf{M}_k + \mathbf{P}_{vv,k}]^{-1} \mathbf{M}_k^T \right\} (\mathbf{S}_{xx,k}^-)^T$$

where n is the dimension of the state, ℓ is the dimension of the measurement, and $\mathbf{M}_k = (\mathbf{H}_{x,k} \mathbf{S}_{xx,k}^-)^T$ is an $n \times \ell$ matrix. The unfactored expression in curly brackets is

what keeps this from being a true square root method at the moment. If this term can be factored, then the posterior SRF can be found.

When the measurement is a scalar, the term we need to factor becomes a symmetric elementary matrix of the form

$$\mathbf{I}_n - \mathbf{M}_k [\mathbf{M}_k^T \mathbf{M}_k + \mathbf{P}_{vv,k}] \mathbf{M}_k^T = \mathbf{I}_n - \frac{\mathbf{m}_k \mathbf{m}_k^T}{P_{vv,k} + \|\mathbf{m}_k\|^2}$$

where $\mathbf{P}_{vv,k} = P_{vv,k}$ is now a positive scalar and $\mathbf{M}_k = \mathbf{m}_k$ is an n -dimensional column vector. Because the matrix square root of symmetric elementary matrices are also symmetric matrices, they are also Cholesky factors. That is,

$$(\mathbf{I}_n - s\mathbf{m}_k \mathbf{m}_k^T) = (\mathbf{I}_n - \sigma \mathbf{m}_k \mathbf{m}_k^T)(\mathbf{I}_n - \sigma \mathbf{m}_k \mathbf{m}_k^T)^T$$

where s and σ are defined as

$$s = \frac{1}{P_{vv,k} + \|\mathbf{m}_k\|^2} \quad \text{and} \quad \sigma = \frac{1 + \sqrt{1 - s\|\mathbf{m}_k\|^2}}{\|\mathbf{m}_k\|^2} = \frac{1 + \sqrt{P_{vv,k}/(P_{vv,k} + \|\mathbf{m}_k\|^2)}}{\|\mathbf{m}_k\|^2}$$

We can apply this factorization via symmetric elementary matrices to the covariance update to find

$$\mathbf{S}_{xx,k}^+ (\mathbf{S}_{xx,k}^+)^T = \mathbf{S}_{xx,k}^- (\mathbf{I}_n - \sigma \mathbf{m}_k \mathbf{m}_k^T) (\mathbf{I}_n - \sigma \mathbf{m}_k \mathbf{m}_k^T)^T (\mathbf{S}_{xx,k}^-)^T$$

from which it follows that the posterior square root factor is

$$\mathbf{S}_{xx,k}^+ = \mathbf{S}_{xx,k}^- (\mathbf{I}_n - \sigma \mathbf{m}_k \mathbf{m}_k^T)$$

which completes the Potter square root filter measurement update.

This version of the square root filter is very interesting from a historical perspective, as this was one of the earliest of a new class of filtering algorithms. Unfortunately, the application of this particular approach is somewhat limited since the Potter square root filter is only formulated for scalar measurements. However, the ramifications of Potter's insights cannot be understated, and we were subsequently able to learn from the methods of Potter and develop a more general square root filter. Square root filters (and variants thereof) are commonplace in modern real-time navigation applications.

4.3.2 The Square Root Kalman Filter

We will now take a look at the Kalman filter from the perspective of square root methods and see how we can leverage computational tools of Cholesky factorization, QR decomposition, and Cholesky factor updating. We will start from the discrete form of the Kalman filter, but the approach can be applied to other methods, as well.

We begin by considering the propagation stage of the Kalman filter, which, as a reminder, is described by

$$\mathbf{m}_{x,k}^- = \mathbf{F}_{x,k-1} \mathbf{m}_{x,k-1}^+$$

$$\mathbf{P}_{xx,k}^- = \mathbf{F}_{x,k-1} \mathbf{P}_{xx,k-1}^+ \mathbf{F}_{x,k-1}^T + \mathbf{F}_{w,k-1} \mathbf{P}_{ww,k-1}^+ \mathbf{F}_{w,k-1}^T$$

We are interested in rewriting this prediction step using only SRFs. For the mean, there is no modification required, as there is no dependence on covariance matrices. To adapt the covariance propagation to SRF propagation, we begin by defining SRFs. For a covariance matrix \mathbf{P}_{aa} , we use \mathbf{S}_{aa} to represent the SRF. This can be applied to the state estimation error covariance matrix, the process noise covariance matrix, the innovation covariance matrix, and the measurement noise covariance matrix. As such, we can substitute for

SRFs in to our covariance propagation to give

$$\mathbf{S}_{xx,k}^-(\mathbf{S}_{xx,k}^-)^T = \mathbf{F}_{x,k-1}\mathbf{S}_{xx,k-1}^+(\mathbf{S}_{xx,k-1}^+)^T\mathbf{F}_{x,k-1}^T + \mathbf{F}_{w,k-1}\mathbf{S}_{ww,k-1}\mathbf{S}_{ww,k-1}^T\mathbf{F}_{w,k-1}^T$$

which can be factored as

$$\mathbf{S}_{xx,k}^-(\mathbf{S}_{xx,k}^-)^T = [\mathbf{F}_{x,k-1}\mathbf{S}_{xx,k-1}^+ \mid \mathbf{F}_{w,k-1}\mathbf{S}_{ww,k-1}][\mathbf{F}_{x,k-1}\mathbf{S}_{xx,k-1}^+ \mid \mathbf{F}_{w,k-1}\mathbf{S}_{ww,k-1}]^T$$

This tells us that our *a priori* SRF of covariance is given by

$$\mathbf{S}_{xx,k}^- = [\mathbf{F}_{x,k-1}\mathbf{S}_{xx,k-1}^+ \mid \mathbf{F}_{w,k-1}\mathbf{S}_{ww,k-1}]$$

which is a non-square matrix. We want to transform this into a square matrix, and we do so by using the QR decomposition. From our previous discussion, we apply a QR decomposition on the transpose of $[\mathbf{F}_{x,k-1}\mathbf{S}_{xx,k-1}^+ \mid \mathbf{F}_{w,k-1}\mathbf{S}_{ww,k-1}]$, obtain the triangular factor output, and take the transpose of that output to produce the square (and triangular) $\mathbf{S}_{xx,k}^-$. This then completes the propagation stage.

With prediction done, let's look at the the Kalman filter measurement update. The

update equations for the Kalman filter are given by

$$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_{x,k}\mathbf{m}_{x,k}^-)$$

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

where the Kalman gain is

$$\mathbf{K}_k = \mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1}$$

and the cross-covariance and innovations covariance are

$$\mathbf{P}_{xz,k}^- = \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T$$

$$\mathbf{P}_{zz,k}^- = \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{H}_{v,k} \mathbf{P}_{vv,k} \mathbf{H}_{v,k}^T$$

Note that we have already explicitly made use of linear measurements to write out the expected measurement, cross covariance, and innovations covariance. Note also that we are not making use of the generalized Joseph form of the covariance update, but that we are rather making use of the form that relies on the Kalman gain, but which can work with nonlinear measurements.

Much like with the propagation stage, the update to the mean requires very little adjustment to accommodate SRFs; in fact, most of the adjustments comes to the covariance update and the innovations covariance. As such, we will start by looking at the innovations covariance. Using our established notation, we can define square root factors for the prior estimation error covariance ($\mathbf{S}_{xx,k}^-$), the innovations covariance ($\mathbf{S}_{zz,k}^-$), and for the measurement noise covariance ($\mathbf{S}_{vv,k}$). Now, substitute these SRFs into the innovation covariance expression to produce

$$\mathbf{S}_{zz,k}^-(\mathbf{S}_{zz,k}^-)^T = \mathbf{H}_{x,k}\mathbf{S}_{xx,k}^-(\mathbf{S}_{xx,k}^-)^T\mathbf{H}_{x,k}^T + \mathbf{H}_{v,k}\mathbf{S}_{vv,k}(\mathbf{S}_{vv,k})^T\mathbf{H}_{v,k}^T$$

which can be factored as

$$\mathbf{S}_{zz,k}^-(\mathbf{S}_{zz,k}^-)^T = [\mathbf{H}_{x,k}\mathbf{S}_{xx,k}^- \mid \mathbf{H}_{v,k}\mathbf{S}_{vv,k}] [\mathbf{H}_{x,k}\mathbf{S}_{xx,k}^- \mid \mathbf{H}_{v,k}\mathbf{S}_{vv,k}]^T$$

Therefore, a QR decomposition on the transpose of $[\mathbf{H}_{x,k}\mathbf{S}_{xx,k}^- \mid \mathbf{H}_{v,k}\mathbf{S}_{vv,k}]$ gives the transpose of $\mathbf{S}_{zz,k}^-$.

Let's now consider the cross covariance and substitute for the SRFs to yield

$$\mathbf{P}_{xz,k}^- = \mathbf{S}_{xx,k}^-(\mathbf{S}_{xx,k}^-)^T\mathbf{H}_{x,k}^T = \mathbf{S}_{xx,k}^-\left[\mathbf{H}_{x,k}\mathbf{S}_{xx,k}^-\right]^T$$

which can be computed based on the *a priori* SRF. Note that we are carrying out the computation in stages by first applying $\mathbf{H}_{x,k}$ to the prior SRF to avoid forming the prior covariance matrix. The cross-covariance matrix is the only covariance matrix that appears in the square-root form of the Kalman filter. All other covariance matrices are replaced by their SRFs.

Next, let's move on to the Kalman gain. If we apply the SRF of the innovations covariance, it follows that

$$\mathbf{K}_k = \mathbf{P}_{xz,k}^- [\mathbf{S}_{zz,k}^-(\mathbf{S}_{zz,k}^-)^T]^{-1} = \mathbf{P}_{xz,k}^- (\mathbf{S}_{zz,k}^-)^{-T} (\mathbf{S}_{zz,k}^-)^{-1}$$

Instead of directly computing the Kalman gain, we can define a set of update factors as

$$\mathbf{U}_k = \mathbf{P}_{xz,k}^- (\mathbf{S}_{zz,k}^-)^{-T}$$

and then the Kalman gain can be computed as

$$\mathbf{K}_k = \mathbf{U}_k (\mathbf{S}_{zz,k}^-)^{-1}$$

With the gain calculated, we can directly compute the updated mean according to our standard Kalman filter update, but, as with all things related to the covariance, the

covariance update for the Kalman filter requires some modifications in order to operate on square root factors.

Let's consider the covariance update and substitute SRFs into the covariance update to get

$$\mathbf{S}_{xx,k}^+ (\mathbf{S}_{xx,k}^+)^T = \mathbf{S}_{xx,k}^- (\mathbf{S}_{xx,k}^-)^T - \mathbf{K}_k \mathbf{S}_{zz,k}^- (\mathbf{S}_{zz,k}^-)^T \mathbf{K}_k^T$$

Note that, by our definition of \mathbf{U}_k , we have $\mathbf{U}_k = \mathbf{K}_k \mathbf{S}_{zz,k}^-$, such that we can write the covariance update, in terms of square root factors, as

$$\mathbf{S}_{xx,k}^+ (\mathbf{S}_{xx,k}^+)^T = \mathbf{S}_{xx,k}^- (\mathbf{S}_{xx,k}^-)^T - \mathbf{U}_k \mathbf{U}_k^T$$

This is in the form of the Cholesky update/downdate procedure, which can be handled one of two ways. Since $\mathbf{U}_k \in \mathbb{R}^{n \times m}$, where m is the number of elements in \mathbf{z}_k , this is a rank- m Cholesky downdate, meaning that the posterior SRF is

$$\mathbf{S}_{xx,k}^+ = \text{cholupdate}\{(\mathbf{S}_{xx,k}^-)^T, \mathbf{U}_k, ' - '\}^T$$

The transposes appearing in the preceding expression are to accommodate the MATLAB algorithm that operates on upper triangular SRFs. The rank- m downdate is perfectly

acceptable, but it can call be broken down into a series of m rank-1 downdates, by noting that

$$\mathbf{U}_k \mathbf{U}_k^T = [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_m] [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_m]^T$$

such that

$$\mathbf{S}_{xx,k}^+ (\mathbf{S}_{xx,k}^+)^T = \mathbf{S}_{xx,k}^- (\mathbf{S}_{xx,k}^-)^T - \mathbf{u}_1 \mathbf{u}_1^T - \mathbf{u}_2 \mathbf{u}_2^T - \cdots - \mathbf{u}_m \mathbf{u}_m^T$$

Each of the \mathbf{u}_i factors can be applied via a sequence of rank-1 downdates to complete the update. Either approach is valid, and it depends on the capabilities of cholupdate as to which one you should use.

Reminder: We took our square root factors to be defined such that $\mathbf{A} = \mathbf{B}\mathbf{B}^T$. Using MATLAB's chol routine means that \mathbf{B} is lower triangular to satisfy the requirement. Therefore, we keep lower-triangular SRFs throughout. Therefore, we have to be careful when using QR decomposition and Cholesky updates in order to conform with input/output expectations. Depending on your Cholesky factorization, QR decomposition, and Cholesky update algorithms, you may need to transpose results or adjust the equations appropriately. To check your algorithms, you can simultaneously implement covariance and SRF filters.

To summarize, we put everything together in a single table

System Model	$\boldsymbol{x}_k = \boldsymbol{F}_{x,k-1}\boldsymbol{x}_{k-1} + \boldsymbol{F}_{w,k-1}\boldsymbol{w}_{k-1}$
Meas. Model	$\boldsymbol{z}_k = \boldsymbol{H}_{x,k}\boldsymbol{x}_k + \boldsymbol{H}_{v,k}\boldsymbol{v}_k$
Init. Cond.	$\boldsymbol{m}_{x,0} = \text{E}\{\boldsymbol{x}(t_0)\}$ $\boldsymbol{S}_{xx,0} = \sqrt{\text{E}\{(\boldsymbol{x}(t_0) - \boldsymbol{m}_{x,0})(\boldsymbol{x}(t_0) - \boldsymbol{m}_{x,0})^T\}}$
Mean Prop.	$\boldsymbol{m}_{x,k} = \boldsymbol{F}_{x,k-1}\boldsymbol{m}_{x,k-1}$
SRF Prop.	$\boldsymbol{S}_{xx,k} = \text{qr}\{[\boldsymbol{F}_{x,k-1}\boldsymbol{S}_{xx,k-1} \mid \boldsymbol{F}_{w,k-1}\boldsymbol{S}_{ww,k-1}]^T\}^T$
Exp. Meas.	$\boldsymbol{m}_{z,k}^- = \boldsymbol{H}_{x,k}\boldsymbol{m}_{x,k}^-$
Cross Cov.	$\boldsymbol{P}_{xz,k}^- = \boldsymbol{S}_{xx,k}^-[\boldsymbol{H}_{x,k}\boldsymbol{S}_{xx,k}^-]^T$
Innov. SRF	$\boldsymbol{S}_{zz,k}^- = \text{qr}\{[\boldsymbol{H}_{x,k}\boldsymbol{S}_{xx,k}^- \mid \boldsymbol{H}_{v,k}\boldsymbol{S}_{vv,k}]^T\}^T$
Upd. Factors	$\boldsymbol{U}_k = \boldsymbol{P}_{xz,k}^-(\boldsymbol{S}_{zz,k}^-)^{-T}$
Kalman Gain	$\boldsymbol{K}_k = \boldsymbol{U}_k(\boldsymbol{S}_{zz,k}^-)^{-1}$
Mean Upd.	$\boldsymbol{m}_{x,k}^+ = \boldsymbol{m}_{x,k}^- + \boldsymbol{K}_k [\boldsymbol{z}_k - \boldsymbol{m}_{z,k}^-]$
SRF Upd.	$\boldsymbol{S}_{xx,k}^+ = \text{cholupdate}\{(\boldsymbol{S}_{xx,k}^-)^T, \boldsymbol{U}_k, \cdot - \cdot\}^T$

Note that we never use the Cholesky factorization during the propagation or update. It is only used to provide inputs on the initial condition as well as the process and measurement noise SRFs. If the process noise and measurement noise covariance matrices are constant, this means that only three applications of the Cholesky factorization are used.

A covariance matrix should never be formed for use inside of an SRF-based filter. If a covariance matrix is formed and used as part of the filter, it ceases to be a square-root filter. Covariance matrices may be formed (and may have to be formed) in order to analyze the results of the filter, but an SRF-based filter should never use covariance matrices internally. The one exception is the cross covariance.

The SRF update for the square-root Kalman filter started from the covariance update

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

Since this form of the covariance update assumes that the optimal gain is applied, one might be inclined to wonder if there is another form of the covariance update that can be leveraged to update SRFs. If the Joseph form is used as

$$\mathbf{P}_{xx,k}^+ = [\mathbf{I}_n - \mathbf{K}_k \mathbf{H}_{x,k}] \mathbf{P}_{xx,k}^- [\mathbf{I}_n - \mathbf{K}_k \mathbf{H}_{x,k}]^T + \mathbf{K}_k \mathbf{H}_{v,k} \mathbf{P}_{vv,k}^- \mathbf{H}_{v,k}^T \mathbf{K}_k^T$$

then SRFs of the posterior, prior, and measurement noise covariances can be applied, such that

$$\begin{aligned}\mathbf{S}_{xx,k}^+ (\mathbf{S}_{xx,k}^+)^T &= [\mathbf{I}_n - \mathbf{K}_k \mathbf{H}_{x,k}] \mathbf{S}_{xx,k}^- (\mathbf{S}_{xx,k}^-)^T [\mathbf{I}_n - \mathbf{K}_k \mathbf{H}_{x,k}]^T \\ &\quad + \mathbf{K}_k \mathbf{H}_{v,k} \mathbf{S}_{vv,k} (\mathbf{S}_{vv,k})^T \mathbf{H}_{v,k}^T \mathbf{K}_k^T\end{aligned}$$

This expression is readily factored, and the resulting non-square posterior SRF is given by

$$\mathbf{S}_{xx,k}^+ = [(\mathbf{I}_n - \mathbf{K}_k \mathbf{H}_{x,k}) \mathbf{S}_{xx,k}^- \mid \mathbf{K}_k \mathbf{H}_{v,k} \mathbf{S}_{vv,k}]$$

Since we try to avoid non-square SRFs, we can apply the QR decomposition as

$$\mathbf{S}_{xx,k}^+ = \text{qr}\left\{ [(\mathbf{I}_n - \mathbf{K}_k \mathbf{H}_{x,k}) \mathbf{S}_{xx,k}^- \mid \mathbf{K}_k \mathbf{H}_{v,k} \mathbf{S}_{vv,k}]^T \right\}^T$$

to ensure that the posterior SRF is square and triangular.

This form of the SRF update allows for any linear gain to be applied in the update, but it requires linear (or linearized) measurements to be used.

This gives us another form of the table

System Model	$\boldsymbol{x}_k = \boldsymbol{F}_{x,k-1}\boldsymbol{x}_{k-1} + \boldsymbol{F}_{w,k-1}\boldsymbol{w}_{k-1}$
Meas. Model	$\boldsymbol{z}_k = \boldsymbol{H}_{x,k}\boldsymbol{x}_k + \boldsymbol{H}_{v,k}\boldsymbol{v}_k$
Init. Cond.	$\boldsymbol{m}_{x,0} = \text{E}\{\boldsymbol{x}(t_0)\}$ $\boldsymbol{S}_{xx,0} = \sqrt{\text{E}\{(\boldsymbol{x}(t_0) - \boldsymbol{m}_{x,0})(\boldsymbol{x}(t_0) - \boldsymbol{m}_{x,0})^T\}}$
Mean Prop.	$\boldsymbol{m}_{x,k} = \boldsymbol{F}_{x,k-1}\boldsymbol{m}_{x,k-1}$
SRF Prop.	$\boldsymbol{S}_{xx,k} = \text{qr}\{[\boldsymbol{F}_{x,k-1}\boldsymbol{S}_{xx,k-1} \mid \boldsymbol{F}_{w,k-1}\boldsymbol{S}_{ww,k-1}]^T\}^T$
Exp. Meas.	$\boldsymbol{m}_{z,k}^- = \boldsymbol{H}_{x,k}\boldsymbol{m}_{x,k}^-$
Cross Cov.	$\boldsymbol{P}_{xz,k}^- = \boldsymbol{S}_{xx,k}^-[\boldsymbol{H}_{x,k}\boldsymbol{S}_{xx,k}^-]^T$
Innov. SRF	$\boldsymbol{S}_{zz,k}^- = \text{qr}\{[\boldsymbol{H}_{x,k}\boldsymbol{S}_{xx,k}^- \mid \boldsymbol{H}_{v,k}\boldsymbol{S}_{vv,k}]^T\}^T$
Upd. Factors	$\boldsymbol{U}_k = \boldsymbol{P}_{xz,k}^-(\boldsymbol{S}_{zz,k}^-)^{-T}$
Kalman Gain	$\boldsymbol{K}_k = \boldsymbol{U}_k(\boldsymbol{S}_{zz,k}^-)^{-1}$
Mean Upd.	$\boldsymbol{m}_{x,k}^+ = \boldsymbol{m}_{x,k}^- + \boldsymbol{K}_k [\boldsymbol{z}_k - \boldsymbol{m}_{z,k}^-]$
SRF Upd.	$\boldsymbol{S}_{xx,k}^+ = \text{qr}\{[(\boldsymbol{I}_n - \boldsymbol{K}_k\boldsymbol{H}_{x,k})\boldsymbol{S}_{xx,k}^- \mid \boldsymbol{K}_k\boldsymbol{H}_{v,k}\boldsymbol{S}_{vv,k}]^T\}^T$

Numerical Example of the Square Root Kalman Filter

To demonstrate the square root Kalman filter, and to offer some means of validating individual efforts, we will present a simple problem with numerical results to compare against. In the following, no units are provided, but the units of each term can be inferred by the problem under consideration.

Consider a robot moving across a floor, such that its state vector is

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

The system is taken to be linear with a constant velocity model, such that $\mathbf{F}_{x,k-1}$ is

$$\mathbf{F}_{x,k-1} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\Delta t = t_k - t_{k-1}$. Process noise is taken to be additive (i.e., $\mathbf{F}_{w,k-1} = \mathbf{I}_4$) with constant covariance

$$\mathbf{P}_{ww} = \text{diag}\left\{\left[1 \times 10^{-6} \ 1 \times 10^{-6} \ 1 \times 10^{-9} \ 1 \times 10^{-9}\right]\right\}$$

Note that the associated SRF is also constant. Assume that it is possible to take measurements of the robot's position on the floor as it moves, such that

$$\mathbf{H}_{x,k} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The measurements are to be collected every $\Delta t = 0.2$ seconds and posses a constant measurement noise covariance of

$$\mathbf{P}_{vv} = \begin{bmatrix} (0.5)^2 & 0 \\ 0 & (0.5)^2 \end{bmatrix}$$

As with the process noise, the associated SRF is also constant. The measurement noise is assumed to be additive, such that $\mathbf{H}_{v,k} = \mathbf{I}_2$. The initial mean and covariance are taken to be given by

$$\mathbf{m}_{x,0} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.5 \\ 1.0 \end{bmatrix} \quad \text{and} \quad \mathbf{P}_{xx,0} = \begin{bmatrix} 0.5000 & 0 & 0 & 0 \\ 0 & 0.5000 & 0 & 0 \\ 0 & 0 & 0.0500 & 0 \\ 0 & 0 & 0 & 0.0500 \end{bmatrix}$$

where the associated SRF, $\mathbf{S}_{xx,0}$, can be readily determined.

The true initial state of the robot is taken to be drawn from a Gaussian, such that $\mathbf{x}_0 \sim p_g(\mathbf{x}_0 ; \mathbf{m}_{x,0}, \mathbf{P}_{x,0})$, which produces

$$\mathbf{x}_{0,\text{true}} = \begin{bmatrix} 0.0805 \\ -0.3076 \\ 0.4881 \\ 1.0307 \end{bmatrix}$$

These numerical results are generated using MATLAB with its random number seed set according to `rng(100)`. Depending on the number generator, or the order in which you call for random numbers, your results may vary a little. To provide a good comparison, some numerical results are presented so that algorithmic outputs can be validated.

The first thing that we do is to compute the SRFs associated with the initial state estimation error covariance matrix, the process noise covariance matrix, and the measurement noise covariance matrix. We then propagate forward one step in time to the time of the

first measurement. This yields the *a priori* mean and square root factor to be

$$\mathbf{m}_{x,1}^- = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.5 \\ 1.0 \end{bmatrix} \quad \text{and} \quad \mathbf{S}_{xx,1}^- = \begin{bmatrix} -0.7085 & 0 & 0 & 0 \\ 0 & -0.7085 & 0 & 0 \\ -0.0141 & 0 & -0.2232 & 0 \\ 0 & -0.0141 & 0 & -0.2232 \end{bmatrix}$$

The prior SRF is found using the “economy form” of the QR decomposition in MATLAB using the developed procedure that ensures a lower triangular output.

At this time, we acquire a measurement, which is given by

$$\mathbf{z}_1 = \begin{bmatrix} 0.2024 \\ 0.2432 \end{bmatrix}$$

The innovation SRF for this measurement is

$$\mathbf{S}_{zz,1}^- = \begin{bmatrix} 0.8672 & 0 \\ 0 & 0.8672 \end{bmatrix}$$

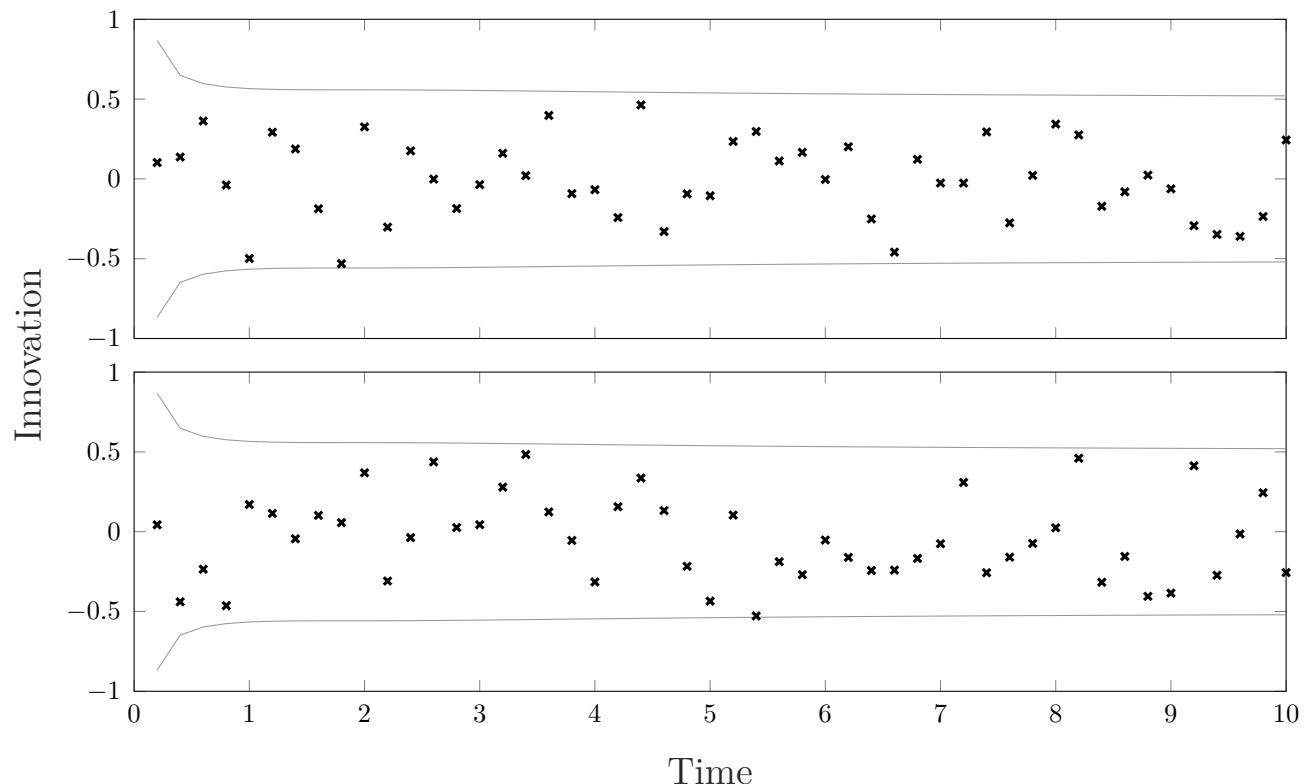
which, like the prior, is found using the “economy form” of the QR decomposition. Due to the structure of $\mathbf{F}_{x,k-1}$ and the initial covariance matrix/SRF for the state, there are

no correlations developed between the two position states, which is why there are no off-diagonal elements in $\mathbf{S}_{zz,1}^-$. Following through the square root Kalman filter equations, we obtain the *a posteriori* mean and square root factor as

$$\mathbf{m}_{x,1}^+ = \begin{bmatrix} 0.1683 \\ 0.2289 \\ 0.5014 \\ 1.0006 \end{bmatrix} \quad \text{and} \quad \mathbf{S}_{xx,1}^+ = \begin{bmatrix} -0.4085 & 0 & 0 & 0 \\ 0 & -0.4085 & 0 & 0 \\ -0.0081 & 0 & -0.2232 & 0 \\ 0 & -0.0081 & 0 & -0.2232 \end{bmatrix}$$

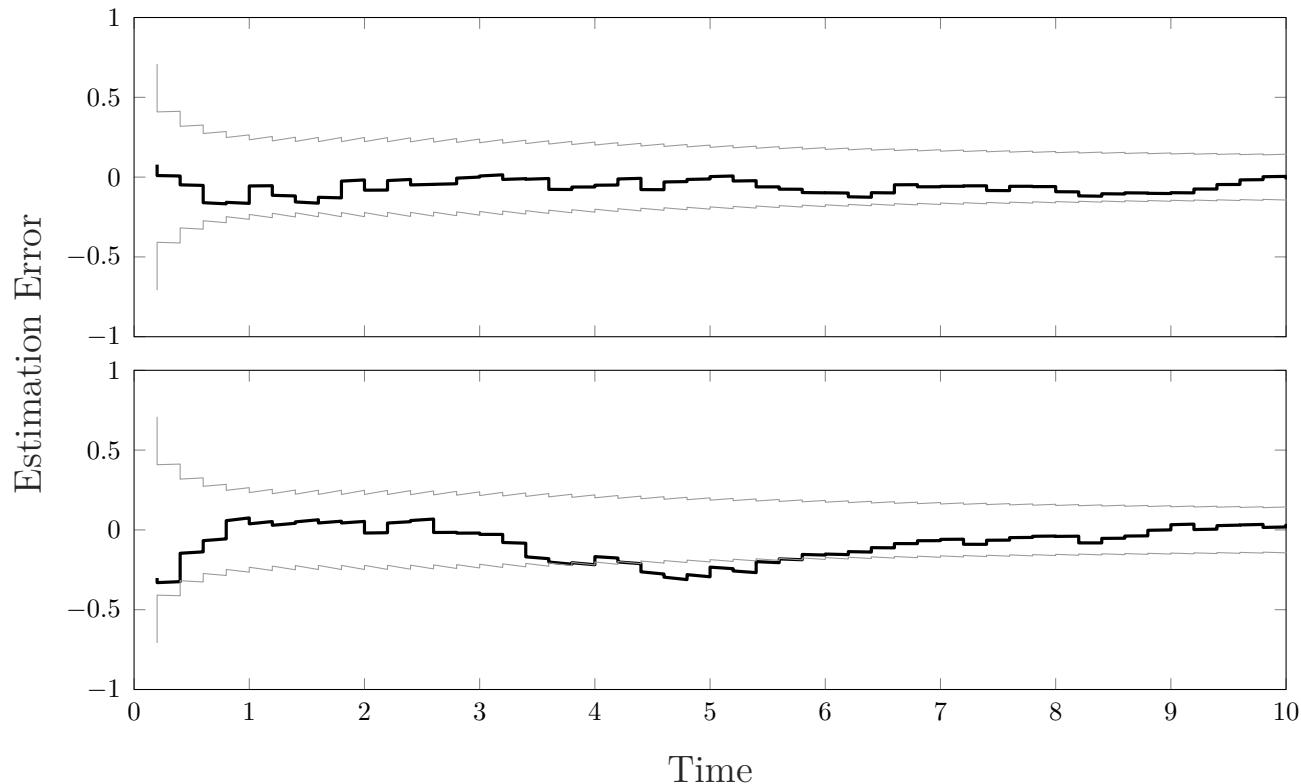
It is interesting to note that the diagonal elements of the SRF are negative. In fact, all elements of the SRF are negative. This is usually concerning when dealing with covariance matrices, as it implies that the variances are negative. When dealing with SRFs, however, this is completely acceptable. That being said, when you want to plot σ intervals, you cannot simply plot the diagonal elements of the SRF. You cannot even plot the absolute value of the diagonal elements. You actually need to form the full covariance matrix in order to determine the standard deviations for each channel of the state.

The full square root Kalman filter is applied for a simulation with a 10 second duration. The corresponding innovations and 1σ interval of the innovations SRF/covariance is given in the following figure.



The resulting estimation error and 1σ interval for the position states are shown in the

following figure.



4.3.3 The Square Root Extended Kalman Filter

Now that we have the square root Kalman filter, we can make use of our knowledge of the extended Kalman filter and build a square root formulation of the EKF. For this case, we consider the discrete version of the EKF, which has the system model

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{w}_{k-1})$$

$$z_k = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k)$$

The initial state has mean $\mathbf{m}_{x,0}$ and SRF $\mathbf{S}_{xx,0}$. The process and measurement noises are taken to be zero-mean and white with SRFs $\mathbf{S}_{ww,k-1}$ and $\mathbf{S}_{vv,k}$, respectively.

Given the similarities between the Kalman and extended Kalman filters, we can bypass many ground-up developments and directly state the results by noting that we

- apply the nonlinear dynamics to propagate the state estimate,
- replace $\mathbf{F}_{x,k-1}$ and $\mathbf{F}_{w,k-1}$ by Jacobians,
- apply the nonlinear measurement function to predict the measurement, and

- replace $\mathbf{H}_{x,k}$ and $\mathbf{H}_{v,k}$ by Jacobians.

System Model	$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{w}_{k-1})$
Meas. Model	$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k)$
Init. Cond.	$\mathbf{m}_{x,0} = \text{E}\{\mathbf{x}(t_0)\}$ $\mathbf{S}_{xx,0} = \sqrt{\text{E}\{(\mathbf{x}(t_0) - \mathbf{m}_{x,0})(\mathbf{x}(t_0) - \mathbf{m}_{x,0})^T\}}$
Mean Prop.	$\mathbf{m}_{x,k} = \mathbf{f}(\mathbf{m}_{x,k-1}, \mathbf{0}_w)$
SRF Prop.	$\mathbf{S}_{xx,k} = \text{qr}\{[\mathbf{F}_x(\mathbf{m}_{x,k-1}) \mathbf{S}_{xx,k-1} \mid \mathbf{F}_w(\mathbf{m}_{x,k-1}) \mathbf{S}_{ww,k-1}]^T\}^T$
Exp. Meas.	$\mathbf{m}_{z,k}^- = \mathbf{h}(\mathbf{m}_{x,k}^-, \mathbf{0}_v)$
Cross Cov.	$\mathbf{P}_{xz,k}^- = \mathbf{S}_{xx,k}^- [\mathbf{H}_x(\mathbf{m}_{x,k}^-) \mathbf{S}_{xx,k}^-]^T$
Innov. SRF	$\mathbf{S}_{zz,k}^- = \text{qr}\{[\mathbf{H}_x(\mathbf{m}_{x,k}^-) \mathbf{S}_{xx,k}^- \mid \mathbf{H}_v(\mathbf{m}_{x,k}^-) \mathbf{S}_{vv,k}]^T\}^T$
Upd. Factors	$\mathbf{U}_k = \mathbf{P}_{xz,k}^- (\mathbf{S}_{zz,k}^-)^{-T}$
Kalman Gain	$\mathbf{K}_k = \mathbf{U}_k (\mathbf{S}_{zz,k}^-)^{-1}$
Mean Upd.	$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k [\mathbf{z}_k - \mathbf{m}_{z,k}^-]$
SRF Upd.	$\mathbf{S}_{xx,k}^+ = \text{cholupdate}\{(\mathbf{S}_{xx,k}^-)^T, \mathbf{U}_k, ' - '\}^T$

While this table is formulated with the Cholesky update formulation of the SRF update, it is also possible to formulate it using the QR decomposition discussed previously.

4.3.4 Comments on Square Root Filters

The square root paradigm is an immensely powerful formulation for estimation that affords enhanced numerical stability at the cost of a (reasonable) increase in computational complexity. These filters are guaranteed to preserve symmetry of the covariance matrix (even though it is never formed), and are extremely resistant to losing positive-definiteness of their error covariance matrices and, by moving stored values closer to unity, are much more resistant to numerical underflow/overflow.

The backbone of these procedures, QR decomposition and Cholesky updates/downdates, are fundamental algorithms that are used in a wide variety of disciplines from biological sciences to computer vision and everything in between. Thanks to the wide use of these two algorithms, extremely computationally efficient routines can be freely obtained for any number of programming languages.

We will see later on that with point-based filters (such as the unscented Kalman filter),

we can get failures in the algorithms when a new set of points is determined. This occurs due to the loss of positive-definiteness in the error covariance, and positive-definiteness is a *requirement* of Cholesky decomposition. Square root filters avoid performing Cholesky decomposition entirely, so they do not suffer from this weakness.

There are, of course, downsides to square root filters. There is no doubt that the square root versions of these filters are more computationally burdensome, and that their implementations are less straightforward. Remember: as presented, we assume that *all* square-root factors are lower triangular matrices. Depending on the routines/algorithms you use for QR or Cholesky updating, you may or may not have to add transposes where appropriate. Note that what is presented here is not necessarily the most efficient way of approaching this problem but instead was designed for ease of exposition.

Derivative-Free Kalman Filtering

For a system of the form

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k$$

which we have referred to previously as the “additive noise model,” we know that, while we can’t use the Kalman filter, we can apply the framework of the Kalman filter. For this type of system, the Kalman framework provides us with the governing equations for approximate, linear, minimum mean-square error estimation. Typically, we break this into propagation and update stages.

By definition, the propagation stage is

$$\mathbf{m}_{x,k}^- = \text{E}\{\mathbf{x}_k\}$$

$$\mathbf{P}_{xx,k}^- = \text{E}\{(\mathbf{x}_k - \text{E}\{\mathbf{x}_k\})(\mathbf{x}_k - \text{E}\{\mathbf{x}_k\})^T\}$$

If it is assumed that the process noise is zero mean with covariance $\mathbf{P}_{ww,k-1}$ and that the process noise is uncorrelated to the state, then it follows that

$$\mathbf{m}_{x,k}^- = \text{E}\{\mathbf{f}(\mathbf{x}_{k-1})\}$$

$$\mathbf{P}_{xx,k}^- = \text{E}\{(\mathbf{f}(\mathbf{x}_{k-1}) - \text{E}\{\mathbf{f}(\mathbf{x}_{k-1})\})(\mathbf{f}(\mathbf{x}_{k-1}) - \text{E}\{\mathbf{f}(\mathbf{x}_{k-1})\})^T\} + \mathbf{P}_{ww,k-1}$$

When we receive new data, the update is given by

$$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{m}_{z,k}^-)$$

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^-\mathbf{K}_k^T - \mathbf{K}_k(\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k\mathbf{P}_{zz,k}^-\mathbf{K}_k^T$$

where \mathbf{K}_k is any linear gain, which includes the Kalman gain that is given by

$$\mathbf{K}_k = \mathbf{P}_{xz,k}^-(\mathbf{P}_{zz,k}^-)^{-1}$$

The expected value of the measurement, the cross-covariance, and the innovation covariance are defined as

$$\mathbf{m}_{z,k}^- = \text{E}\{\mathbf{z}_k\}$$

$$\mathbf{P}_{xz,k}^- = \text{E}\{(\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T\}$$

$$\mathbf{P}_{zz,k}^- = \text{E}\{(\mathbf{z}_k - \mathbf{m}_{z,k}^-)(\mathbf{z}_k - \mathbf{m}_{z,k}^-)^T\}$$

If the measurement noise is taken to be zero mean with covariance $\mathbf{P}_{vv,k}$ and to be uncorrelated with the state, then it follows that

$$\mathbf{m}_{z,k}^- = \text{E}\{\mathbf{h}(\mathbf{x}_k)\}$$

$$\mathbf{P}_{xz,k}^- = \text{E}\{(\mathbf{x}_k - \text{E}\{\mathbf{x}_k\})(\mathbf{h}(\mathbf{x}_k) - \text{E}\{\mathbf{h}(\mathbf{x}_k)\})^T\}$$

$$\mathbf{P}_{zz,k}^- = \text{E}\{(\mathbf{h}(\mathbf{x}_k) - \text{E}\{\mathbf{h}(\mathbf{x}_k)\})(\mathbf{h}(\mathbf{x}_k) - \text{E}\{\mathbf{h}(\mathbf{x}_k)\})^T\} + \mathbf{P}_{vv,k}$$

Thus, approximate, linear, minimum mean-square error estimation requires the calculation of five expected values:

$$\mathbf{m}_{x,k}^- = \text{E}\{\mathbf{f}(\mathbf{x}_{k-1})\}$$

$$\mathbf{P}_{xx,k}^- = \text{E}\{(\mathbf{f}(\mathbf{x}_{k-1}) - \text{E}\{\mathbf{f}(\mathbf{x}_{k-1})\})(\mathbf{f}(\mathbf{x}_{k-1}) - \text{E}\{\mathbf{f}(\mathbf{x}_{k-1})\})^T\} + \mathbf{P}_{ww,k-1}$$

$$\mathbf{m}_{z,k}^- = \text{E}\{\mathbf{h}(\mathbf{x}_k)\}$$

$$\mathbf{P}_{xz,k}^- = \text{E}\{(\mathbf{x}_k - \text{E}\{\mathbf{x}_k\})(\mathbf{h}(\mathbf{x}_k) - \text{E}\{\mathbf{h}(\mathbf{x}_k)\})^T\}$$

$$\mathbf{P}_{zz,k}^- = \text{E}\{(\mathbf{h}(\mathbf{x}_k) - \text{E}\{\mathbf{h}(\mathbf{x}_k)\})(\mathbf{h}(\mathbf{x}_k) - \text{E}\{\mathbf{h}(\mathbf{x}_k)\})^T\} + \mathbf{P}_{vv,k}$$

If we can compute these five expectations, we can assemble a form of the Kalman filter. Whenever the dynamics and measurements are linear, the results of the expectations in combination with propagation and update framework give us the governing equations that describe the Kalman filter. We also know that it is possible to extend this framework to handle nonlinear systems using first-order Taylor series expansions to compute the expectations, and this process gives rise to the extended Kalman filter.

Recent developments in estimation have sought to improve upon the EKF's use of linearization to approximate expectations when confronted with nonlinear dynamics and measurements. There are a variety of ways to improve on the first-order Taylor series expansion, with the most obvious one being to extend the order of the Taylor series expansion. In fact, this was pursued soon after the extended Kalman filter was created.

Another class of algorithms that emerged has a common trait of avoiding series expansions in favor of quadrature-type approximation methods for the expectations. To put these into a common perspective, we can recall the definition of the expected value and view the five expectations as the integral equations

$$\mathbf{m}_{x,k}^- = \int \mathbf{f}(\mathbf{x}_{k-1}) p(\mathbf{x}_{k-1}) d\mathbf{x}_{k-1}$$

$$\mathbf{P}_{xx,k}^- = \int (\mathbf{f}(\mathbf{x}_{k-1}) - \mathbf{m}_{x,k}^-)(\mathbf{f}(\mathbf{x}_{k-1}) - \mathbf{m}_{x,k}^-)^T p(\mathbf{x}_{k-1}) d\mathbf{x}_{k-1} + \mathbf{P}_{ww,k-1}$$

$$\mathbf{m}_{z,k}^- = \int \mathbf{h}(\mathbf{x}_k) p(\mathbf{x}_k) d\mathbf{x}_k$$

$$\mathbf{P}_{xz,k}^- = \int (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{h}(\mathbf{x}_k) - \mathbf{m}_{z,k}^-)^T p(\mathbf{x}_k) d\mathbf{x}_k$$

$$\mathbf{P}_{zz,k}^- = \int (\mathbf{h}(\mathbf{x}_k) - \mathbf{m}_{z,k}^-)(\mathbf{h}(\mathbf{x}_k) - \mathbf{m}_{z,k}^-)^T p(\mathbf{x}_k) d\mathbf{x}_k + \mathbf{P}_{vv,k}$$

where $p(\mathbf{x}_{k-1})$ is the probability density function (pdf) of the state after the inclusion of any available information at t_{k-1} and $p(\mathbf{x}_k)$ is the pdf of the state prior to the inclusion of any available information at t_k . Provided that we can evaluate (or approximate) these integrals, we can apply the Kalman filter framework. Any resulting variant of the Kalman filter is still a linear filter, even though we are operating with nonlinear systems.

Each of the integrals required is of the form

$$I = \int \mathbf{g}(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}$$

A “simple” procedure for approximating these expectation integrals is to use Monte Carlo integration

$$I \approx \frac{1}{N} \sum_{i=1}^N \mathbf{g}(\mathbf{x}^{(i)})$$

where there are N randomly sampled points, $\mathbf{x}^{(i)}$, drawn from the density $p(\mathbf{x})$. This method is very general in nature, but it converges as \sqrt{N} . The algorithm that results from this sampling approach is sometimes called the Monte Carlo Kalman filter (MCKF),

as this is a Monte Carlo evaluation of the integrals. A general rule of thumb is to use $N = 10^n$ sample points. Even just working with three-dimensional position and velocity would require 10^6 points if this rule of thumb is used, and this leads to the so-called “curse of dimensionality.” This curse of dimensionality is what we wish to avoid with the class of derivative-free Kalman filters.

Derivative-free Kalman filters are so-called because they avoid the use of derivative operations, like linearization, to compute the expected values required in the Kalman framework. Effectively, we want to find a way to use a point-based approach, but we want to choose our points intelligently. When developing the derivative-free approaches, it is helpful to note that each of the five expectations/integrals may be viewed as computing statistics of the nonlinear transformation

$$\mathbf{y} = \mathbf{g}(\mathbf{x})$$

where the mean and covariance of \mathbf{x} are known and we want to compute

1. the mean of \mathbf{y}
2. the covariance of \mathbf{y}

3. the cross-covariance of \mathbf{x} and \mathbf{y}

For the propagation stage, our nonlinear function is $\mathbf{f}(\mathbf{x}_{k-1})$, and the mean and covariance of \mathbf{y} are the propagated mean and covariance. For the update stage, our nonlinear function is $\mathbf{h}(\mathbf{x}_k)$, and the mean, covariance, and cross-covariance of \mathbf{y} are required to compute the Kalman gain, covariance update, and mean update. Thus, if we can compute statistics through this general nonlinear function, then we can implement the Kalman filter framework directly.

5.1 The Unscented Kalman Filter

The unscented transform (UT) is a relatively recent numerical method that can also be used for approximating the joint distribution of random variables \mathbf{x} and \mathbf{y} , where the pdf of \mathbf{x} is known and $\mathbf{y} = \mathbf{g}(\mathbf{x})$. When we say that the pdf of \mathbf{x} is known, we need to have some parameterization of it. In the following discussion, we will assume that $p(\mathbf{x}) = p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx})$ is a Gaussian pdf with known mean and covariance. The assumption of Gaussianity is *not* required, but it does greatly simplify the interpretation of the UT.

We will consider some generic nonlinear transformation $\mathbf{g}(\mathbf{x})$ to discuss the UT in general. Once we start applying the technique to estimation for dynamic systems, this nonlinear transformation will take the form of our system dynamics and measurement model.

Where methods such as linearization attempt to approximate the behavior of $\mathbf{g}(\mathbf{x})$ (via Taylor Series), the UT instead attempts to match the first and second moments of the target distribution (i.e. the target mean and covariance). That is, instead of approximating the nonlinear function, we are attempting to approximate moments of the distribution.

The central idea of the UT is to deterministically choose a fixed (and relative small) number of so-called “sigma points” to capture the mean and covariance of the original distribution of \mathbf{x} exactly. These sigma points are then subjected to the nonlinear function, and mean and covariance are then extracted from these transformed points. While this may feel very reminiscent to the Monte Carlo integration approach, they are in fact quite different due to the deterministic sampling approach used by the UT.

To begin our discussion, we need to review the matrix square root factor. We define

a matrix square root factor for some square matrix \mathbf{A} as any matrix that satisfies

$$\sqrt{\mathbf{A}}\sqrt{\mathbf{A}}^T = \mathbf{A}$$

The question then is, how does one compute the square root of a matrix? While it may be tempting to use the MATLAB command `sqrtm`, this returns a matrix $\sqrt{\mathbf{A}}$ which satisfies $\sqrt{\mathbf{A}}\sqrt{\mathbf{A}} = \mathbf{A}$ and **not** $\sqrt{\mathbf{A}}\sqrt{\mathbf{A}}^T = \mathbf{A}$. Fortunately, there are a few methods that return the matrix we want, and we will discuss two of them here.

The first method that is commonly employed is eigenvalue/eigenvector decomposition (or spectral decomposition) of \mathbf{A} . That is, the matrix \mathbf{A} is decomposed into the form

$$\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^T$$

where Λ is a diagonal matrix containing the eigenvalues of \mathbf{A} and \mathbf{V} is an orthogonal matrix containing its corresponding eigenvectors. As we are looking for the square root factor of \mathbf{A} , we are interested in finding $\sqrt{\mathbf{V}\Lambda\mathbf{V}^T}$. It turns out that

$$\sqrt{\mathbf{V}\Lambda\mathbf{V}^T} = \mathbf{V}\sqrt{\Lambda}$$

This can be proven by simply squaring both sides (but let's just look at the right hand side):

$$[\mathbf{V}\sqrt{\Lambda}][\mathbf{V}\sqrt{\Lambda}]^T = \mathbf{V}\sqrt{\Lambda}\sqrt{\Lambda}^T\mathbf{V}^T = \mathbf{V}\Lambda\mathbf{V}^T = \mathbf{A}$$

which is exactly the result we expect.

One interesting thing about matrix square root factors is that there are an infinite number of them. For instance,

$$\sqrt{\mathbf{V}\Lambda\mathbf{V}^T} = \mathbf{V}\sqrt{\Lambda}\mathbf{V}^T$$

is also a valid square root factor of the matrix \mathbf{A} . In fact, for any matrix \mathbf{U} such that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ (meaning that \mathbf{U} is orthonormal)

$$\sqrt{\mathbf{V}\Lambda\mathbf{V}^T} = \mathbf{V}\sqrt{\Lambda}\mathbf{U}^T$$

is a valid square root factor of the matrix \mathbf{A} . To see this, we simply form the product of this matrix with its transpose, or

$$[\mathbf{V}\sqrt{\Lambda}\mathbf{U}^T][\mathbf{V}\sqrt{\Lambda}\mathbf{U}^T]^T = \mathbf{V}\sqrt{\Lambda}\mathbf{U}^T\mathbf{U}\sqrt{\Lambda}^T\mathbf{V}^T = \mathbf{V}\sqrt{\Lambda}\sqrt{\Lambda}^T\mathbf{V}^T = \mathbf{A}$$

The most natural selection for the square root factor using the spectral decomposition is the base representation

$$\sqrt{\mathbf{A}} = \mathbf{V}\sqrt{\Lambda}$$

Since Λ is a diagonal matrix, its square root is the matrix which contains the square roots of its diagonal entries on its own diagonal, and this is easy to compute. If \mathbf{A} is known to be positive-definite and symmetric (such as what we might expect from a covariance matrix), we should see no issues with negative or complex eigenvalues. This method can be implemented in MATLAB with the `eig` function.

The second method that is commonly used by many is known as the Cholesky factor of a matrix \mathbf{A} . As opposed to the previous method, right away we assume that \mathbf{A} is a Hermitian (which here simply requires it is square and self-adjoint), positive-definite matrix. The Cholesky factor \mathbf{L} is said to be the lower triangular matrix which satisfies

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

This is a result which can be easily obtained in MATLAB via the command `chol`. Be careful, however, as MATLAB naturally returns an *upper* triangular matrix, and we are

interested in the *lower* triangular version. This is remedied by simply transposing what is provided by the command or via a modified function call given by `chol(A, 'lower')`. As with the spectral factorization method, we see that there are an infinite number of square root factors by simply post-multiplying the Cholesky factor by any square orthonormal matrix of appropriate dimension.

Note that for each method, depending on the properties of \mathbf{A} , the specific resulting square root factor may be unique; square root factors, however, are not, in general, unique. Let's look at some numerical examples of these two methods. Say we are interested in finding a square root factor of some matrix \mathbf{A} .

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & -1 \\ -1 & 2 & 0 \\ -1 & 0 & 3 \end{bmatrix}$$

If we were to find a square root factor via eigen-decomposition, we would get

$$\sqrt{\mathbf{A}}_{\text{eig}} = \mathbf{V} \sqrt{\boldsymbol{\Lambda}} = \begin{bmatrix} -0.2931 & -0.4491 & -0.8440 \\ -0.1560 & 1.2931 & 0.5509 \\ -0.1018 & -0.6881 & 1.5863 \end{bmatrix}$$

If we were to find a square root factor via Cholesky decomposition, we would get

$$\sqrt{\mathbf{A}}_{\text{chol}} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & -1 & 1 \end{bmatrix}$$

Both of these can be easily checked by simply checking to see if $\sqrt{\mathbf{A}}\sqrt{\mathbf{A}}^T = \mathbf{A}$.

Now that we have a good feeling for matrix square root factors, let's go back to the unscented transform and discuss how we generate points and transform them in order to approximate expectation integrals. Given the nonlinear function $\mathbf{y} = \mathbf{g}(\mathbf{x})$, where \mathbf{x} is Gaussian with mean, \mathbf{m}_x , and covariance, \mathbf{P}_{xx} , we want to approximate the mean and covariance of \mathbf{y} and the cross-covariance between \mathbf{x} and \mathbf{y} . That is, we want to approximate

$$\mathbf{m}_y = \int \mathbf{g}(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}$$

$$\mathbf{P}_{yy} = \int (\mathbf{g}(\mathbf{x}) - \mathbf{m}_y)(\mathbf{g}(\mathbf{x}) - \mathbf{m}_y)^T p(\mathbf{x}) d\mathbf{x}$$

$$\mathbf{P}_{xy} = \int (\mathbf{x} - \mathbf{m}_x)(\mathbf{g}(\mathbf{x}) - \mathbf{m}_y)^T p(\mathbf{x}) d\mathbf{x}$$

The first step of the UT is to draw sigma points for the input, \mathbf{x} . For a random variable $\mathbf{x} \in \mathbb{R}^n$ with pdf $p(\mathbf{x}) = p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx})$, a set of $2n + 1$ sigma points is given by

$$\boldsymbol{\chi}^{(0)} = \mathbf{m}_x$$

$$\boldsymbol{\chi}^{(i)} = \mathbf{m}_x + \sqrt{n + \lambda} [\mathbf{S}_{xx}]_i$$

$$\boldsymbol{\chi}^{(i+n)} = \mathbf{m}_x - \sqrt{n + \lambda} [\mathbf{S}_{xx}]_i$$

for $i = 1, \dots, n$ where \mathbf{S}_{xx} is an SRF such that $\mathbf{P}_{xx} = \mathbf{S}_{xx} \mathbf{S}_{xx}^T$, and $[\cdot]_i$ denotes the i^{th} column of the matrix. The UT contains several user-selected parameters: α and κ are parameters that determine the spread of the sigma points around the mean, and λ is a scaling parameter defined such that

$$n + \lambda = \alpha^2(n + \kappa)$$

Conventionally, α is small and positive, e.g., $10^{-4} \leq \alpha \leq 1$, and κ is often set such that $n + \kappa = 3$. Each sigma point also has two weights associated with it. There are mean

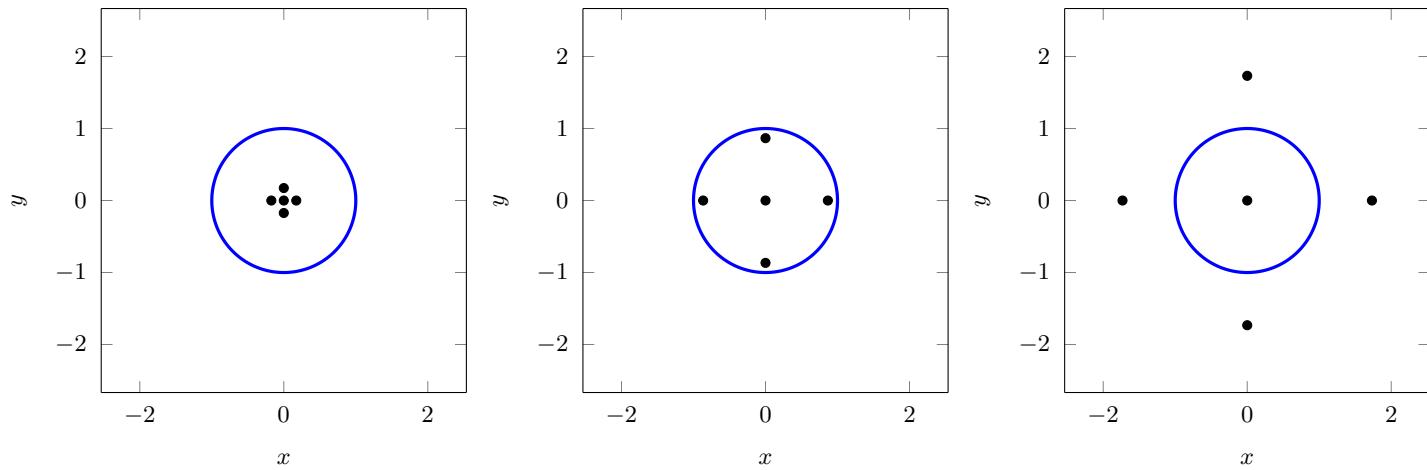
weights, $w_i^{(m)}$, and covariance weights, $w_i^{(c)}$, for $i = 0, \dots, 2n$, which are defined as

$$\begin{aligned} w_0^{(m)} &= \frac{\lambda}{n + \lambda} & w_0^{(c)} &= \frac{\lambda}{n + \lambda} + (1 - \alpha^2 + \beta) \\ w_i^{(m)} &= \frac{1}{2(n + \lambda)} & w_i^{(c)} &= \frac{1}{2(n + \lambda)} \end{aligned}$$

It is worth noting that, in general, the mean weights sum to one, but the covariance weights do not unless $(1 - \alpha^2 + \beta) = 0$. The value β is an additional nonnegative algorithm parameter that can be used for incorporating prior information on the non-Gaussian distribution of \boldsymbol{x} (see Wan and van der Merwe¹ for additional details regarding how to obtain these weights and techniques for selecting β). In the case that no prior information is desired to be added, β should be set to zero. In the case that the prior is Gaussian, the optimal choice turns out to be $\beta = 2$.

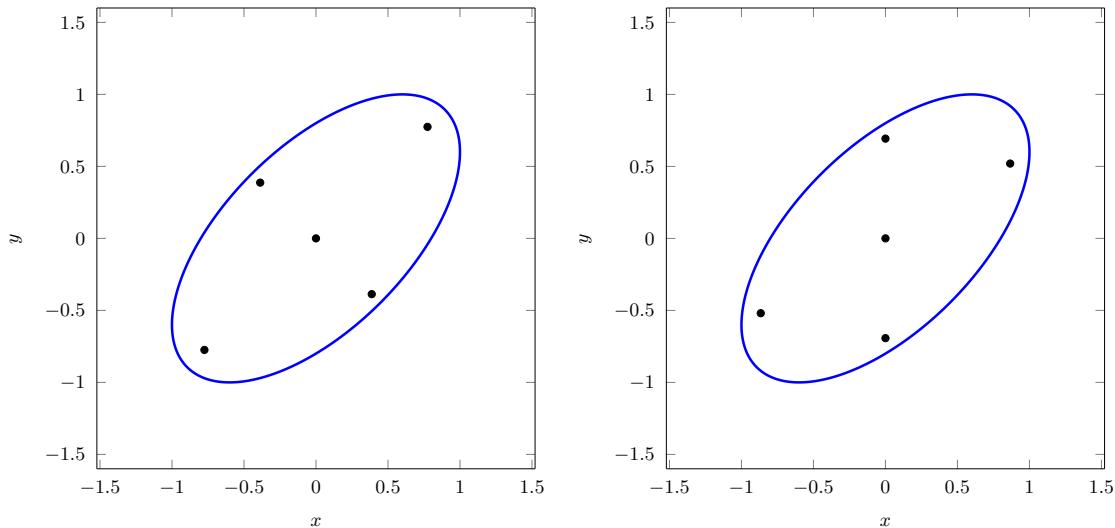
Let's look at a few cases of the sigma point generation for different values of α to show how this parameter controls the spread of the sigma points. We will use a zero-mean identity-covariance input and α values of 0.1, 0.5, and 1.0.

¹Wan, E. A. and van der Merwe, R., *The unscented Kalman filter*, Ch. 7 of Haykin, S. (ed.), **Kalman Filtering and Neural Networks**, Wiley, 2001.



It is clear that smaller values of α lead to sigma points more tightly clustered about the mean of the distribution, and that increasing α spreads out the sigma points away from the mean.

We can also look at how the choice of the square root factor influences the determination of the sigma points for the input. Here, we use both a spectral factorization and a Cholesky decomposition, both with $\alpha = 0.5$.



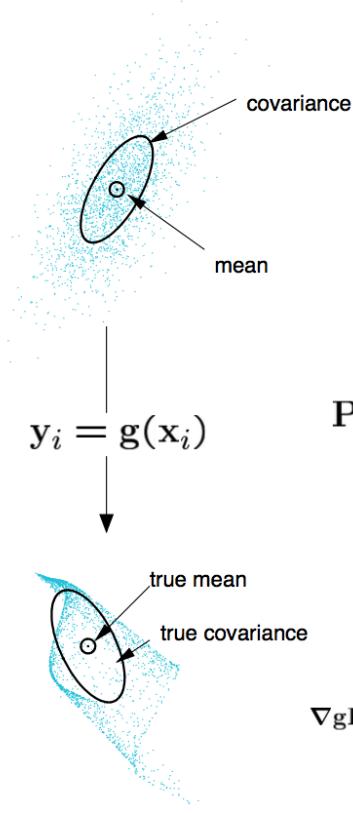
Once sigma points are generated for the input, the nonlinear transformation is applied to each point to get a set of transformed sigma points as

$$\mathbf{y}^{(i)} = \mathbf{g}(\mathbf{x}^{(i)}) , \quad i = 0, \dots, 2n$$

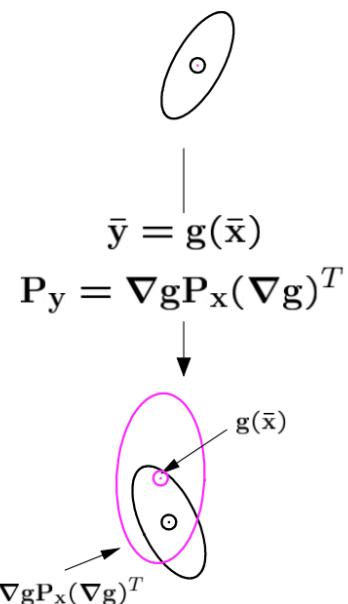
This is visually depicted in the following figure (taken from van der Merwe²)

²van der Merwe, R., *Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models*, Ph.D. thesis, Oregon Health and Science University, 2004.

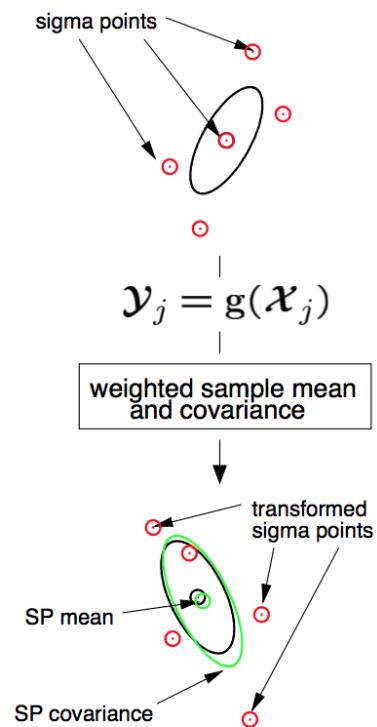
Actual (sampling)



Linearized (EKF)



Sigma–Point



The final step in the UT is to approximate the mean, covariance, and cross-covariance as

$$\boldsymbol{m}_y \approx \sum_{i=0}^{2n} w_i^{(m)} \boldsymbol{\mathcal{Y}}^{(i)}$$

$$\boldsymbol{P}_{yy} \approx \sum_{i=0}^{2n} w_i^{(c)} (\boldsymbol{\mathcal{Y}}^{(i)} - \boldsymbol{m}_y) (\boldsymbol{\mathcal{Y}}^{(i)} - \boldsymbol{m}_y)^T$$

$$\boldsymbol{P}_{xy} \approx \sum_{i=0}^{2n} w_i^{(c)} (\boldsymbol{\mathcal{X}}^{(i)} - \boldsymbol{m}_x) (\boldsymbol{\mathcal{Y}}^{(i)} - \boldsymbol{m}_y)^T$$

Let's take a look at an application of the unscented transform. In this case, we will consider the nonlinear transformation from polar coordinates to Cartesian coordinates given by

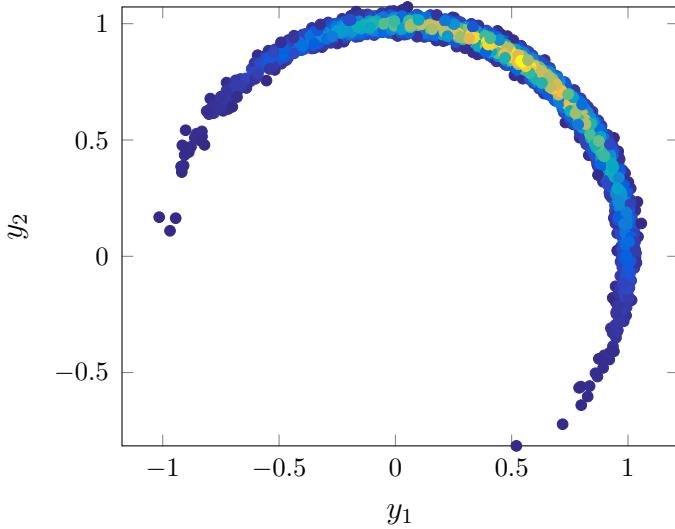
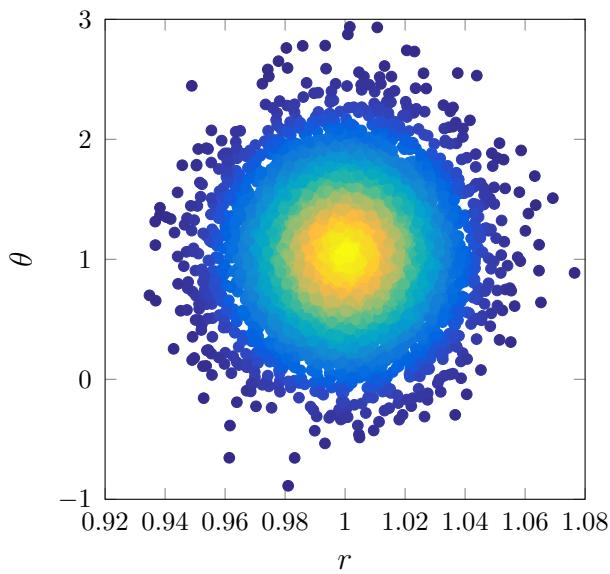
$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} r \cos \theta \\ r \sin \theta \end{bmatrix} = \boldsymbol{g}(\boldsymbol{x})$$

where

$$\boldsymbol{x} = \begin{bmatrix} r \\ \theta \end{bmatrix}, \quad \boldsymbol{m}_x = \begin{bmatrix} 1 \\ 60^\circ \end{bmatrix}, \quad \text{and} \quad \boldsymbol{P}_{xx} = \begin{bmatrix} (0.02)^2 & 0 \\ 0 & (30^\circ)^2 \end{bmatrix}$$

Our goal is to use knowledge of the mean and covariance of \boldsymbol{x} to approximate the mean and covariance of \boldsymbol{y} .

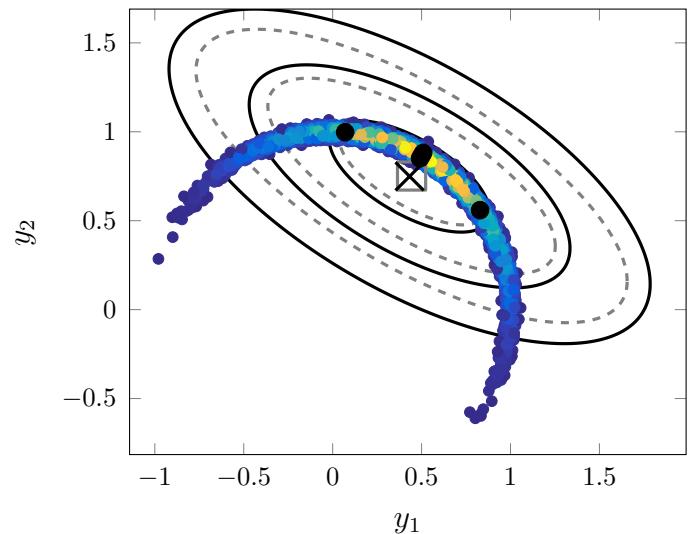
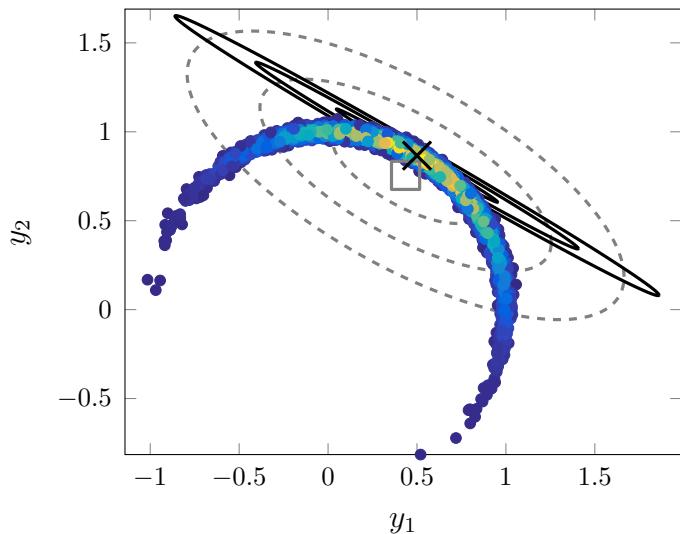
We want to have some sort of reference so that we can assess the performance of any methods that we employ to approximate the statistics of \boldsymbol{y} . To produce this reference, we use Monte Carlo sampling by drawing a set of N samples from a Gaussian distribution with mean \boldsymbol{m}_x and covariance \boldsymbol{P}_{xx} . For each sample, $\boldsymbol{x}^{(i)}$, we apply the nonlinear function and generate a set of N transformed samples, $\boldsymbol{y}^{(i)}$. Using these samples, we compute the “true” mean and covariance of \boldsymbol{y} as the sample mean and sample covariance. This gives us a point of comparison for other approximation methods. A set of 5,000 samples of \boldsymbol{x} and the corresponding set of transformed samples in \boldsymbol{y} are illustrated in the next figures.



Linearization and the unscented transform are used to compute the approximate mean and covariance. We apply linearization in the usual way, and we apply the UT with $\alpha = 0.5$, $\kappa = 6$, and $\beta = 2$.

For each transformation method – Monte Carlo, linearization, and the UT – the 1, 2,

and 3σ curves are plotted. This is a Gaussian illustration, even though the resulting pdf is clearly non-Gaussian. This is simply to illustrate differences in mean and covariance approximations and, since the Gaussian distribution is fully characterized by a mean and covariance, it makes a convenient visualization tool.



In the figures, the dashed curves belong to the Monte Carlo sample mean and covariance, and the solid lines belong to the mean and covariance resulting from linearization (left)

and the UT (right). Furthermore, their corresponding means are marked by a square for the Monte Carlo method and an \times for linearization and the UT. For the right figure, we also have the $2n + 1 = 5$ sigma points drawn as black dots.

In this case, the UT outperforms linearization in both the transformed mean *and* covariance in the sense that it is closer to the Monte Carlo results. Note, however, that this is not to say it will *always* outperform linearization. This problem is simply an illustration.

We now have the UT to approximate the mean, covariance, and cross-covariance for a transformation of the form

$$\mathbf{y} = \mathbf{g}(\mathbf{x})$$

where the mean and covariance of the input are known. Unfortunately, this doesn't quite match up to our filtering problems, since there's no noise in the transformation. Our filtering problem involves a nonlinear transformation subjected to additive noise, i.e., a transformation of the form

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \mathbf{n}$$

where the noise is represented by \mathbf{n} and is often taken to be zero mean with covariance \mathbf{P}_{nn} . This model holds for both our dynamical system with \mathbf{n} representing the process noise and for our observational system with \mathbf{n} representing the measurement noise. Remember that we do not require this noise to be Gaussian distributed in general, but it is an assumption we will make as it affords a convenient and straightforward explanation.

We can fairly easily leverage the UT to compute the statistics in the presence of additive noise. When the noise is additive and uncorrelated to the state, we have that

$$\mathbf{m}_y = \int \mathbf{g}(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}$$

$$\mathbf{P}_{yy} = \int (\mathbf{g}(\mathbf{x}) - \mathbf{m}_y)(\mathbf{g}(\mathbf{x}) - \mathbf{m}_y)^T p(\mathbf{x}) d\mathbf{x} + \mathbf{P}_{nn}$$

$$\mathbf{P}_{xy} = \int (\mathbf{x} - \mathbf{m}_x)(\mathbf{g}(\mathbf{x}) - \mathbf{m}_y)^T p(\mathbf{x}) d\mathbf{x}$$

We've already developed the method for estimating each of the preceding integrals, so we can directly state the result of the UT in the presence of additive noise. The first step is to draw our state sigma points, $\mathcal{X}^{(i)}$, and to determine the mean and covariance weights

$w_i^{(m)}$ and $w_i^{(c)}$, for $i = 0, \dots, 2n$. We then determine the transformed sigma points as

$$\mathbf{y}^{(i)} = \mathbf{g}(\mathbf{x}^{(i)}) , \quad i = 0, \dots, 2n$$

Finally, we compute the mean, covariance, and cross-covariance as

$$\mathbf{m}_y \approx \sum_{i=0}^{2n} w_i^{(m)} \mathbf{y}^{(i)}$$

$$\mathbf{P}_{yy} \approx \sum_{i=0}^{2n} w_i^{(c)} (\mathbf{y}^{(i)} - \mathbf{m}_y) (\mathbf{y}^{(i)} - \mathbf{m}_y)^T + \mathbf{P}_{nn}$$

$$\mathbf{P}_{xy} \approx \sum_{i=0}^{2n} w_i^{(c)} (\mathbf{x}^{(i)} - \mathbf{m}_x) (\mathbf{y}^{(i)} - \mathbf{m}_y)^T$$

In a Kalman filtering sense, this enables us to perform both propagation (or prediction) and measurement updates (or correction) in the presence of additive noise. Recall that we need the mean and covariance equations for propagation and that we need the mean, covariance, and cross-covariance equations for the update.

At this point, we have all of the tools required to formulate the unscented Kalman filter for a system with discrete, nonlinear dynamics and measurements that is subjected to additive noise, i.e., a system of the form

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k$$

where process noise and measurement noise are both zero-mean, white-noise sequences that are both uncorrelated with the state.

The unscented Kalman filter works within the Kalman framework to propagate and update the mean and covariance for the state by using the unscented transform to approximate the mean, covariance, and cross-covariance (with the input) of the output of the nonlinear transformations. Given initial values of the mean and covariance, $\mathbf{m}_{x,0}$ and $\mathbf{P}_{xx,0}$, along with a sequence of data, \mathbf{z}_k , for $k = 1, 2, \dots$, the objective is to sequentially propagate the mean and covariance between measurements and update the mean and covariance using the measurement data.

For the propagation stage, we want to propagate the mean $\mathbf{m}_{x,k-1}^+$ and covariance $\mathbf{P}_{xx,k-1}^+$ from t_{k-1} to t_k . This begins by computing a SRF of $\mathbf{P}_{xx,k-1}^+$ that is given by $\mathbf{S}_{xx,k-1}^+$. We use this mean and SRF to generate a set of sigma points for $i = 0, 1, \dots, 2n$ as

$$\boldsymbol{\chi}_{k-1}^{(0)} = \mathbf{m}_{x,k-1}^+$$

$$\boldsymbol{\chi}_{k-1}^{(i)} = \mathbf{m}_{x,k-1}^+ + \sqrt{n + \lambda} [\mathbf{S}_{xx,k-1}^+]_i$$

$$\boldsymbol{\chi}_{k-1}^{(i+n)} = \mathbf{m}_{x,k-1}^+ - \sqrt{n + \lambda} [\mathbf{S}_{xx,k-1}^+]_i$$

which are accompanied by weights

$$w_0^{(m)} = \frac{\lambda}{n + \lambda} \quad w_0^{(c)} = \frac{\lambda}{n + \lambda} + (1 - \alpha^2 + \beta)$$

$$w_i^{(m)} = \frac{1}{2(n + \lambda)} \quad w_i^{(c)} = \frac{1}{2(n + \lambda)}$$

where $n + \lambda = \alpha^2(n + \kappa)$. Each sigma point is propagated via the dynamics to yield a set of transformed sigma points as

$$\boldsymbol{\chi}_k^{(i)} = \mathbf{f}(\boldsymbol{\chi}_{k-1}^{(i)})$$

which are then used to compute the *a prior* mean and covariance

$$\mathbf{m}_{x,k}^- = \sum_{i=0}^{2n} w_i^{(m)} \boldsymbol{\chi}_k^{(i)}$$

$$\mathbf{P}_{xx,k}^- = \sum_{i=0}^{2n} w_i^{(c)} (\boldsymbol{\chi}_k^{(i)} - \mathbf{m}_{x,k}^-) (\boldsymbol{\chi}_k^{(i)} - \mathbf{m}_{x,k}^-)^T + \mathbf{P}_{ww,k-1}$$

Note that the propagation stage of the UKF is dependent on our selection of the parameters α , κ , and β .

For the update stage, we want to fuse new information contained in \mathbf{z}_k with our prior mean, $\mathbf{m}_{x,k}^-$ and covariance $\mathbf{P}_{xx,k}^-$. Computing a SRF of $\mathbf{P}_{xx,k}^-$ that is given by $\mathbf{S}_{xx,k}^-$ enables us to generate a set of prior sigma points for $i = 0, 1, \dots, 2n$ as

$$\boldsymbol{\chi}_k^{(0)} = \mathbf{m}_{x,k}^-$$

$$\boldsymbol{\chi}_k^{(i)} = \mathbf{m}_{x,k}^- + \sqrt{n + \lambda} [\mathbf{S}_{xx,k}^-]_i$$

$$\boldsymbol{\chi}_k^{(i+n)} = \mathbf{m}_{x,k}^- - \sqrt{n + \lambda} [\mathbf{S}_{xx,k}^-]_i$$

and associated mean and covariance weights

$$\begin{aligned} w_0^{(m)} &= \frac{\lambda}{n + \lambda} & w_0^{(c)} &= \frac{\lambda}{n + \lambda} + (1 - \alpha^2 + \beta) \\ w_i^{(m)} &= \frac{1}{2(n + \lambda)} & w_i^{(c)} &= \frac{1}{2(n + \lambda)} \end{aligned}$$

where $n + \lambda = \alpha^2(n + \kappa)$. Each of the prior sigma points is transformed via the nonlinear measurement function as

$$\mathbf{z}_k^{(i)} = \mathbf{h}(\mathbf{x}_k^{(i)})$$

from which the predicted measurement, innovation covariance, and cross-covariance are approximated as

$$\begin{aligned} \mathbf{m}_{z,k}^- &= \sum_{i=0}^{2n} w_i^{(m)} \mathbf{z}^{(i)} \\ \mathbf{P}_{zz,k}^- &= \sum_{i=0}^{2n} w_i^{(c)} (\mathbf{z}^{(i)} - \mathbf{m}_{z,k}^-)(\mathbf{z}^{(i)} - \mathbf{m}_{z,k}^-)^T + \mathbf{P}_{vv,k} \end{aligned}$$

$$\mathbf{P}_{xz,k}^- = \sum_{i=0}^{2n} w_i^{(c)} (\mathcal{X}_k^{-(i)} - \mathbf{m}_{x,k}^-)(\mathcal{Z}^{(i)} - \mathbf{m}_{z,k}^-)^T$$

Finally, we can perform the update as

$$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{m}_{z,k}^-)$$

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- \mathbf{K}_k^T - \mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

where the Kalman gain is given by

$$\mathbf{K}_k = \mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1}$$

As with the propagation stage, the update stage of the UKF depends on our selection of the parameters α , κ , and β . These choices can be the same for each stage or they can be different. They can also be constant in time or change in time. Most often, the same set of parameters is used for both propagation and update, and these parameters are held constant.

It might be tempting in the update stage to re-use the propagated sigma points from the

propagation stage. In general, this is not permissible. The *a priori* sigma points have to be generated from the prior mean and covariance in order to include the effects of the process noise. In the special case where there is no process noise, however, it is possible to re-use the propagated sigma points as the *a priori* sigma points.

One advantage of the UKF over methods such as the EKF is that it is not based on a linear approximation at a single point, but instead uses additional points in approximating the nonlinearity. Additionally, the UKF does not require that derivatives of the system dynamics and the measurement model be taken, whereas the EKF does require these derivatives. This means that the UKF can be applied to non-differentiable systems. One disadvantage of the UKF when compared to the EKF is that it requires slightly more computational operations than the EKF.

In our treatment of the UT and the UKF so far, we have only considered additive noise models, but we know that the more general case is a transformation of the form

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{n})$$

where we assume that we still know the means and covariances of the inputs, \mathbf{x} and \mathbf{n} . In this case, we still want to know the mean, covariance, and cross-covariance (with \mathbf{x})

of \mathbf{y} .

One of the beneficial things about the UT is it allows us to fairly easily extend our treatment of the noise. We can simply augment the state with the noise to form an augmented state of the form

$$\mathbf{x} = \begin{bmatrix} \mathbf{x} \\ \mathbf{n} \end{bmatrix}$$

where the mean and covariance of \mathbf{x} are \mathbf{m}_x and \mathbf{P}_{xx} and the mean and covariance of \mathbf{n} are $\mathbf{0}$ and \mathbf{P}_{nn} . If we assume that the noise is independent of the state, then the mean and covariance of the augmented state are

$$\mathbf{m}_x = \begin{bmatrix} \mathbf{m}_x \\ \mathbf{0} \end{bmatrix} \quad \text{and} \quad \mathbf{P}_{xx} = \begin{bmatrix} \mathbf{P}_{xx} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_{nn} \end{bmatrix}$$

It is worth noting that non-zero mean or non-independent noises can easily be handled in this augmented formulation. With the augmented input, our transformation takes the form

$$\mathbf{y} = \mathbf{g}(\mathbf{x})$$

We can now use our previously developed UT approach to compute the mean, covariance, and cross-covariance of \mathbf{y} . The only difference is in the number of sigma points and in determining the cross-covariance between \mathbf{x} and \mathbf{y} , since we have an augmented input to deal with.

Let the dimension of \mathbf{x} be n_x and the dimension of \mathbf{n} be n_n . Then, the dimension of \mathbf{x} is $n_x = n_x + n_n$. We first determine an SRF for the augmented covariance that is given by \mathbf{S}_{xx} . Next, we generate a set of $2n_x + 1$ sigma point for the augmented state as

$$\boldsymbol{\chi}_{\text{aug}}^{(0)} = \mathbf{m}_x$$

$$\boldsymbol{\chi}_{\text{aug}}^{(i)} = \mathbf{m}_x + \sqrt{n_x + \lambda_{\text{aug}}} [\mathbf{S}_{\text{xx}}]_i$$

$$\boldsymbol{\chi}_{\text{aug}}^{(i+n_x)} = \mathbf{m}_x - \sqrt{n_x + \lambda_{\text{aug}}} [\mathbf{S}_{\text{xx}}]_i$$

where

$$n_x + \lambda_{\text{aug}} = \alpha_{\text{aug}}^2 (n_x + \kappa_{\text{aug}})$$

Associated with each of the sigma points is also the set of mean and covariance weights, which are computed as before and are now represented symbolically by $w_{i,\text{aug}}^{(m)}$ and $w_{i,\text{aug}}^{(c)}$,

for $i = 0, \dots, 2n_x$. We use the parameters α_{aug} , κ_{aug} , and β_{aug} to determine the weights and sigma points. Note that the subscript “aug” is used to indicate that we are working with the augmented input, but there is no direct augmentation that occurs with these selected parameters.

Once we have the sigma points for the augmented input, we separate the augmented sigma points into the parts that correspond to \mathbf{x} and \mathbf{n} , i.e.,

$$\boldsymbol{\chi}_{\text{aug}}^{(i)} = \begin{bmatrix} \boldsymbol{\chi}_x^{(i)} \\ \boldsymbol{\chi}_n^{(i)} \end{bmatrix}$$

We can then use these separated sigma points to transform the sigma points through the nonlinear function as

$$\boldsymbol{\gamma}^{(i)} = \mathbf{g}(\boldsymbol{\chi}_x^{(i)}, \boldsymbol{\chi}_n^{(i)}) , \quad \text{for } i = 0, 1, \dots, 2n_{\text{aug}}$$

The final step is then to compute the approximate mean, covariance, and cross-covariance as

$$\mathbf{m}_y \approx \sum_{i=0}^{2n_{\text{aug}}} w_{i,\text{aug}}^{(m)} \boldsymbol{\gamma}^{(i)}$$

$$\mathbf{P}_{yy} \approx \sum_{i=0}^{2n_{\text{aug}}} w_{i,\text{aug}}^{(c)} (\mathcal{Y}^{(i)} - \mathbf{m}_y)(\mathcal{Y}^{(i)} - \mathbf{m}_y)^T$$

$$\mathbf{P}_{xy} \approx \sum_{i=0}^{2n_{\text{aug}}} w_{i,\text{aug}}^{(c)} (\mathcal{X}_x^{(i)} - \mathbf{m}_x)(\mathcal{Y}^{(i)} - \mathbf{m}_y)^T$$

Note that there is no addition of \mathbf{P}_{nn} into \mathbf{P}_{yy} . This is because that effect is captured by the transformation of the separated sigma points through the nonlinear function. Note also that the computation of \mathbf{P}_{xy} only depends on the part of the generated sigma points that correspond to the original state, \mathbf{x} .

We have previously focused on the form of the UKF when we have both additive process noise and measurement noise. As we now know, it is straightforward to treat non-additive noises and to treat state-correlated noises via the UT. We can take these developments and directly apply them within the UKF to formulate a UKF that operates on these types of models. The specifics will not be detailed, but it essentially just comes down to augmenting the state vector and proceeding in the usual fashion. Note, however, that augmenting the state vector increases the number of sigma points used in the UKF and thus increases the computational complexity. When additive noises are present, it

is more computationally efficient to use the additive noise form of the UKF instead of augmenting the state.

An advantage that is commonly discussed when applying the UKF is the accuracy of the method. The unscented transform is a third-order method in the sense that the estimate of the mean of $\mathbf{g}(\mathbf{x})$ is exact for polynomials up to order three. However, the covariance approximation is exact only for first-order polynomials, because the square of a second-order polynomial is already a polynomial of order four, and the UT does not compute the exact result for fourth-order polynomials. In this sense, the UT is only a first-order method. With suitable selection of parameters ($\kappa = 3 - n$), it is possible to get some of the fourth-order terms appearing in the covariance computation correct also for quadratic functions, but not all of them. For more details, see Särkkä,³ or for an in-depth derivation and discussion, see van der Merwe.⁴

Our discussion of the UKF has focused on discrete-time dynamics models paired with discrete-time measurement models. For both the KF and EKF, however, we were also

³Särkkä, S., **Bayesian Filtering and Smoothing**, Cambridge, 2013.

⁴van der Merwe, R., *Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models*, Ph.D. thesis, Oregon Health and Science University, 2004.

able to formulate algorithms that worked on systems with continuous-time dynamics and discrete-time measurements. This same class of algorithms can be developed for the UKF, but some extensive modifications are required. As such, we will not explicitly develop that method here. Instead, for more details, refer to Särkkä.⁵

A different option is to consider a system in which we discretize the nonlinear, continuous-time dynamics and subject the discretization to an additive process noise. In this way, we can use the existing framework that we have constructed for the UKF, and we can implement that framework with continuous-time dynamics. The drawback to this approach is that we cannot accommodate continuous-time process noise. To see how we make this connection, consider a noiseless, continuous-time dynamical system of the form

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$$

The solution to the initial value problem is given by

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \int_{t_{k-1}}^{t_k} \mathbf{f}(\mathbf{x}(\tau)) d\tau$$

⁵Särkkä, S. “On Unscented Kalman Filtering for State Estimation of Continuous-Time Nonlinear Systems,” *IEEE Transactions on Automatic Control*, Vol. 52, No. 9, pp. 1631-1641, 2007.

That is, this is the result of solving the differential equation subject to initial condition $\mathbf{x}(t_{k-1}) = \mathbf{x}_{k-1}$. We can view this result as the discretization of the continuous-time dynamics, and we can subject it to discrete-time process noise to produce a system model as

$$\mathbf{x}_k = \left[\mathbf{x}_{k-1} + \int_{t_{k-1}}^{t_k} \mathbf{f}(\mathbf{x}(\tau)) d\tau \right] + \mathbf{w}_{k-1}$$

The term in square brackets is most often implemented as numerical integration. This is what should be returned when you integrate the continuous-time differential equations from t_{k-1} to t_k subject to the initial condition $\mathbf{x}(t_{k-1}) = \mathbf{x}_{k-1}$.

The benefit of discretizing the dynamics in this way is that we now have a system model of the form

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1}$$

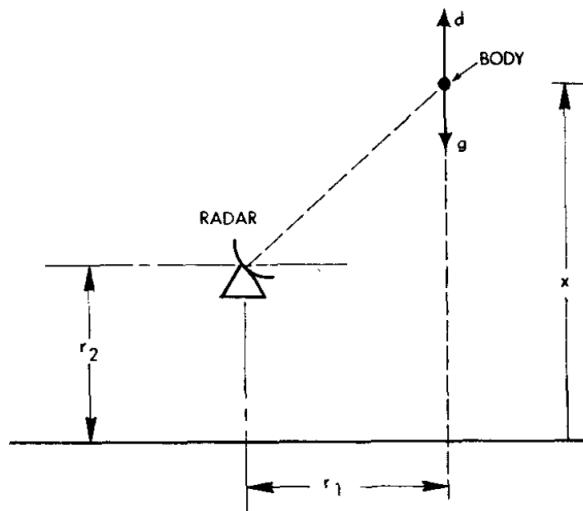
This discrete-time, noisy model for the dynamics is the same model upon which we built the UKF. Note that we are using (and have been using) \mathbf{f} to represent both continuous-time and discrete-time systems. These are different functions, but they serve the same purpose. They both move states forward in time.

5.1.1 Example of the UKF

The following example is taken and modified from Gelb (1974). We have previously looked at a similar problem, but we will use the current version as an example for applying the UKF and compare it to the EKF.

Consider the problem of tracking a body falling freely through the atmosphere. The object falls in a straight line, but a radar observing it is offset from the object horizontally by r_1 and is r_2 off the ground.

A radar measurement of the noisy distance to the object is received every 0.1 [sec], and we want to use this information to estimate the height and speed of the object, as well as its inverse ballistic coefficient. That is, we define the state as



$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x \\ \dot{x} \\ 1/\beta \end{bmatrix}$$

where x is the height of the falling body above the ground and β is the ballistic coefficient of the object.

The equations of motion for the system, assuming that the ballistic coefficient is constant, are given by

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ d - g \\ 0 \end{bmatrix} = \boldsymbol{f}(\boldsymbol{x})$$

with

$$d = \rho x_2^2 x_3 / 2 \quad \text{and} \quad \rho = \rho_0 \exp \left\{ -x_1 / k_\rho \right\}$$

where d is drag acceleration, $g = 32.2 \text{ ft/sec}^2$ is the acceleration of gravity, ρ is atmospheric density (with $\rho_0 = 3.4 \times 10^{-3} \text{ lb sec}^2/\text{ft}^4$ as the atmospheric density at sea level),

and $k_\rho = 22,000$ ft is a decay constant. The dynamics are nonlinear, but there is no process noise included in the equations of motion for this system.

Due to the tracking radar's offset, we have nonlinear measurements of range, which are modeled as

$$z = \sqrt{r_1^2 + (x_1 - r_2)^2} + v$$

where $r_1 = 1000$ ft, $r_2 = 10$ ft, and v is the zero-mean measurement noise with constant (co)variance $P_{vv} = 100$ ft 2 .

The initial mean and covariance of the state are given by

$$\mathbf{m}_{x,0} = \begin{bmatrix} 10^5 \text{ [ft]} \\ -6000 \text{ [ft/sec]} \\ 1/2000 \text{ [lb/ft}^2\text{]}^{-1} \end{bmatrix}$$

$$\mathbf{P}_{xx,0} = \begin{bmatrix} 1000 \text{ [ft}^2\text{]} & 0 & 0 \\ 0 & 2 \times 10^3 \text{ [ft}^2/\text{sec}^2\text{]} & 0 \\ 0 & 0 & 1/(2.5 \times 10^5) \text{ [lb}^2/\text{ft}^4\text{]}^{-1} \end{bmatrix}$$

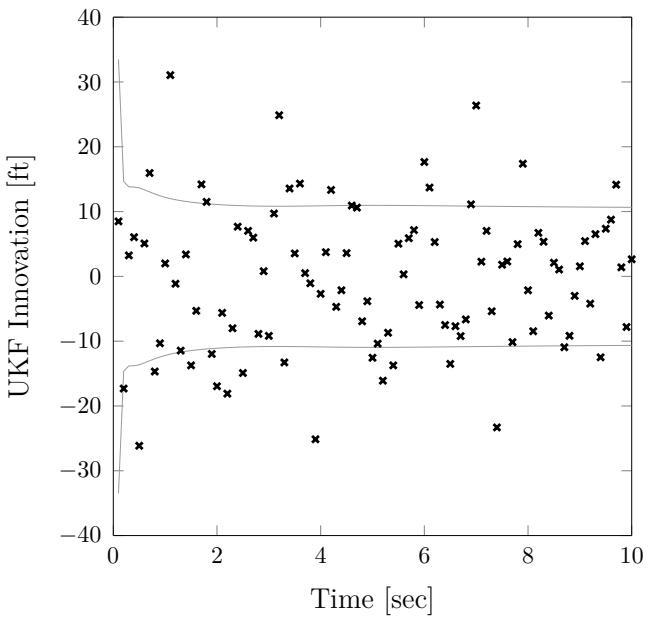
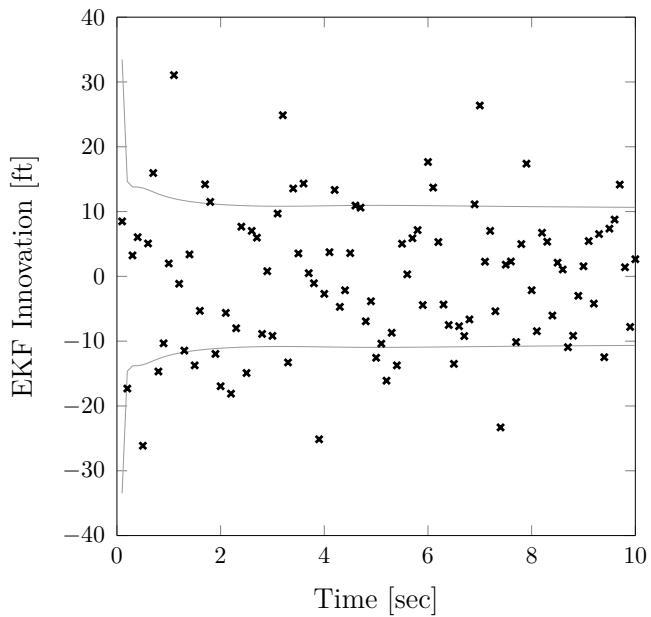
The initial true state is generated using a Gaussian distribution for \mathbf{x}_0 with mean $\mathbf{m}_{x,0}$ and covariance $\mathbf{P}_{xx,0}$. Samples of the measurement noise are generated using a Gaussian distribution with a mean of zero and a variance of P_{vv} .

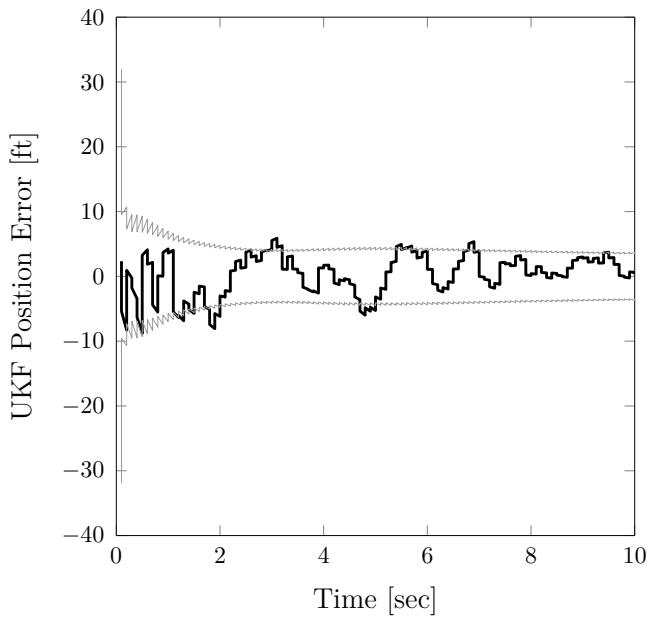
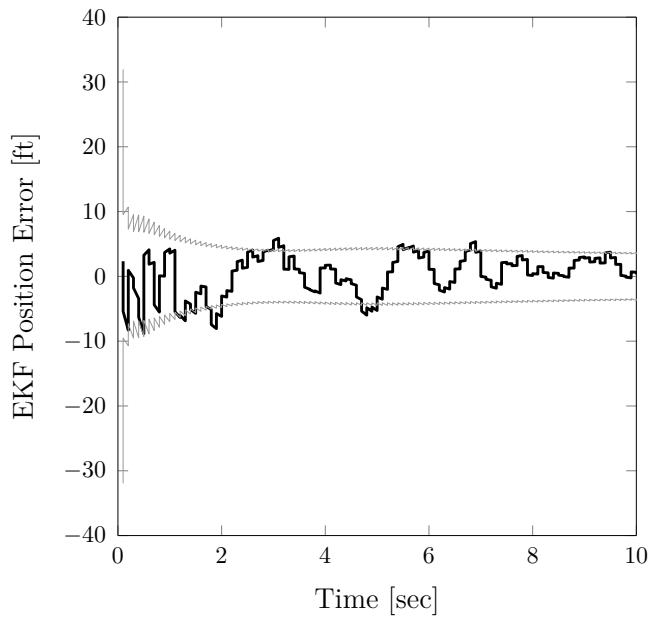
Both the EKF and the UKF are applied to this problem to estimate the state of the system using range measurements.

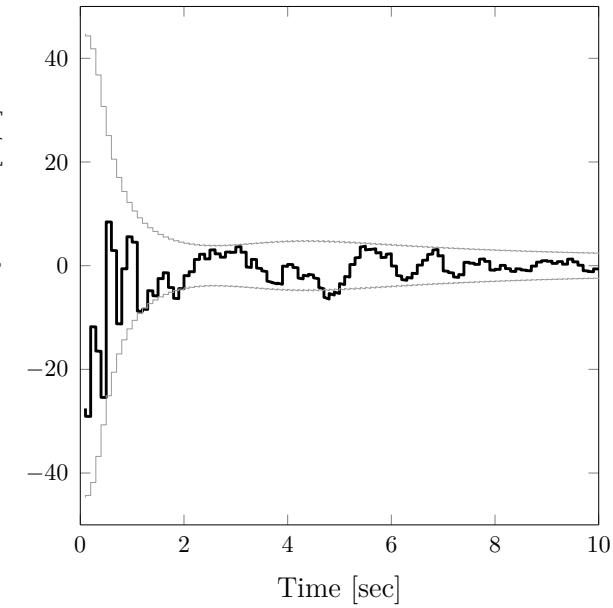
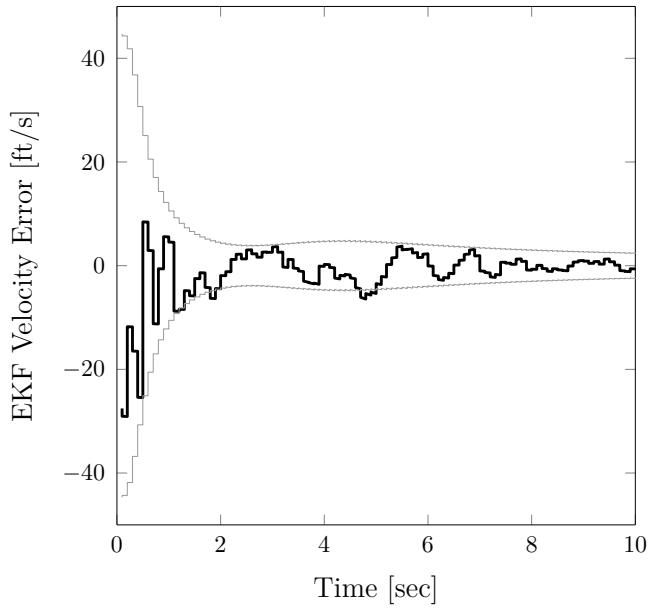
The EKF is applied in the usual fashion, using numerical integration of the mean and covariance for the propagation and using the generalized Joseph form of the covariance update.

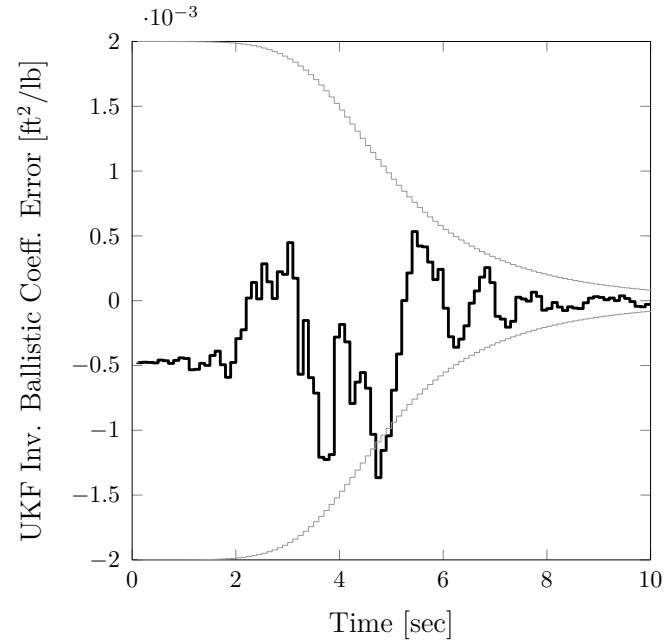
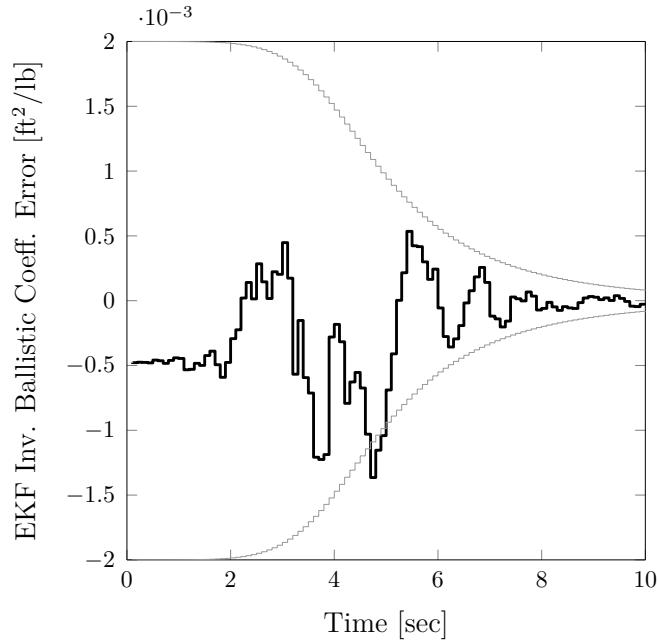
The UKF is implemented using Cholesky decomposition for the SRFs. Both the propagation and update stages use parameters of $\alpha = 0.5$, $\kappa = 0$, and $\beta = 2$. Note that this β is not the ballistic coefficient. The propagation stage uses numerical integration to propagate sigma points between t_{k-1} and t_k , and the update stage makes use of the generalized Joseph form of the covariance update.

The following figures provide the innovations and state estimation errors, along with their corresponding 1σ intervals for both the EKF and UKF.









Note how similar the plots are. For the innovations and every channel of the state estimation error, the results are nearly identical for this problem. Unlike our polar-to-Cartesian conversion example, the EKF (using linearization) closely matches the results of the UKF (using the unscented transform).

5.2 The Gauss-Hermite Kalman Filter

Shortly after the UKF was published, Ito and Xiong⁶ pointed out that the UKF can be considered a special case of so-called “Gaussian filters.” In these Gaussian filters, the nonlinear filtering problem is solved using Gaussian assumed density approximations. Essentially, where we used generic pdfs, $p(\mathbf{x})$, in the Kalman framework for evaluating expectation integrals, we will now make the assumed density approximation that these pdfs are actually Gaussian, $p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx})$, with some specified mean and covariance.

If we go back to the Kalman filter framework that we defined for systems of the form

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k$$

where the process noise and measurement noise are both taken to be zero-mean, white-noise sequences that are uncorrelated with the state, we obtain the so-called Gaussian filter by replacing the generic pdfs with Gaussian pdfs, such that the expectation integrals

⁶Ito, K. and Xiong, K., “Gaussian Filters for Nonlinear Filtering Problems,” *IEEE Transactions on Automatic Control*, Vol. 45, No. 5, May 2000, pp. 910–927.

become

$$\mathbf{m}_{x,k}^- = \int \mathbf{f}(\mathbf{x}_{k-1}) p_g(\mathbf{x}_{k-1}; \mathbf{m}_{x,k-1}^+, \mathbf{P}_{xx,k-1}^+) d\mathbf{x}_{k-1}$$

$$\mathbf{P}_{xx,k}^- = \int (\mathbf{f}(\mathbf{x}_{k-1}) - \mathbf{m}_{x,k}^-)(\mathbf{f}(\mathbf{x}_{k-1}) - \mathbf{m}_{x,k}^-)^T p_g(\mathbf{x}_{k-1}; \mathbf{m}_{x,k-1}^+, \mathbf{P}_{xx,k-1}^+) d\mathbf{x}_{k-1} + \mathbf{P}_{ww,k-1}$$

$$\mathbf{m}_{z,k}^- = \int \mathbf{h}(\mathbf{x}_k) p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^-, \mathbf{P}_{xx,k}^-) d\mathbf{x}_k$$

$$\mathbf{P}_{xz,k}^- = \int (\mathbf{x}_k - \mathbf{m}_{x,k}^-)(\mathbf{h}(\mathbf{x}_k) - \mathbf{m}_{z,k}^-)^T p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^-, \mathbf{P}_{xx,k}^-) d\mathbf{x}_k$$

$$\mathbf{P}_{zz,k}^- = \int (\mathbf{h}(\mathbf{x}_k) - \mathbf{m}_{z,k}^-)(\mathbf{h}(\mathbf{x}_k) - \mathbf{m}_{z,k}^-)^T p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^-, \mathbf{P}_{xx,k}^-) d\mathbf{x}_k + \mathbf{P}_{vv,k}$$

That is, we now explicitly represent the pdfs as Gaussian pdfs using the posterior at t_{k-1} and the prior at t_k . The rest of the Kalman framework still applies. We use a linear gain, such as the Kalman gain given by

$$\mathbf{K}_k = \mathbf{P}_{xz,k}^- (\mathbf{P}_{zz,k}^-)^{-1}$$

to update the mean and covariance of the state as

$$\boldsymbol{m}_{x,k}^+ = \boldsymbol{m}_{x,k}^- + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{m}_{z,k}^-)$$

$$\boldsymbol{P}_{xx,k}^+ = \boldsymbol{P}_{xx,k}^- - \boldsymbol{P}_{xz,k}^- \boldsymbol{K}_k^T - \boldsymbol{K}_k (\boldsymbol{P}_{xz,k}^-)^T + \boldsymbol{K}_k \boldsymbol{P}_{zz,k}^- \boldsymbol{K}_k^T$$

The advantage to formulating the expectation integrals with the Gaussian assumption is that we can make use of various well-studied and powerful quadrature and cubature methods to approximate the integrals. This expands our derivative-free approaches beyond the unscented transform.

Under the assumed density framework, the integrals become what are known as Gaussian integrals, each of which takes the form

$$\int \boldsymbol{g}(\boldsymbol{x}) p_g(\boldsymbol{x}; \boldsymbol{m}_x, \boldsymbol{P}_{xx}) d\boldsymbol{x}$$

We will consider a nonlinear transformation of the form

$$\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{n}$$

where $\mathbf{g}(\mathbf{x})$ can be used to represent either the dynamical system or the measurement relationship. Given the inputs to the transformation to be Gaussian-distributed as

$$\mathbf{x} \sim p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) \quad \text{and} \quad \mathbf{n} \sim p_g(\mathbf{n}; \mathbf{0}, \mathbf{P}_{nn})$$

the moment matching based Gaussian approximation to the joint distribution of \mathbf{x} leads to a joint Gaussian distribution between \mathbf{x} and \mathbf{y} as is given by

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \sim p_g\left(\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}; \begin{bmatrix} \mathbf{m}_x \\ \mathbf{m}_y \end{bmatrix}, \begin{bmatrix} \mathbf{P}_{xx} & \mathbf{P}_{xy} \\ \mathbf{P}_{xy}^T & \mathbf{P}_{yy} \end{bmatrix}\right)$$

where the mean, covariance, and cross-covariance are

$$\mathbf{m}_y = \int \mathbf{g}(\mathbf{x}) p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) d\mathbf{x}$$

$$\mathbf{P}_{yy} = \int (\mathbf{g}(\mathbf{x}) - \mathbf{m}_y)(\mathbf{g}(\mathbf{x}) - \mathbf{m}_y)^T p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) d\mathbf{x} + \mathbf{P}_{nn}$$

$$\mathbf{P}_{xy} = \int (\mathbf{x} - \mathbf{m}_x)(\mathbf{g}(\mathbf{x}) - \mathbf{m}_y)^T p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) d\mathbf{x}$$

Implicit in the preceding result is that \mathbf{n} is additive, zero-mean, and uncorrelated with \mathbf{x} .

If we use these moment matching relationships to approximate a filtering algorithm, we get the assumed density filter (ADF) or what is called the Gaussian filter by some. That is, every propagation and update step assumes that the state distribution is Gaussian. This is not an assumption we have really made before, but it is now necessary in order to create rules for handling the integrations required for the mean, covariance, and cross-covariance calculations.

What do we gain by this assumed density framework? The advantage to such a moment matching formulation is that it enables the use of many well-known numerical integration techniques. These techniques include, but are certainly not limited to, Gauss-Hermite quadrature and cubature rules, Bayes-Hermite quadrature, or even Monte Carlo integration to approximate the required integrals.

As currently expressed, the Gaussian assumed density filter is a theoretical construct; we need to formulate numerical techniques to approximate the integrals of interest and construct a practical filter. One such way to do this is to use the Gauss-Hermite integration rule.

We are interested in approximating Gaussian integrals of the form

$$\begin{aligned} & \int \mathbf{g}(\mathbf{x}) p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) d\mathbf{x} \\ &= |2\pi \mathbf{P}_{xx}|^{-1/2} \int \mathbf{g}(\mathbf{x}) \exp \left\{ -\frac{1}{2} (\mathbf{x} - \mathbf{m}_x)^T \mathbf{P}_{xx}^{-1} (\mathbf{x} - \mathbf{m}_x) \right\} d\mathbf{x} \end{aligned}$$

where $\mathbf{g}(\mathbf{x})$ is some arbitrary function and we have applied the definition of a Gaussian pdf. Notably, this is a vector integral, and the Gauss-Hermite rule is a scalar integration rule. As such, we will take one-dimensional Gauss-Hermite integration and generalize it into multiple dimensions.

Formally, this is done by taking the Cartesian product of one-dimensional quadratures in each direction. Unfortunately, this approach is no stranger to the *curse of dimensionality*; the required number of evaluation points is exponential with respect to number of dimensions. This occurs because the volume of the space defined by the Cartesian product of one-dimensional spaces rapidly grows as the number of products increases, meaning that many more points are required to characterize features within that space.

One-dimensional Gauss-Hermite quadrature refers directly to the special case of Gaussian quadratures with a unit Gaussian weight function to form an approximation of the form

$$\int_{-\infty}^{\infty} g(x) p_g(x; 0, 1) dx \approx \sum_i w^{(i)} g(x^{(i)})$$

where $w^{(i)}$ and $x^{(i)}$ are the weights and evaluation points respectively for $i = 1, \dots, p$. This is sometimes termed as the “probabilist’s definition” of Gaussian quadrature as it employs the normalized version of the Gaussian function.

Gaussian functions are a class of functions that are exponential with a negative quadratic argument. We refer to the normalized version (the version that integrates to one) when we say Gaussian, and this is the probabilist’s definition of the Gaussian function.

There are an infinite number of ways to select the weights and evaluation points, so we must pick rules to define the selection of these parameters. In Gauss-Hermite quadrature, the weights and point locations are chosen such that the integration becomes exact with a polynomial integrand.

It turns out that selecting the p^{th} -order Hermite polynomial, denoted $H_p(x)$, results in an integral approximation which is exact for polynomials up to order $2p - 1$ and permits for weights to be computed in closed-form. The Hermite polynomial of order p is defined as (these are the probabilist's polynomials)

$$H_p(x) = (-1)^p \exp\left\{\frac{x^2}{2}\right\} \frac{d^p}{dx^p} \exp\left\{\frac{-x^2}{2}\right\}$$

Plugging in a few values of p gives the first few Hermite polynomials to be

$$H_0(x) = 1 \quad H_3(x) = x^3 - 3x$$

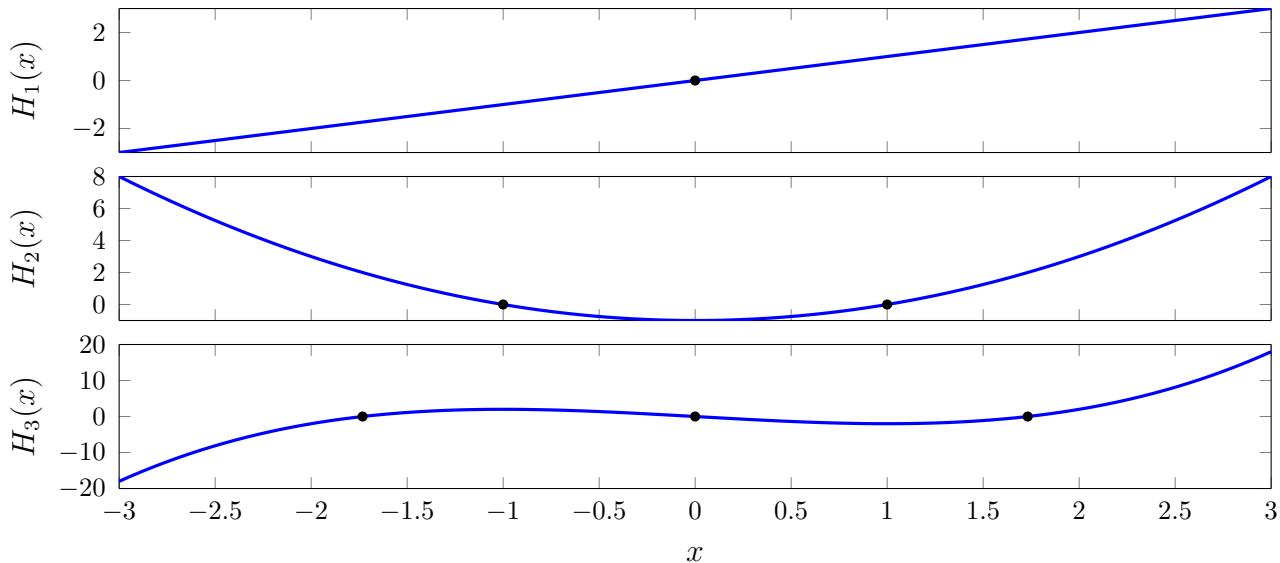
$$H_1(x) = x \quad H_4(x) = x^4 - 6x^2 + 3$$

$$H_2(x) = x^2 - 1 \quad H_5(x) = x^5 - 10x^3 + 15x$$

Further polynomials are readily obtained via the recursion

$$H_{p+1}(x) = xH_p(x) - pH_{p-1}(x)$$

The first three of these polynomials, including the roots of the polynomials, are visualized as



Using the same weights and evaluation points, integrals over non-unit Gaussian weight functions $p_g(x; m_x, P_{xx})$ can be evaluated using a simple change of integration variable:

$$\int_{-\infty}^{\infty} g(x)p_g(x; m_x, P_{xx})dx = \int_{-\infty}^{\infty} g(P_{xx}^{1/2}\xi + m_x)p_g(\xi; 0, 1)d\xi$$

The process for Gauss-Hermite quadrature of the general one-dimensional integral is

1. Compute the unit evaluation points as the roots $\xi^{(i)}$ for $i = 1, \dots, p$ of the Hermite polynomial $H_p(x)$. It is actually not required to form the polynomial to find the roots of the polynomial; instead, it is more numerically stable to compute the roots as eigenvalues of a suitable tridiagonal companion matrix.⁷
2. Compute the weights as

$$w^{(i)} = p! / p^2 H_{p-1}^2(\xi^{(i)})$$

3. Approximate the integral as

$$\int_{-\infty}^{\infty} g(x) p_g(x; m_x, P_{xx}) dx \approx \sum_{i=1}^p w^{(i)} g(P^{1/2} \xi^{(i)} + m)$$

Approximating multi-dimensional integrals requires a generalization of the change of variables. Let $P_{xx} = S_{xx} S_{xx}^T$, where S_{xx} is a square-root factor (e.g., the Cholesky factor) of P_{xx} . If we define a new vector integration variable ξ according to

$$\mathbf{x} = \mathbf{m}_x + \mathbf{S}_{xx} \boldsymbol{\xi}$$

⁷Golub, G.H. and Welsch, J.H., “Calculation of Gauss Quadrature Rules,” *Mathematics of Computation*, Vol. 23, No. 106, 1969, pp. 221–230.

we can perform a change of variables to express the multi-dimensional integral as

$$\int \mathbf{g}(\mathbf{x}) p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) d\mathbf{x} = \int \mathbf{g}(\mathbf{m}_x + \mathbf{S}_{xx}\boldsymbol{\xi}) p_g(\boldsymbol{\xi}; \mathbf{0}_n, \mathbf{I}_n) d\boldsymbol{\xi}$$

where n is the dimension of \mathbf{x} . The multi-dimensional Gaussian distribution with zero mean and identity covariance can be written as the product of n univariate Gaussian distributions, each with zero mean and unit variance, such that the multi-dimensional integral can be expressed as the product of n integrals, or

$$\int \mathbf{g}(\mathbf{x}) p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) d\mathbf{x} = \int \cdots \int \mathbf{g}(\mathbf{m}_x + \mathbf{S}_{xx}\boldsymbol{\xi}) p_g(\xi_1; 0, 1) d\xi_1 \cdots p_g(\xi_n; 0, 1) d\xi_n$$

Then, each of the one-dimensional integrals can be approximated with Gauss-Hermite quadrature, which produces an approximation of the form

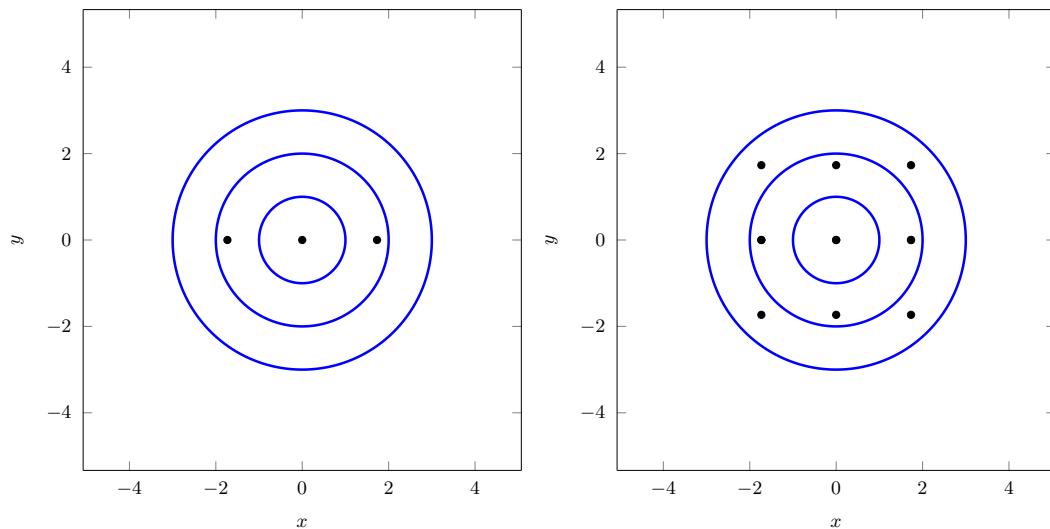
$$\int \mathbf{g}(\mathbf{x}) p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) d\mathbf{x} \approx \sum_{i_1, \dots, i_n} w^{(i_1)} \times \cdots \times w^{(i_n)} \mathbf{g}(\mathbf{m} + \mathbf{S}\boldsymbol{\xi}^{(i_1, \dots, i_n)})$$

The weights $w^{(i_k)}$ for $k = 1, \dots, n$ are simply the corresponding one-dimensional Gauss-Hermite weights and $\boldsymbol{\xi}^{(i_1, \dots, i_n)}$ in an n -dimensional vector with one-dimensional unit evaluation point $\xi^{(i_k)}$ at element k . For compactness, we often use the same multi-index

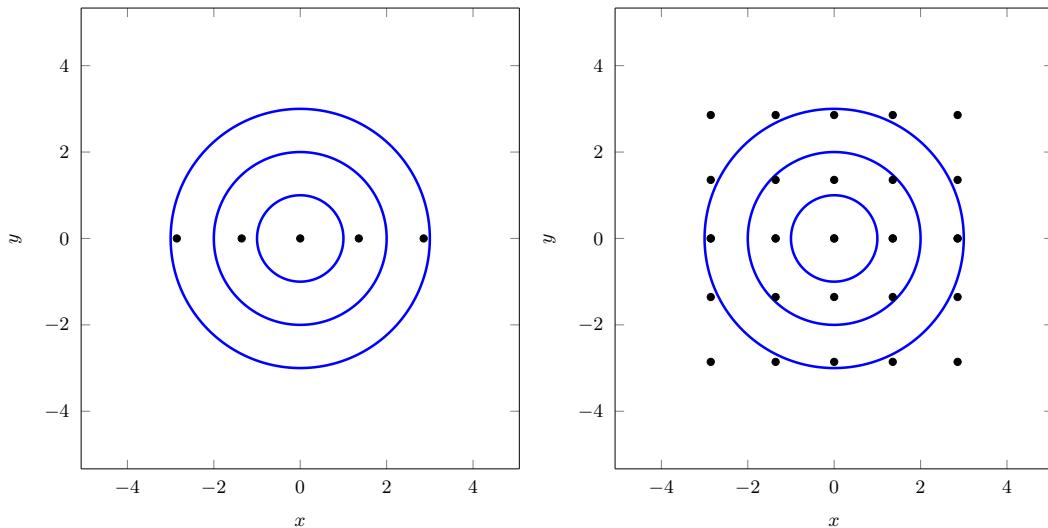
notation for the weights that we use for the quadrature points, such that

$$w^{(i_1, \dots, i_n)} = w^{(i_1)} \times \cdots \times w^{(i_n)}$$

The following figure illustrates the quadrature point generation in higher dimensions, where a $p = 3$ rule is extended from one dimension (left) to two dimensions (right).



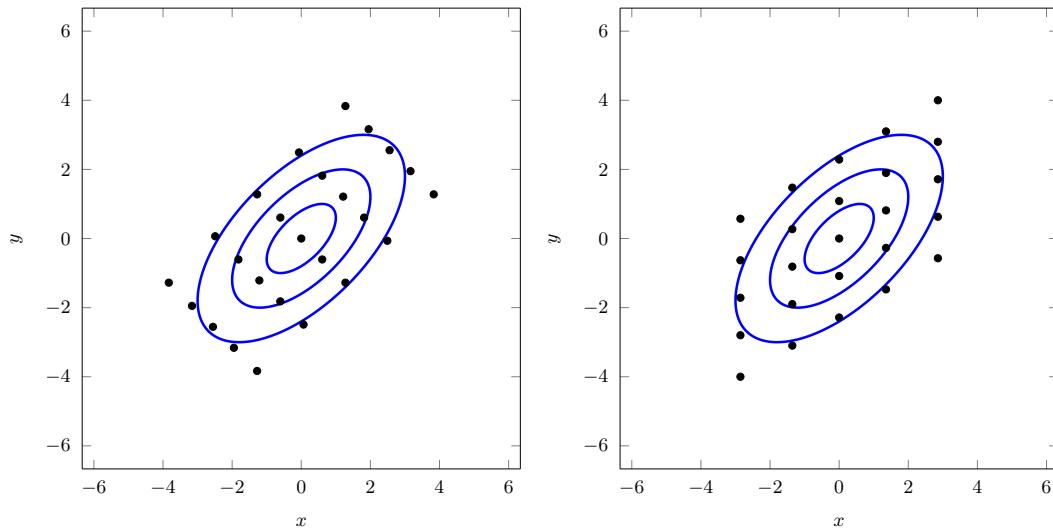
Essentially, the quadrature rule is established for each dimension, and all combinations are taken across the dimensions to establish the complete quadrature rule. As another example, the following figure illustrates the process for a 5th-order rule extended from one dimension (left) to two dimensions (right).



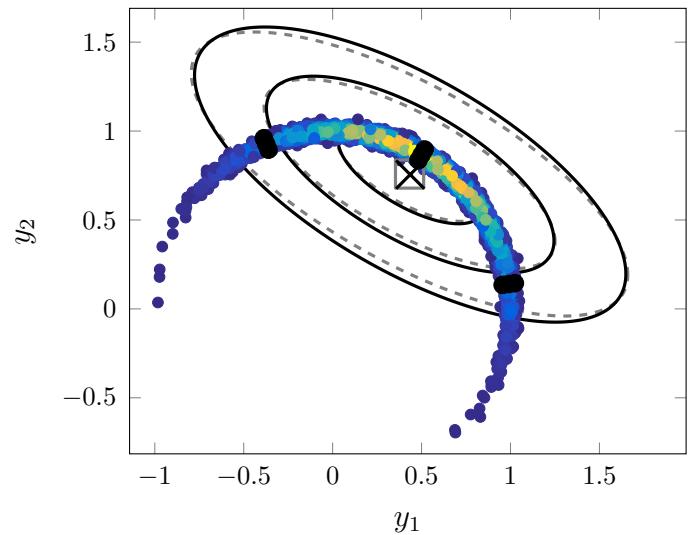
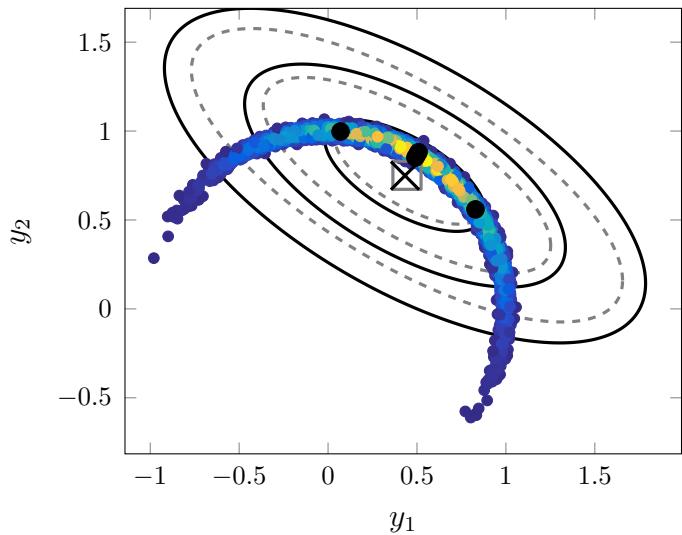
Note that the p^{th} order multi-dimensional Gauss-Hermite integration is exact for monomials of the form $x_1^{d_1} x_2^{d_2} \cdots x_n^{d_n}$, and their arbitrary linear combinations, where each of

the orders $d_i \leq 2p - 1$. The number of points required for an n -dimensional integral with p^{th} order rule is p^n , which quickly becomes infeasible as the number of dimensions grows.

Similar to the UT, we can use different SRFs in Gauss-Hermite quadrature, and, as expected, different SRFs lead to different locations of the quadrature points. Here, the difference between using a spectral factorization (left) and a Cholesky decomposition (right) is illustrated for the generation of a set of quadrature points for $p = 5$ and $n = 2$.



Let's take another look at our polar-to-Cartesian coordinates example, which we previously used to compare linearization and the UT to Monte Carlo integration. This time, we will compare the UT and Gauss-Hermite quadrature. The UT is configured identically to the previous time it was used, and Gauss-Hermite quadrature is employed with $p = 3$; for this two-dimensional case, this means that there is a total of 3^2 quadrature points.



Gauss-Hermite quadrature does a very good job of matching the Monte Carlo results here. Visually, it clearly outperforms the UT in the determination of the covariance.

We have the general form of the Gaussian filtering equations; these rely on approximating multidimensional Gaussian integrals. We also have an approximation for multidimensional Gaussian integrals using Gauss-Hermite quadrature. If we put these two things together, we get the Gauss-Hermite Kalman filter (GHKF), which is also called the quadrature Kalman filter (QKF).

We will consider the system to be described via the additive noise model, such that

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k$$

We are also assuming that the process noise and measurement noise are both zero-mean, white-noise sequences that are both uncorrelated with the state.

Given initial values of the mean and covariance, $\mathbf{m}_{x,0}$ and $\mathbf{P}_{xx,0}$, along with a sequence of data, \mathbf{z}_k , for $k = 1, 2, \dots$, we want to propagate the mean and covariance between

measurements and use the measurement information to update the mean and covariance when it is available.

We begin the GHKF by selecting a quadrature rule, which is done by selecting the order, p , of the one-dimensional quadrature rule and specifying the number of dimensions, n . This provides for us the unit quadrature points $\boldsymbol{\xi}^{(i_1, \dots, i_n)}$ and the associate quadrature weights $w^{(i_1, \dots, i_n)}$. These points and weights will be used in both the propagation and update stages, but they only need to be generated one time.

For the propagation stage, we want to propagate the mean $\mathbf{m}_{x,k-1}^+$ and covariance $\mathbf{P}_{xx,k-1}^+$ from t_{k-1} to t_k . We start by computing a SRF of $\mathbf{P}_{xx,k-1}^+$, which is given by $\mathbf{S}_{xx,k-1}^+$. We then transform the unit quadrature points according to

$$\boldsymbol{\chi}_{k-1}^{(i_1, \dots, i_n)} = \mathbf{m}_{x,k-1}^+ + \mathbf{S}_{xx,k-1}^+ \boldsymbol{\xi}^{(i_1, \dots, i_n)}, \quad i_1, \dots, i_n = 1, \dots, p$$

Each quadrature point is transformed the nonlinear dynamics as

$$\boldsymbol{\chi}_k^{(i_1, \dots, i_n)} = \mathbf{f}(\boldsymbol{\chi}_{k-1}^{(i_1, \dots, i_n)}), \quad i_1, \dots, i_n = 1, \dots, p$$

To complete the propagation stage, we use the transformed quadrature points and the computed quadrature weights to compute the *a priori* mean and covariance according to

$$\mathbf{m}_{x,k}^- = \sum_{i_1, \dots, i_n} w^{(i_1, \dots, i_n)} \boldsymbol{\chi}_k^{(i_1, \dots, i_n)}$$

$$\mathbf{P}_{xx,k}^- = \sum_{i_1, \dots, i_n} w^{(i_1, \dots, i_n)} (\boldsymbol{\chi}_k^{(i_1, \dots, i_n)} - \mathbf{m}_{x,k}^-)(\boldsymbol{\chi}_k^{(i_1, \dots, i_n)} - \mathbf{m}_{x,k}^-)^T + \mathbf{P}_{ww,k-1}$$

We have to remember to include the process noise covariance to account for the presence of noise in the dynamics. The quadrature approximation of the expectation integral does not account for this term.

For the update stage, we want to use new measurement data contained in \mathbf{z}_k to update our understanding of the state of the system and our confidence in that state estimate. The SRF of the prior covariance is computed and given by $\mathbf{S}_{xx,k}^-$, which allows us to form prior quadrature points as

$$\boldsymbol{\chi}_k^{(i_1, \dots, i_n)} = \mathbf{m}_{x,k}^- + \mathbf{S}_{xx,k}^- \boldsymbol{\xi}^{(i_1, \dots, i_n)}, \quad i_1, \dots, i_n = 1, \dots, p$$

where the unit points $\boldsymbol{\xi}^{(i_1, \dots, i_n)}$ are the same as those used in the propagation stage. Each of the prior quadrature points is transformed through the measurement model to produce

$$\mathbf{z}^{(i_1, \dots, i_n)} = \mathbf{h}(\mathbf{x}_k^{(i_1, \dots, i_n)}) , \quad i_1, \dots, i_n = 1, \dots, p$$

from which the predicted measurement, innovation covariance, and cross-covariance are

$$\mathbf{m}_{z,k}^- = \sum_{i_1, \dots, i_n} w^{(i_1, \dots, i_n)} \mathbf{z}_k^{(i_1, \dots, i_n)}$$

$$\mathbf{P}_{zz,k}^- = \sum_{i_1, \dots, i_n} w^{(i_1, \dots, i_n)} (\mathbf{z}_k^{(i_1, \dots, i_n)} - \mathbf{m}_{z,k}^-) (\mathbf{z}_k^{(i_1, \dots, i_n)} - \mathbf{m}_{z,k}^-)^T + \mathbf{P}_{vv,k}$$

$$\mathbf{P}_{xz,k}^- = \sum_{i_1, \dots, i_n} w^{(i_1, \dots, i_n)} (\mathbf{x}_k^{(i_1, \dots, i_n)} - \mathbf{m}_{x,k}^-) (\mathbf{z}_k^{(i_1, \dots, i_n)} - \mathbf{m}_{z,k}^-)^T$$

where it is important to remember to include the measurement noise covariance in the calculation of the innovation covariance. Finally, we can perform the update as

$$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k (\mathbf{z}_k - \mathbf{m}_{z,k}^-)$$

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^- \mathbf{K}_k^T - \mathbf{K}_k (\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k \mathbf{P}_{zz,k}^- \mathbf{K}_k^T$$

where the Kalman gain is given by

$$\mathbf{K}_k = \mathbf{P}_{xz,k}^{-} (\mathbf{P}_{zz,k}^{-})^{-1}$$

This completes the GHKF for the additive noise model.

We can also address non-additive noise models of the form

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{w}_{k-1})$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k)$$

To handle these types of systems, we define augmented states, where the input state to each nonlinear function is augmented with the corresponding noise. These augmented states have associated means and covariances that can be used to define quadrature points and weights. Since the dimensions of the noises do not have to be the same, this means that we may end up with two distinct sets of unit quadrature points that are used individually for the propagation and update stages of the GHKF. Otherwise, the approach is very similar to the one that we have previously developed for the UKF. Beware, however, as the rapid growth of computational requirements with the increased dimension of the augmented state over that of the original state often makes the procedure computationally infeasible.

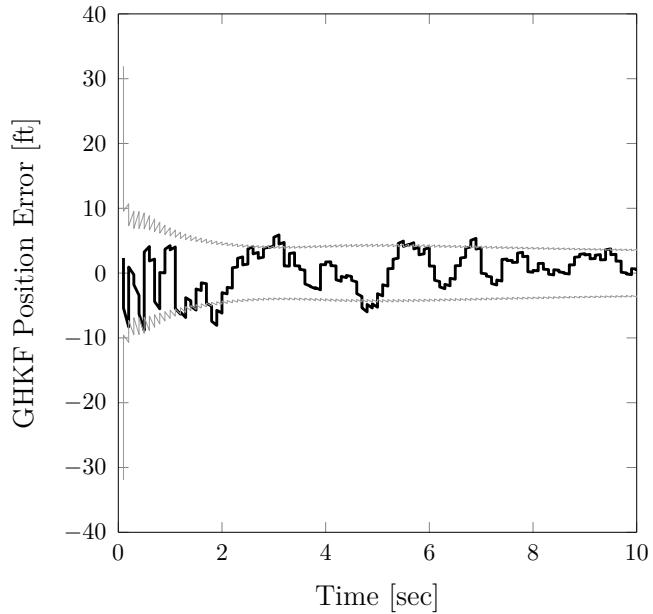
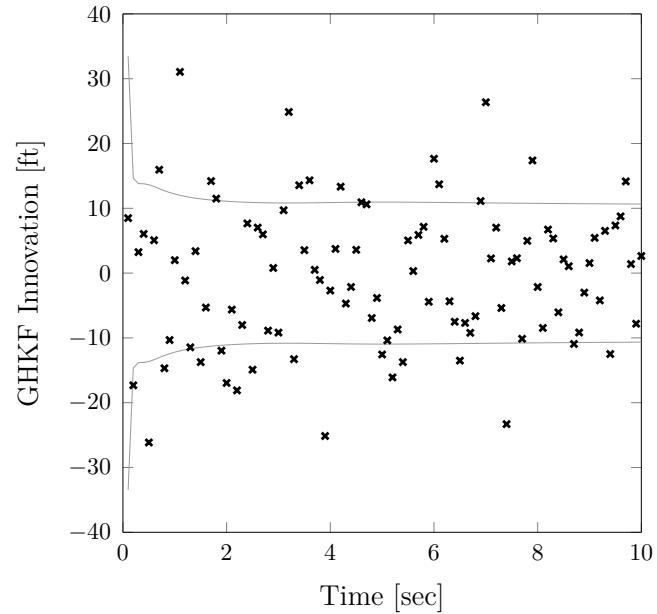
5.2.1 GHKF Example

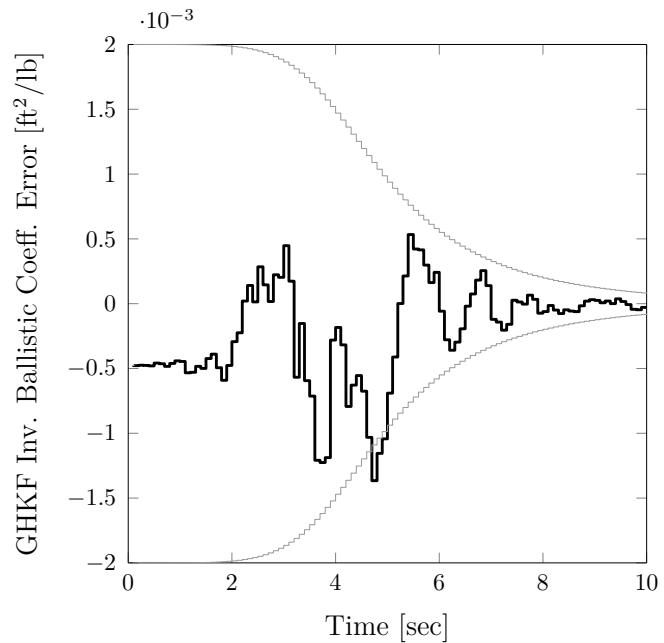
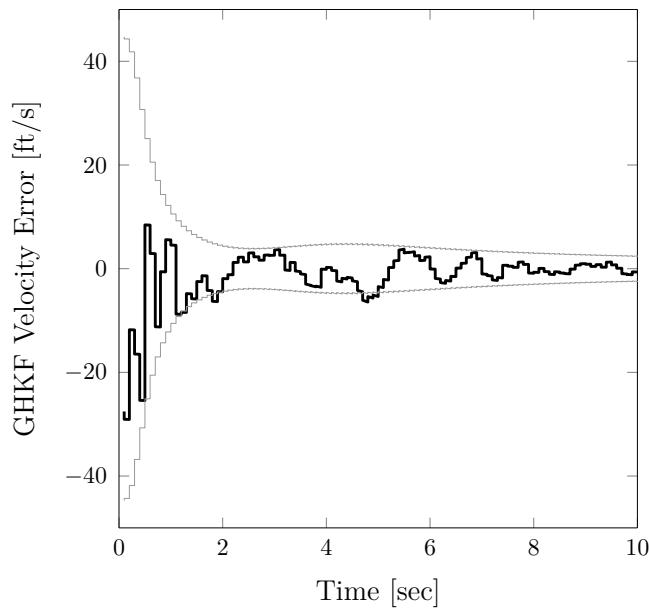
To provide another comparison point for the GHKF, we will take another look at the example from Gelb that we used to compare the UKF to the EKF.

We will use the exact same simulation setup, but this time we will use the GHKF. To implement the GHKF, we choose to use a third-order Gauss-Hermite quadrature scheme, and, since our state dimension is 3, we end up with a total of 27 quadrature points. The unit quadrature points are generated one time. Every application of the Gauss-Hermite quadrature rule for computing expectations is done by transforming the unit quadrature points. We use numerical integration in the propagation step as the nonlinear transformation for the dynamics, and we use the generalized Joseph form of the covariance update.

The following figures provide the innovations and state estimation errors, along with their corresponding 1σ intervals for the GHKF. Comparing against the previous set of results obtained with the EKF and the UKF, we note remarkably similar behavior. In fact, there is no visual difference between the results obtained with any of the three filters. Even though the GHKF can perform better than the EKF and the UKF, it does

not in all cases.





5.3 The Cubature Kalman Filter

The Gauss-Hermite approach to numerical integration is quite successful, but it relies on the extension of one-dimensional quadrature rules into multiple dimensions, and this leads to an exponential growth in quadrature points as the dimensionality grows. An alternative is *cubature*, which is an integration framework which naturally permits integration in multiple dimensions.

Recall from our previous discussion that the expected value of a nonlinear function with respect to an arbitrary Gaussian distribution, given by

$$\int \mathbf{g}(\mathbf{x}) p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) d\mathbf{x}$$

can always be transformed into an expectation integral over the unit Gaussian distribution

$$\int \mathbf{g}(\mathbf{m}_x + \mathbf{S}_{xx}\boldsymbol{\xi}) p_g(\boldsymbol{\xi}; \mathbf{0}_n, \mathbf{I}_n) d\boldsymbol{\xi}$$

where \mathbf{S}_{xx} is a square-root factor of \mathbf{P}_{xx} . To develop the cubature technique, let's assume that a change of variables is not required. Accordingly, we can start by considering the

multidimensional unit Gaussian integral

$$\int \mathbf{g}(\boldsymbol{\xi}) p_g(\boldsymbol{\xi}; \mathbf{0}_n, \mathbf{I}_n) d\boldsymbol{\xi}$$

We now want to form a $2n$ -point approximation of the form

$$\int \mathbf{g}(\boldsymbol{\xi}) p_g(\boldsymbol{\xi}; \mathbf{0}_n, \mathbf{I}_n) d\boldsymbol{\xi} \approx w \sum_{i=1}^{2n} \mathbf{g}(c\mathbf{u}^{(i)})$$

where w is a weight and c is a parameter that both need to be determined. Furthermore, the points $\mathbf{u}^{(i)}$ belong to the symmetric set $[\mathbf{1}]$ with generator $(1, 0, \dots, 0)$:

$$[\mathbf{1}] = \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} -1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \dots \right\}$$

Because the point set is symmetric, the rule is exact for all monomials of the form $x_1^{d_1} x_2^{d_2} \dots x_n^{d_n}$, if at least one of the exponents d_i is odd. Thus, we can construct a rule

that is exact up to third degree by determining the coefficients w and c such that the approximation becomes exact for selections $g_j(\boldsymbol{\xi}) = 1$ and $g_j(\boldsymbol{\xi}) = \xi_j^2$. Since we are working with the unit Gaussian pdf, we know that the true values of the expected values for these inputs of g_j are given by

$$\int p_g(\boldsymbol{\xi}; \mathbf{0}_n, \mathbf{I}_n) d\boldsymbol{\xi} = 1$$

$$\int \xi_j^2 p_g(\boldsymbol{\xi}; \mathbf{0}_n, \mathbf{I}_n) d\boldsymbol{\xi} = 1$$

Substituting these functions into the cubature rule and equating both sides (by using the preceding results, it follows that w and c may be chosen to satisfy the equalities

$$w \sum_{i=1}^{2n} 1 = w \cdot 2n = 1$$

$$w \sum_{i=1}^{2n} [cu_j^{(i)}]^2 = w \cdot 2c^2 = 1$$

We can solve the first equation for w , and then use this solution in the second equation to solve for c , which yields

$$w = \frac{1}{2n} \quad \text{and} \quad c = \sqrt{n}$$

This means that we have the following cubature rule, which is exact for monomials up to third degree:

$$\int p_g(\boldsymbol{\xi}; \mathbf{0}_n, \mathbf{I}_n) d\boldsymbol{\xi} \approx \frac{1}{2n} \sum_{i=1}^{2n} \mathbf{g}(\sqrt{n}\mathbf{u}^{(i)})$$

We can now easily extend the method to arbitrary mean and covariance by using our change of variables technique. The third-order spherical cubature approximation to the multidimensional expectation integral is given by

$$\int \mathbf{g}(\mathbf{x}) p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) d\mathbf{x} \approx \frac{1}{2n} \sum_{i=1}^{2n} \mathbf{g}(\mathbf{m}_x + \mathbf{S}_{xx}\boldsymbol{\xi}^{(i)})$$

The cubature points are given by

$$\boldsymbol{\xi}^{(i)} = \begin{cases} \sqrt{n} \mathbf{e}_i & i = 1, \dots, n \\ -\sqrt{n} \mathbf{e}_{i-n} & i = n+1, \dots, 2n \end{cases}$$

where \mathbf{e}_i denotes a unit vector in the direction of the coordinate axis i . An interesting thing to note is that this cubature method has the useful property that its weights are always positive, which is not always true for the UT or Gauss-Hermite approaches.

It is also easy to see that the above approximation is a special case of the unscented transform with parameters $\alpha = 1$, $\beta = 0$, and $\kappa = 0$. With this parameter selection, the mean weight is zero and the unscented transform is effectively a $2n$ -point approximation as well. Given this connection to the UT, we might wonder if it is possible to generalize the cubature approach to a $2n + 1$ point approximation. This can be done by including a “zeroth” cubature point at the mean of the input, which leads to an approximation of the form

$$\int \mathbf{g}(\boldsymbol{\xi}) p_g(\boldsymbol{\xi}; \mathbf{0}_n, \mathbf{I}_n) d\boldsymbol{\xi} \approx w_0 \mathbf{g}(\mathbf{0}_n) + w \sum_i \mathbf{g}(c \mathbf{u}^{(i)})$$

We can now solve for the parameters w_0 , w , and c such that we get the exact result with selections $g_j(\boldsymbol{\xi}) = 1$ and $g_j(\boldsymbol{\xi}) = \xi_j^2$. The solution is given by

$$w_0 = \frac{\kappa}{n + \kappa}, \quad w = \frac{1}{2(n + \kappa)}, \quad \text{and} \quad c = \sqrt{n + \kappa}$$

where κ is a free parameter. This yields a cubature integration rule for arbitrary Gaussian pdfs that is given by

$$\int \mathbf{g}(\mathbf{x}) p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) d\mathbf{x} \approx \frac{\kappa}{n + \kappa} \mathbf{g}(\mathbf{m}_x) + \frac{1}{2(n + \kappa)} \sum_{i=1}^{2n} \mathbf{g}(\mathbf{m}_x + \mathbf{S}_{xx} \boldsymbol{\xi}^{(i)})$$

where

$$\boldsymbol{\xi}^{(i)} = \begin{cases} \sqrt{n + \kappa} \mathbf{e}_i & i = 1, \dots, n \\ -\sqrt{n + \kappa} \mathbf{e}_{i-n} & i = n + 1, \dots, 2n \end{cases}$$

and \mathbf{S}_{xx} is an SRF of \mathbf{P}_{xx} , as before.

This integration rule corresponds to the unscented transform as previously discussed

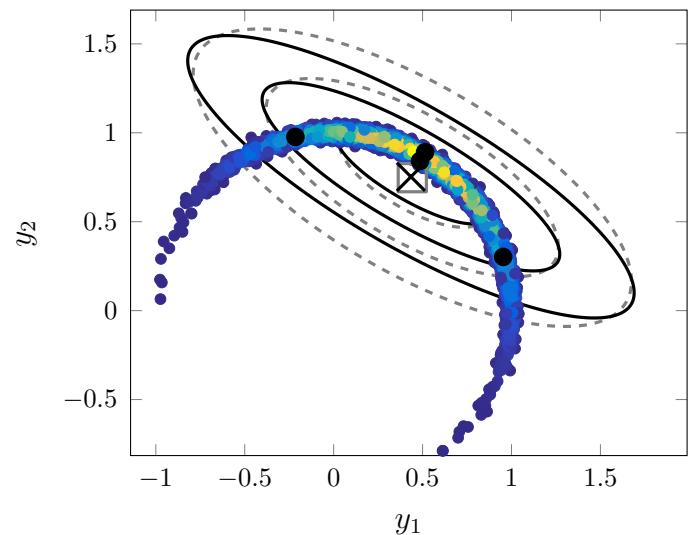
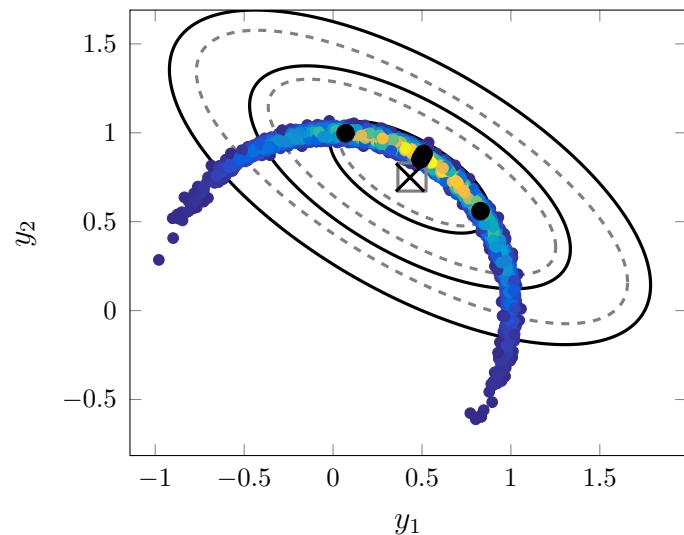
with $\alpha = 1$, $\beta = 0$, and where κ is left as a free parameter. With the selection of $\kappa = 3 - n$, we can also match the fourth order moments of the distribution, but with the price that when the dimension of the input is $n > 3$, we get negative weights. This is sometimes a problem in that the approximation rules can become unstable.

We developed the cubature rule to be exact “up to third degree.” Note that here, when we say this method is third-order accurate, this means a different thing than it does for Gauss-Hermite quadrature. The p^{th} order Gauss-Hermite quadrature rule is exact for monomials up to order $2p - 1$, which means that third-order Gauss-Hermite quadrature is exact for monomials up to fifth order. Third-order spherical cubature is exact only for monomials up to third order.

It is also possible to derive symmetric rules that are exact for higher than third order. However, this is no longer possible for a number of cubature points which is linear, i.e. $\mathcal{O}(n)$, in the state dimension. For example, for a fifth-order rule, the required number of points is proportional to n^2 .

Let’s take one last look at our polar-to-Cartesian coordinates example, which we have used to compare linearization, the UT, and Gauss-Hermite quadrature to one another

and to Monte Carlo integration. Since the cubature rule shares some similarities with the UT, we will once again look at the results in comparison to the UT. The UT is configured identically to the previous times it was used, and the cubature method is employed using the $2n$ rule, such that there are no tuning parameters selected.



When compared to the UT, the $2n$ cubature method performs quite well. In light of the fact that no tuning parameters are required, this level of performance makes cubature

an attractive option.

Just as we did with the GHKF, we can apply the third order spherical cubature integration rule to the expectations in the previously developed Gaussian filter equations. We call the resulting combination of the Gaussian integral with the approximate expectations the cubature Kalman filter (CKF).

We will consider the system to be described via the additive noise model, such that

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k$$

where the process noise and measurement noise are both zero-mean, white-noise sequences that are both uncorrelated with the state.

Given initial values of the mean and covariance, $\mathbf{m}_{x,0}$ and $\mathbf{P}_{xx,0}$, along with a sequence of data, \mathbf{z}_k , for $k = 1, 2, \dots$, we want to propagate the mean and covariance between measurements and use the measurement information to update the mean and covariance when it is available. We need to select a set of cubature points; here, we use the set

of $2n$ cubature points, but the equivalent form using $2n+1$ is a straightforward extension.

For the propagation stage, we want to propagate the mean $\mathbf{m}_{x,k-1}^+$ and covariance $\mathbf{P}_{xx,k-1}^+$ from t_{k-1} to t_k . The SRF of $\mathbf{P}_{xx,k-1}^+$, which is given by $\mathbf{S}_{xx,k-1}^+$, is used to generate the quadrature points at t_{k-1} as

$$\boldsymbol{\chi}_{k-1}^{(i)} = \mathbf{m}_{x,k-1}^+ + \mathbf{S}_{xx,k-1}^+ \boldsymbol{\xi}^{(i)}$$

where the set of $2n$ unit cubature points is defined as

$$\boldsymbol{\xi}^{(i)} = \begin{cases} \sqrt{n} \mathbf{e}_i & i = 1, \dots, n \\ -\sqrt{n} \mathbf{e}_{i-n} & i = n+1, \dots, 2n \end{cases}$$

Then, each of the cubature points is transformed through the nonlinear dynamics to yield propagated cubature points as

$$\boldsymbol{\chi}_k^{(i)} = \mathbf{f}(\boldsymbol{\chi}_{k-1}^{(i)})$$

The propagation stage is completed by computing the *a priori* mean and covariance as

$$\mathbf{m}_{x,k}^- = \frac{1}{2n} \sum_{i=1}^{2n} \boldsymbol{\chi}_k^{(i)}$$

$$\mathbf{P}_{xx,k}^- = \frac{1}{2n} \sum_{i=1}^{2n} (\boldsymbol{\chi}_k^{(i)} - \mathbf{m}_{x,k}^-)(\boldsymbol{\chi}_k^{(i)} - \mathbf{m}_{x,k}^-)^T + \mathbf{P}_{ww,k-1}$$

where it is important to remember that the effects of process noise need to be included in the propagated covariance matrix.

When confronted with new measurement data, \mathbf{z}_k , we use that information to update our state estimate and covariance. The update begins by determining the SRF of the *a priori* covariance matrix, $\mathbf{S}_{xx,k}^-$. The *a priori* mean and SRF are used to generate cubature points as

$$\boldsymbol{\chi}_k^{(i)} = \mathbf{m}_{x,k}^- + \mathbf{S}_{xx,k}^- \boldsymbol{\xi}^{(i)}$$

where the unit cubature points are

$$\boldsymbol{\xi}^{(i)} = \begin{cases} \sqrt{n} \mathbf{e}_i & i = 1, \dots, n \\ -\sqrt{n} \mathbf{e}_{i-n} & i = n+1, \dots, 2n \end{cases}$$

Each cubature point is transformed through the nonlinear measurement function to yield

$$\mathcal{Z}^{(i)} = \mathbf{h}(\mathcal{X}_k^{(i)})$$

and these cubature points are used to compute the predicted measurement, innovation covariance, and cross-covariance as

$$\mathbf{m}_{z,k}^- = \frac{1}{2n} \sum_{i=1}^{2n} \mathcal{Z}_k^{(i)}$$

$$\mathbf{P}_{zz,k}^- = \frac{1}{2n} \sum_{i=1}^{2n} (\mathcal{Z}_k^{(i)} - \mathbf{m}_{z,k}^-)(\mathcal{Z}_k^{(i)} - \mathbf{m}_{z,k}^-)^T + \mathbf{P}_{vv,k}$$

$$\mathbf{P}_{xz,k}^- = \frac{1}{2n} \sum_{i=1}^{2n} (\mathcal{X}_k^{(i)} - \mathbf{m}_{x,k}^-)(\mathcal{Z}_k^{(i)} - \mathbf{m}_{z,k}^-)^T$$

where it is important to remember to include the effects of the measurement noise in the innovation covariance. Finally, we can perform the update as

$$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{m}_{z,k}^-)$$

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{P}_{xz,k}^-\mathbf{K}_k^T - \mathbf{K}_k(\mathbf{P}_{xz,k}^-)^T + \mathbf{K}_k\mathbf{P}_{zz,k}^-\mathbf{K}_k^T$$

where the Kalman gain is given by

$$\mathbf{K}_k = \mathbf{P}_{xz,k}^-(\mathbf{P}_{zz,k}^-)^{-1}$$

This completes the CKF for the additive noise model.

As with the UKF and the GHKF, we can also address non-additive noise models of the form

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{w}_{k-1})$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k)$$

To handle these types of systems, we define augmented states, where the input state to each nonlinear function is augmented with the corresponding noise. These augmented

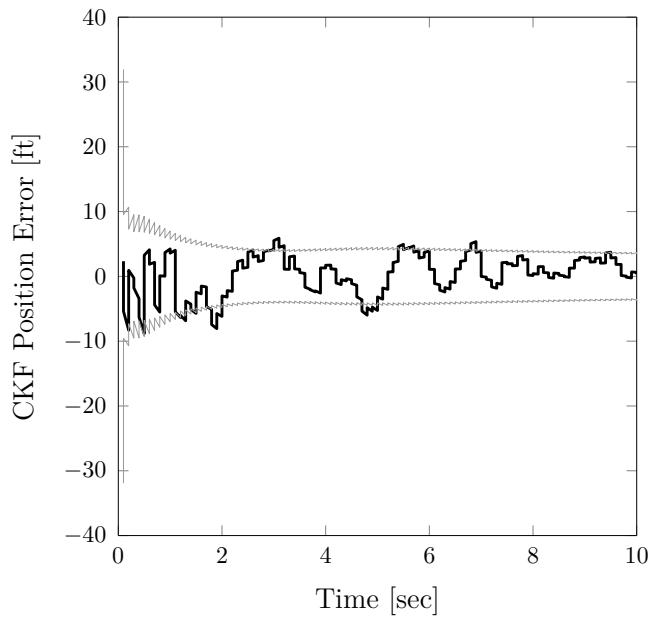
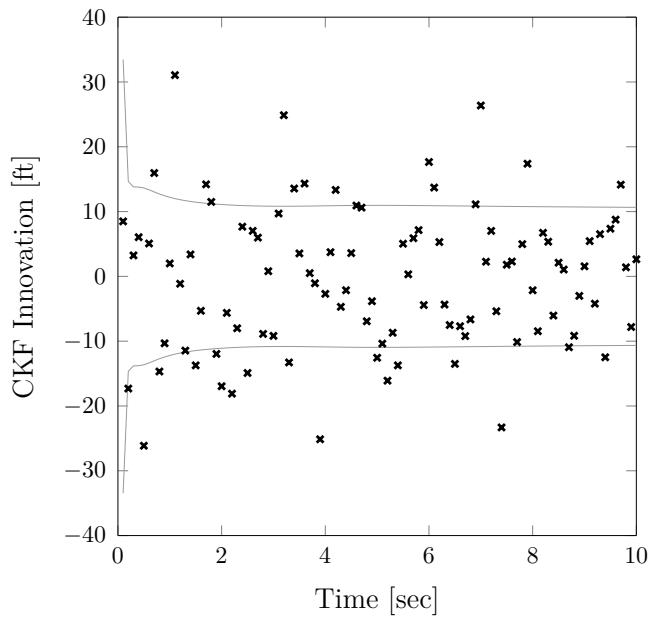
states have associated means and covariances that can be used to define cubature points and weights. The augmentation procedure means that more cubature points are generated for the propagation and for the update. Unlike the GHKF, however, the number of cubature points does not grow exponentially with the dimension of the (augmented) input. Otherwise, the approach follows the same process to the method that we have previously developed for the UKF.

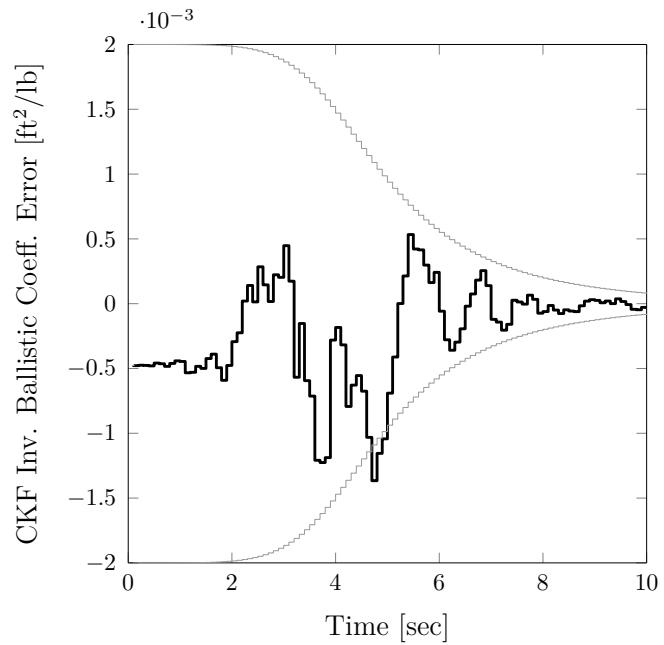
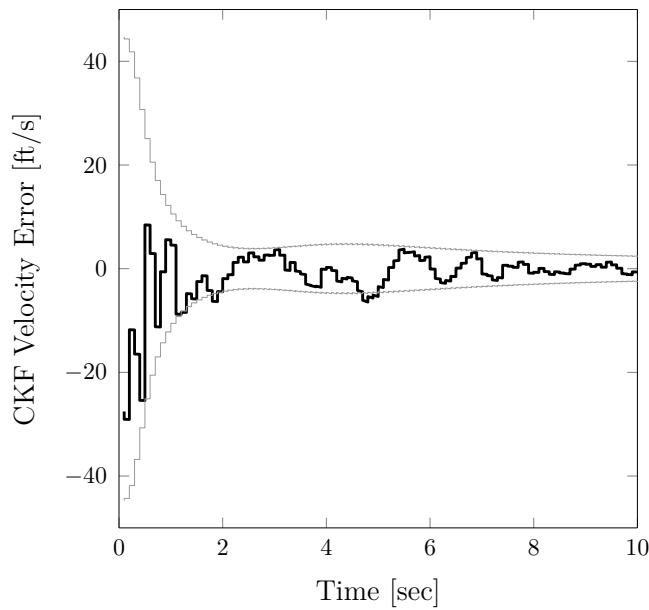
5.3.1 CKF Example

As a final comparison point for our derivative-free Kalman filters, we take on last look at the example from Gelb that we have used to compare the EKF, UKF, and GHKF.

The same simulation setup is used again, but this time we apply the CKF to estimate the position, velocity, and inverse ballistic coefficient of the falling body using range measurements. To implement the CKF, we make use of the $2n$ cubature rule, which contains no tuning parameters. As with our other implementations, we use numerical integration in the propagation step as the nonlinear transformation for the dynamics, and we use the generalized Joseph form of the covariance update.

The following figures provide the innovations and state estimation errors, along with their corresponding 1σ intervals for the CKF. Comparing against the previous set of results obtained with the EKF, the UKF, and the GHKF, we note that similar estimation performance is obtained via the CKF. As before, there is very little, to no, difference in the performance. One thing that we should note is that the exact same sequence of measurement data is used for all of these implementations. Different measurement data would lead to a different sequence of innovations and estimation errors.





5.4 Comments on Derivative-Free Filtering

When should we use the UKF, GHKF, or CKF?

The UKF is very popular with people who are looking for a fairly lightweight, derivative-free alternative to the EKF. However, it is more computationally expensive and it requires the calculation of Cholesky factors. While Cholesky decomposition is not an extremely expensive procedure, it does add computational complexity. Furthermore, if your estimation error covariance matrix is to *ever* lose its positive-definiteness, Cholesky decomposition will fail, and your UKF will fail with it. It is possible to circumvent this by combining the square-root filtering ideology with the UKF to formulate a square-root UKF.

The GHKF can offer more accurate estimates of the first two moments of the target distributions for certain problems (such as the polar-to-Cartesian transformation we saw before). This comes at the cost of more points to evaluate our functions at and more computational burden. This is especially true with higher-dimensional problems, due to the exponential growth of quadrature points. Sparse grid methods have been developed that prune away some of the less important quadrature points, and these methods can

offer much of the performance of the GHKF at a somewhat reduced computational cost.

The CKF is an attractive option that is fairly lightweight, like the UKF. An advantage of the CKF over the UKF, however, is that it requires no tuning parameters (for the $2n$ formulation) and thus not sensitive to the selection of these parameters. It shares a disadvantage with the UKF and GHKF, in that it also is vulnerable to failure by loss of positive-definiteness.

The most important thing to remember, however, is that not one of these methods is inherently better than the other. The choice of filter is very problem dependent, with things like the choice of tuning parameters and computational complexity to consider.

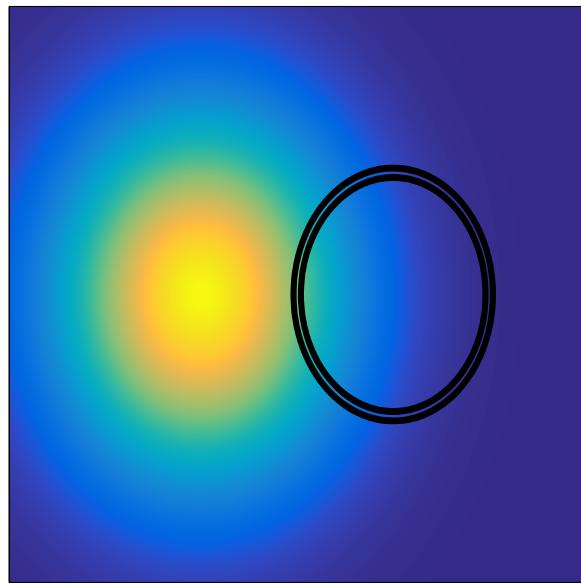
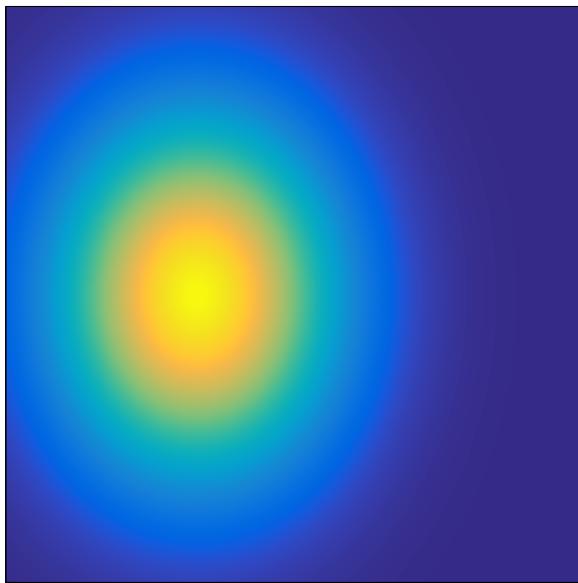
6

Bayesian Filtering

Up to this point, we have focused on least squares and Kalman filter methods. These approaches share many things in common, including their utilization of mean and covariance. When discussing derivative-free Kalman filtering and moment matching methods, we made use of assumed density representations, where we repeatedly assume that the *a priori* and *a posteriori* densities are Gaussian to generate quadrature or cubature points. In other methods, while we worked with moments of a distribution, we didn't explicitly rely on a probability density function (pdf) representation.

With Bayesian filtering methods, we move away from descriptions using only mean and covariance or descriptions that rely on Gaussianity. While we can accommodate Gaussian pdfs, we will also be able to make use of more information about the pdf with the

Bayesian approach. To see what we mean without getting into too much depth, let's consider a motivating example. For this example, we are interested in estimating the position of a vehicle approach the origin, and we are at the origin able to acquire range measurements as the vehicle approaches from the left. The *a priori* uncertainty is taken to be Gaussian and is illustrated below (to the left).



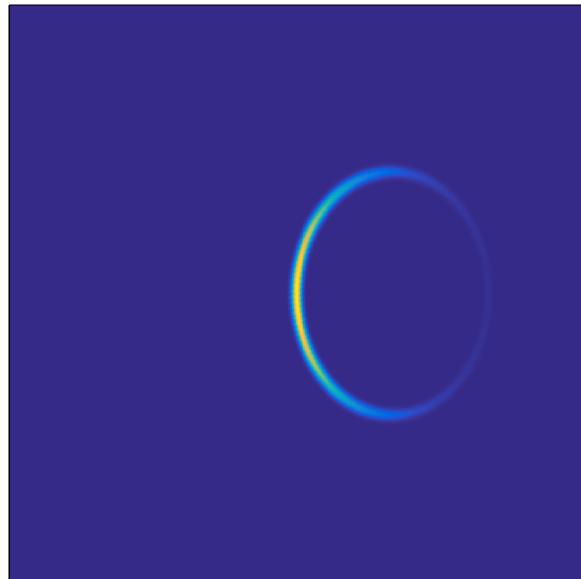
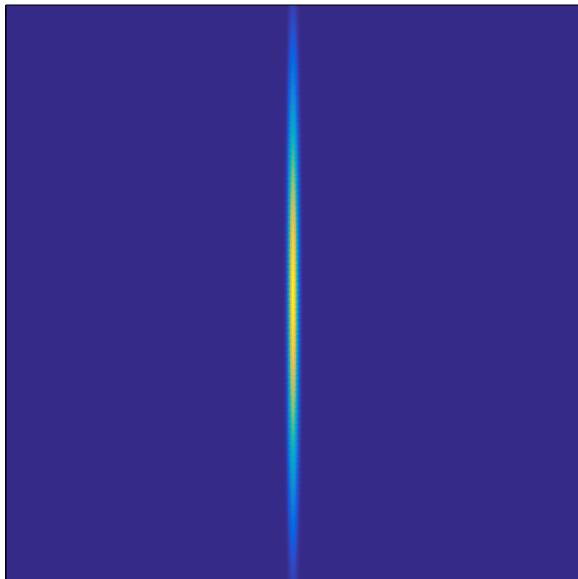
At this time, we take a very precise measurement of the range of the vehicle from the origin, and this is illustrated as the ring above (to the right).

What happens if we take use the EKF to update our state estimate and covariance? This is illustrated below (to the left), where we have plotted a Gaussian pdf using the posterior mean and covariance from the EKF. This posterior distribution should recall memories of the polar-to-Cartesian example that we investigated previously. The EKF is linearizing, and a tangent to the range circle is the outcome.

However, we probably have the intuitive feeling that if we aren't confined to mean and covariance that we should end up with a different posterior uncertainty. In fact, we might already expect that the posterior pdf should be curved, such that there is posterior uncertainty where there is both prior evidence and data to support the prior evidence. This is what is illustrated in the following figure (on the right).

This curved pdf cannot be found by using an EKF (or any other variant of the KF). All we get from least squares and Kalman filter methods is mean and covariance. The Kalman filter can be modified to work with moments beyond mean and covariance, but those are specialized forms of the Kalman filtering paradigm. As we have presented the

Kalman filter, all we get is mean and covariance. Sometimes, that is more than enough to do the job, but sometimes we want a *complete* description of the uncertainty. This is what we will attempt to achieve by considering Bayesian methods.



6.1 Probabilistic State Space Models

A probabilistic state space model replaces our usual system model that is comprised of dynamics and measurements. We will focus here on discrete-time dynamics and measurements, and the probabilistic state space model consists of a sequence of conditional pdfs for the true states of the system and the measurements of the system take the form

$$\mathbf{x}_k \sim p(\mathbf{x}_k | \mathbf{x}_{k-1})$$

$$\mathbf{z}_k \sim p(\mathbf{z}_k | \mathbf{x}_k)$$

for $k = 1, 2, \dots$, where

- \mathbf{x}_k (typically $\in \mathbb{R}^n$) is the state vector of interest at time step k .
- \mathbf{z}_k (typically $\in \mathbb{R}^m$) is the vector containing the measurement at time step k .
- $p(\mathbf{x}_k | \mathbf{x}_{k-1})$ describes the time evolution of the stochastic process. Note that this can be a probability density, a counting measure, or both depending on the nature of \mathbf{x}_k (continuous, discrete, or hybrid). For the problems that we will discuss, where we have continuous-valued states, this will be a pdf.

- $p(\mathbf{z}_k | \mathbf{x}_k)$ is the probabilistic measurement model that describes our understanding of the measurements given our current state.

The notation $p(\mathbf{a} | \mathbf{b})$ denotes the pdf of \mathbf{a} conditioned on \mathbf{b} ; that is, $p(\mathbf{a} | \mathbf{b})$ is the pdf of \mathbf{a} given that \mathbf{b} has “occurred.” As an example, $p(\mathbf{z}_k | \mathbf{x}_k)$ denotes the pdf of \mathbf{z}_k given knowledge of the state \mathbf{x}_k .

It is assumed that the model (i.e., both the state and measurement processes) is Markovian, which means two things:

1. The sequence of states $\{\mathbf{x}_k : k = 0, 1, 2, \dots\}$ form a Markov sequence (or Markov chain if the states are discrete). This means that any \mathbf{x}_k at time k (and later times as well) given the previous state \mathbf{x}_{k-1} at $k - 1$ are independent of any state prior to $k - 1$, such that

$$p(\mathbf{x}_k | \mathbf{x}_{0:k-1}, \mathbf{z}_{1:k-1}) = p(\mathbf{x}_k | \mathbf{x}_{k-1})$$

Additionally, the past is independent of the future (say at time ℓ) given the present, or

$$p(\mathbf{x}_{k-1} | \mathbf{x}_{k:\ell}, \mathbf{z}_{k:\ell}) = p(\mathbf{x}_{k-1} | \mathbf{x}_k)$$

2. The current measurement \mathbf{z}_k given the current state \mathbf{x}_k is conditionally independent of the measurement and state histories, or

$$p(\mathbf{z}_k \mid \mathbf{x}_{1:k}, \mathbf{z}_{1:k-1}) = p(\mathbf{z}_k \mid \mathbf{x}_k)$$

With this assumption of a Markovian model, the joint prior distribution of the states is

$$p(\mathbf{x}_{0:\ell}) = p(\mathbf{x}_0) \prod_{k=1}^{\ell} p(\mathbf{x}_k \mid \mathbf{x}_{k-1})$$

Additionally, the joint likelihood of the measurements is

$$p(\mathbf{z}_{1:\ell} \mid \mathbf{x}_{0:\ell}) = \prod_{k=1}^{\ell} p(\mathbf{z}_k \mid \mathbf{x}_k)$$

Generating the posterior distribution given the prior and measurement likelihood is an inverse probability problem. That is, we must reverse the conditioning on our likelihood to improve our understanding of our state. This is computed using Bayes' rule via

$$p(\mathbf{x}_{0:\ell} \mid \mathbf{z}_{1:\ell}) = \frac{p(\mathbf{z}_{1:\ell} \mid \mathbf{x}_{0:\ell})p(\mathbf{x}_{0:\ell})}{p(\mathbf{z}_{1:\ell})}$$

$$\propto p(\mathbf{z}_{1:\ell} \mid \mathbf{x}_{0:\ell}) p(\mathbf{x}_{0:\ell})$$

This is *the* solution to the inverse probability problem. In principle, it is the ideal tool for incorporating new information into a prior understanding of a stochastic process. In spirit, this is very similar to the batch estimation problem, wherein a sequence of measurement data is used to produce estimates of the states of the system.

In practice, however, Bayes' rule often becomes infeasible to compute in this form, particularly for real-time applications, because the number of computations per time step increases as new observations arrive. This is similar to the problem that motivated our development of sequential batch estimators.

To overcome the infeasibility present in Bayes' rule, we will develop methods to compute the posterior distribution *recursively*, such that a constant number of operations are required per time step. To do this, we will not consider the computation of the full posterior at all, but instead we will focus on what will be called *filtering* and *prediction* distributions. You can think of these as the Bayesian versions of the update and propagate steps with which we are familiar from our work on Kalman filtering.

6.2 Bayesian Filtering Equations

The objective of recursive Bayesian filtering is to compute the posterior distribution or *filtering distribution* of the state \mathbf{x}_k at each step k using (or given) the entire measurement history up to step k , i.e.,

$$p(\mathbf{x}_k | \mathbf{z}_{1:k})$$

The *predicted distribution*, on the other hand, describes our understanding of \mathbf{x}_k given all the measurements up to $k - 1$, and it is written as

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1})$$

Recursive Bayesian filtering can be outlined as follows:

- **Initialization:** The recursion starts from the prior distribution $p(\mathbf{x}_0)$
- **Prediction:** The predicted distribution of state \mathbf{x}_k at step k , given the dynamic model of the state, can be computed by the Chapman-Kolmogorov equation

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k | \mathbf{x}_{k-1}) p(\mathbf{x}_{k-1} | \mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1}$$

which describes the evolution of pdf through time and is fundamentally true under the Markovian assumptions we made earlier.

- **Correction:** Given a measurement \mathbf{z}_k at step k , the posterior distribution of the state \mathbf{x}_k is computed via Bayes' rule as

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) = \frac{1}{c_k} p(\mathbf{z}_k | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{z}_{1:k-1})$$

where c_k is a normalization constant given by

$$c_k = \int p(\mathbf{z}_k | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) d\mathbf{x}_k$$

which ensures that the filtering distribution is a valid pdf.

How do we show this?

Consider the joint distribution of \mathbf{x}_k and \mathbf{x}_{k-1} given the set of previous measurements $\mathbf{z}_{1:k-1}$, which can be computed as

$$\begin{aligned} p(\mathbf{x}_k, \mathbf{x}_{k-1} | \mathbf{z}_{1:k-1}) &= p(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{z}_{1:k-1}) p(\mathbf{x}_{k-1} | \mathbf{z}_{1:k-1}) \\ &= p(\mathbf{x}_k | \mathbf{x}_{k-1}) p(\mathbf{x}_{k-1} | \mathbf{z}_{1:k-1}) \end{aligned}$$

Note that the disappearance of the measurement history $\mathbf{z}_{1:k-1}$ is due to our previous assumption that the sequence $\{\mathbf{x}_k : k = 1, 2, \dots\}$ is a Markov process. The Chapman-Kolmogorov equation is then obtained by “integrating out” the dependence on \mathbf{x}_{k-1} to obtain the marginal distribution

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k | \mathbf{x}_{k-1}) p(\mathbf{x}_{k-1} | \mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1}$$

The marginal distribution of \mathbf{x}_k given \mathbf{z}_k and $\mathbf{z}_{1:k-1}$ (or equivalently, given $\mathbf{z}_{1:k}$) can be computed by Bayes’ rule

$$\begin{aligned} p(\mathbf{x}_k | \mathbf{z}_{1:k}) &= \frac{1}{c_k} p(\mathbf{z}_k | \mathbf{x}_k, \mathbf{z}_{1:k-1}) p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) \\ &= \frac{1}{c_k} p(\mathbf{z}_k | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) \end{aligned}$$

Note too that the disappearance of the measurement history $\mathbf{z}_{1:k-1}$ is due to the assumed conditional independence of \mathbf{z}_k with the measurement history given \mathbf{x}_k .

By making use of the Markovian property, we have stripped the whole state histories

from our computations in favor of only considering the state at time k . The combination of the Chapman-Kolmogorov equation and Bayes' rule is what we will collectively refer to as *Bayesian estimation* or the *Bayesian framework*. Just to summarize, we are now interested in propagating the probability density function (pdf) using the Chapman-Kolmogorov equation

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k | \mathbf{x}_{k-1}) p(\mathbf{x}_{k-1} | \mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1}$$

between measurements and updating the pdf using Bayes' Rule as

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{z}_{1:k-1})}{\int p(\mathbf{z}_k | \boldsymbol{\xi}_k) p(\boldsymbol{\xi}_k | \mathbf{z}_{1:k-1}) d\boldsymbol{\xi}_k}$$

when new measurement data are received. The Bayesian framework provides a starting point for developing practical methods for the estimation of dynamic systems.

6.3 The Bayesian Kalman Filter

What happens if we assume that

1. the initial pdf is Gaussian,
2. the process noise is Gaussian,
3. the measurement noise is Gaussian,
4. the dynamical system is linear, and
5. the observational system is linear?

As it turns out, we get an algorithmic equivalent to the Kalman filter. Note that we are assuming Gaussianity now, but we did not make use of this property in our original derivation of the Kalman filter. Just as a reminder, $p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx})$ represents a Gaussian distribution in \mathbf{x} , with mean \mathbf{m}_x and covariance \mathbf{P}_{xx} , which is defined by

$$p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) = |2\pi\mathbf{P}|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2}(\mathbf{x} - \mathbf{m}_x)^T \mathbf{P}_{xx}^{-1} (\mathbf{x} - \mathbf{m}_x) \right\}$$

We want to show that making the aforementioned assumptions, the Bayesian framework leads us to the Kalman filter. To begin, we need to take a look at the dynamical and

observational models in the case that they are linear and that the noises are Gaussian. For the dynamics, we have

$$\boldsymbol{x}_k = \boldsymbol{F}_{x,k-1} \boldsymbol{x}_{k-1} + \boldsymbol{w}_{k-1}$$

We can solve for the noise term as

$$\boldsymbol{w}_{k-1} = \boldsymbol{x}_k - \boldsymbol{F}_{x,k-1} \boldsymbol{x}_{k-1}$$

If the process noise is assumed to be zero-mean, Gaussian, then it follows that

$$\boldsymbol{w}_{k-1} \sim p_g(\boldsymbol{w}_{k-1}; \boldsymbol{0}_w, \boldsymbol{P}_{ww,k-1})$$

or

$$\boldsymbol{x}_k - \boldsymbol{F}_{x,k-1} \boldsymbol{x}_{k-1} \sim p_g(\cdot; \boldsymbol{0}_w, \boldsymbol{P}_{k-1})$$

We can also “add” $\boldsymbol{F}_{x,k-1} \boldsymbol{x}_{k-1}$ to both sides, which simply changes the mean on the right-hand side

$$\boldsymbol{x}_k \sim p_g(\boldsymbol{x}_k; \boldsymbol{F}_{x,k-1} \boldsymbol{x}_{k-1}, \boldsymbol{P}_{ww,k-1})$$

Therefore, in the language of conditional probabilities, the linear Gaussian dynamical system assumption yields

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}) = p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1} \mathbf{x}_{k-1}, \mathbf{P}_{ww,k-1})$$

We also need to consider the observational system. The linear assumption leads to the measurement model

$$\mathbf{z}_k = \mathbf{H}_{x,k} \mathbf{x}_k + \mathbf{v}_k$$

By noting that this is in the same form as the dynamical system, we can observe that following the same process leads us to the result that

$$p(\mathbf{z}_k | \mathbf{x}_k) = p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{x}_k, \mathbf{P}_{vv,k})$$

At this point, we have made use of our assumptions that the process noise and measurement noise are Gaussian and that our dynamics and measurements are linear. This has produced the likelihood models

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}) = p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1} \mathbf{x}_{k-1}, \mathbf{P}_{ww,k-1})$$

$$p(\mathbf{z}_k | \mathbf{x}_k) = p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{x}_k, \mathbf{P}_{vv,k})$$

We now need to combine these with our predictor/corrector relationships

$$p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1}) = \int p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}) p(\boldsymbol{x}_{k-1} \mid \boldsymbol{z}_{1:k-1}) d\boldsymbol{x}_{k-1}$$

$$p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k}) = \frac{p(\boldsymbol{z}_k \mid \boldsymbol{x}_k) p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1})}{\int p(\boldsymbol{z}_k \mid \boldsymbol{\xi}_k) p(\boldsymbol{\xi}_k \mid \boldsymbol{z}_{1:k-1}) d\boldsymbol{\xi}_k}$$

and our assumption that the initial pdf is Gaussian to formulate the desired estimator. Before we make that next step, though, let's make note that if we assume that the filtering distribution at $k - 1$ is Gaussian and show that the predicted and filtering distributions at k are both Gaussian, then we have a closed recursion for the pdf. This will fold in our assumption that the prior is Gaussian without making it an explicit step in the process.

It helps to take a step back and look at a result from 1964 that we will call Ho's equation, which is given by¹

$$p_g(\boldsymbol{x}; \boldsymbol{m}_x, \boldsymbol{P}_{xx}) p_g(\boldsymbol{z}; \boldsymbol{H}_x \boldsymbol{x}, \boldsymbol{P}_{vv}) = p_g(\boldsymbol{z}; \boldsymbol{H}_x \boldsymbol{m}_x, \boldsymbol{H}_x \boldsymbol{P}_{xx} \boldsymbol{H}_x^T + \boldsymbol{P}_{vv}) p_g(\boldsymbol{x}; \boldsymbol{\mu}_x, \boldsymbol{\Pi}_{xx})$$

¹Y.-C. Ho and R. C. K. Lee, "A Bayesian Approach to Problems in Stochastic Estimation and Control," *IEEE Transactions on Automatic Control*, Vol. 9, No. 4, 1964, pp. 333–339.

where

$$\boldsymbol{\mu}_x = \mathbf{m}_x + \mathbf{K}(\mathbf{z} - \mathbf{H}_x \mathbf{m}_x)$$

$$\boldsymbol{\Pi}_{xx} = \mathbf{P}_{xx} - \mathbf{K}\mathbf{H}_x\mathbf{P}_{xx}$$

$$\mathbf{K} = \mathbf{P}_{xx}\mathbf{H}_x^T(\mathbf{H}_x\mathbf{P}_{xx}\mathbf{H}_x^T + \mathbf{P}_{vv})^{-1}$$

This relationship transforms the product of a Gaussian distribution in \mathbf{x} and a Gaussian distribution in \mathbf{z} linearly conditioned on \mathbf{x} to the product of a Gaussian distribution in \mathbf{z} and a Gaussian distribution in \mathbf{x} linearly conditioned on \mathbf{z} . This result also contains some very familiar-looking terms that we have seen in the context of least-squares estimation and Kalman filtering.

In order to prove Ho's equation, its left-hand side will be manipulated to yield its right-hand side. The matrix inversion lemma (MIL)² and Sylvester's determinant theorem³ are required in the process of the proof. This proof is omitted from Ho and Lee's orig-

²M. A. Woodbury, "Inverting modified matrices," *Memorandum report*, Vol. 42, 1950, p. 106.

³J. J. Sylvester, "On the relation between the minor determinants of linearly equivalent quadratic functions," *Philosophical Magazine*, Vol. 1, 1851, pp. 295–305.

inal paper, and it is not very easily found in literature. Several references^{4,5,6} use Ho’s equation in deriving the Bayesian Kalman or other Gaussian-based filters; however, these references omit the proof and cite another source for the equation, which can ultimately be tracked back to Ho and Lee’s original paper.

In order to begin the proof, the definition of the Gaussian distribution is substituted into the left-hand side of Ho’s equation for both $p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx})$ and $p_g(\mathbf{z}; \mathbf{H}_x \mathbf{x}, \mathbf{P}_{vv})$. Then, the exponential terms are combined such that the product of the Gaussians is

$$p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx})p_g(\mathbf{z}; \mathbf{H}_x \mathbf{x}, \mathbf{P}_{vv}) = |2\pi \mathbf{P}_{xx}|^{-\frac{1}{2}} |2\pi \mathbf{P}_{vv}|^{-\frac{1}{2}} \\ \times \exp \left\{ -\frac{1}{2} \left[(\mathbf{x} - \mathbf{m}_x)^T \mathbf{P}_{xx}^{-1} (\mathbf{x} - \mathbf{m}_x) + (\mathbf{z} - \mathbf{H}_x \mathbf{x})^T \mathbf{P}_{vv}^{-1} (\mathbf{z} - \mathbf{H}_x \mathbf{x}) \right] \right\}$$

One thing to watch out for is that \mathbf{x} and \mathbf{z} are not, in general, the same dimension. We now have to go through a few miles of algebra, so we’ll provide the general idea of how the proof proceeds. These steps make most sense as you go through the proof yourself,

⁴B. Ristic, S. Arulampalam, and N. Gordon, **Beyond the Kalman Filter**. Artech House, 2004.

⁵B. D. Anderson and J. B. Moore, **Optimal Filtering**. Dover Publications, 1979.

⁶B. T. Vo, *Random Finite Sets in Multi-Object Filtering*. PhD thesis, The University of Western Australia, October 2008.

so a loose interpretation as they are given is advised.

1. Terms in the $[\dots]$ are expanded and grouped to make a quadratic equation in \mathbf{x}
2. The definition of $\boldsymbol{\Pi}_{xx}$ is introduced via $\boldsymbol{\Pi}_{xx}^{-1} = \mathbf{H}_x^T \mathbf{P}_{vv}^{-1} \mathbf{H}_x + \mathbf{P}_{xx}^{-1}$
3. The MIL is applied to get $\boldsymbol{\Pi}_{xx}$, which introduces the definition of \mathbf{K}
4. Terms are introduced and regrouped
5. The term $\boldsymbol{\mu}_x$ is introduced as $\boldsymbol{\mu}_x = \boldsymbol{\Pi}_{xx}(\mathbf{H}_x^T \mathbf{P}_{vv}^{-1} \mathbf{z} + \mathbf{P}_{xx}^{-1} \mathbf{m}_x)$
6. By noting that $\mathbf{K} = \boldsymbol{\Pi}_{xx} \mathbf{H}_x^T \mathbf{P}_{vv}^{-1}$, the standard equation for $\boldsymbol{\mu}_x$ is recovered
7. And then we go through a *lot* of manipulation...

These steps lead us to a result that we express as

$$p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) p_g(\mathbf{z}; \mathbf{H}_x \mathbf{x}, \mathbf{P}_{vv}) = |2\pi \mathbf{P}_{xx}|^{-\frac{1}{2}} |2\pi \mathbf{P}_{vv}|^{-\frac{1}{2}} \\ \times \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_x)^T \boldsymbol{\Pi}_{xx}^{-1} (\mathbf{x} - \boldsymbol{\mu}_x) \right\}$$

$$\times \exp \left\{ -\frac{1}{2}(\mathbf{z} - \mathbf{H}_x \mathbf{m}_x)^T (\mathbf{H}_x \mathbf{P}_{xx} \mathbf{H}_x^T + \mathbf{P}_{vv})^{-1} (\mathbf{z} - \mathbf{H}_x \mathbf{m}_x) \right\}$$

If we think about the objective (the statement of Ho's equation), we note that the right-hand side of the preceding expression has the correct exponentials in order to be the Gaussian distributions for $p_g(\mathbf{z}; \mathbf{H}_x \mathbf{m}_x, \mathbf{H}_x \mathbf{P}_{xx} \mathbf{H}_x^T + \mathbf{P}_{vv})$ and $p_g(\mathbf{x}; \boldsymbol{\mu}_x, \boldsymbol{\Pi}_{xx})$. The only problem is the normalization constants are not correct. We can write out the Gaussian distributions and solve for the exponential terms to rewrite the right-hand side of Ho's equation in terms of the Gaussian pdfs to yield

$$p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) p_g(\mathbf{z}; \mathbf{H}_x \mathbf{x}, \mathbf{P}_{vv}) = d p_g(\mathbf{x}; \boldsymbol{\mu}_x, \boldsymbol{\Pi}_{xx}) p_g(\mathbf{z}; \mathbf{H}_x \mathbf{m}_x, \mathbf{H}_x \mathbf{P}_{xx} \mathbf{H}_x^T + \mathbf{P}_{vv})$$

where

$$d = \frac{|2\pi \mathbf{P}_{xx}|^{-\frac{1}{2}} |2\pi \mathbf{P}_{vv}|^{-\frac{1}{2}}}{|2\pi \boldsymbol{\Pi}_{xx}|^{-\frac{1}{2}} |2\pi (\mathbf{H}_x \mathbf{P}_{xx} \mathbf{H}_x^T + \mathbf{P}_{vv})|^{-\frac{1}{2}}}$$

As such, we clearly see that if $d = 1$, we have completed our task in that we have arrived at the result we want for Ho's equation. To see if this is the case, we simplify this term according to

$$d = [|\boldsymbol{\Pi}_{xx} \mathbf{P}_{xx}^{-1}| |(\mathbf{H}_x \mathbf{P}_{xx} \mathbf{H}_x^T + \mathbf{P}_{vv}) \mathbf{P}_{vv}^{-1}|]^{\frac{1}{2}} = [|\boldsymbol{\Pi}_{xx} \mathbf{P}_{xx}^{-1}| |\mathbf{H}_x \mathbf{P}_{xx} \mathbf{H}_x^T \mathbf{P}_{vv}^{-1} + \mathbf{I}_m|]^{\frac{1}{2}}$$

This is where we use Sylvester's determinant theorem, which is given by⁷

$$|\mathbf{I}_n + \mathbf{B}\mathbf{A}| = |\mathbf{I}_p + \mathbf{A}\mathbf{B}|$$

and we can use this result to write the determinant term as

$$d = [|\boldsymbol{\Pi}_{xx} \mathbf{P}_{xx}^{-1} || \mathbf{P}_{xx} \mathbf{H}_x^T \mathbf{P}_{vv}^{-1} \mathbf{H}_x + \mathbf{I}_n |]^{\frac{1}{2}}$$

which changes the right hand side from the product of an n -dimensional determinant with an m -dimensional determinant to the product of two n -dimensional determinants. From the definition of $\boldsymbol{\Pi}_{xx}$, we have

$$\mathbf{H}_x^T \mathbf{P}_{vv}^{-1} \mathbf{H}_x = \boldsymbol{\Pi}_{xx}^{-1} - \mathbf{P}_{xx}^{-1}$$

such that the determinant term can be manipulated to yield

$$d = [|\boldsymbol{\Pi}_{xx} \mathbf{P}_{xx}^{-1} || \mathbf{P}_{xx} (\boldsymbol{\Pi}_{xx}^{-1} - \mathbf{P}_{xx}^{-1}) + \mathbf{I}_n |]^{\frac{1}{2}} = 1$$

⁷J. J. Sylvester, "On the relation between the minor determinants of linearly equivalent quadratic functions," *Philosophical Magazine*, Vol. 1, 1851, pp. 295–305.

which is exactly the result we want. That is, we have successfully shown the result of Ho's equation that

$$p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) p_g(\mathbf{z}; \mathbf{H}_x \mathbf{x}, \mathbf{P}_{vv}) = p_g(\mathbf{z}; \mathbf{H}_x \mathbf{m}_x, \mathbf{H}_x \mathbf{P}_{xx} \mathbf{H}_x^T + \mathbf{P}_{vv}) p_g(\mathbf{x}; \boldsymbol{\mu}_x, \boldsymbol{\Pi}_{xx})$$

There's one other relationship that we can get from Ho's equation that turns out to be useful, and that result comes from considering the integral of Ho's equation with respect to \mathbf{x} . This leads us to

$$\begin{aligned} & \int p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) p_g(\mathbf{z}; \mathbf{H}_x \mathbf{x}, \mathbf{P}_{vv}) d\mathbf{x} \\ &= \int p_g(\mathbf{z}; \mathbf{H}_x \mathbf{m}_x, \mathbf{H}_x \mathbf{P}_{xx} \mathbf{H}_x^T + \mathbf{P}_{vv}) p_g(\mathbf{x}; \boldsymbol{\mu}_x, \boldsymbol{\Pi}_{xx}) d\mathbf{x} \end{aligned}$$

We will leave the left-hand side alone, but on the right-hand side, the Gaussian pdf in \mathbf{z} does not involve \mathbf{x} in any way, so we can move it outside the integral to yield

$$\begin{aligned} & \int p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) p_g(\mathbf{z}; \mathbf{H}_x \mathbf{x}, \mathbf{P}_{vv}) d\mathbf{x} \\ &= p_g(\mathbf{z}; \mathbf{H}_x \mathbf{m}_x, \mathbf{H}_x \mathbf{P}_{xx} \mathbf{H}_x^T + \mathbf{P}_{vv}) \int p_g(\mathbf{x}; \boldsymbol{\mu}_x, \boldsymbol{\Pi}_{xx}) d\mathbf{x} \end{aligned}$$

Since the integrand is a valid pdf, its integral must be one, and we find that

$$\int p_g(\mathbf{x}; \mathbf{m}_x, \mathbf{P}_{xx}) p_g(\mathbf{z}; \mathbf{H}_x \mathbf{x}, \mathbf{P}_{vv}) d\mathbf{x} = p_g(\mathbf{z}; \mathbf{H}_x \mathbf{m}_x, \mathbf{H}_x \mathbf{P}_{xx} \mathbf{H}_x^T + \mathbf{P}_{vv})$$

which we will refer to as the integral form of Ho's equation.

At this point, it's worth reminding ourselves what we know and what we're doing. We know that for linear dynamics/measurements and Gaussian process/measurement noise, both the transition density and the measurement likelihood are Gaussian, such that

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}) = p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1} \mathbf{x}_{k-1}, \mathbf{P}_{ww,k-1})$$

$$p(\mathbf{z}_k | \mathbf{x}_k) = p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{x}_k, \mathbf{P}_{vv,k})$$

We also know that the Chapman-Kolmogorov equation and Bayes' rule govern the behavior of our pdf and are given by

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k | \mathbf{x}_{k-1}) p(\mathbf{x}_{k-1} | \mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1}$$

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{z}_{1:k-1})}{\int p(\mathbf{z}_k | \boldsymbol{\xi}_k) p(\boldsymbol{\xi}_k | \mathbf{z}_{1:k-1}) d\boldsymbol{\xi}_k}$$

Our objective is to show that, for a linear-Gaussian system, a Gaussian input to the Chapman-Kolmogorov equation produces a Gaussian output and that a Gaussian input to Bayes' rule produces a Gaussian output. This means that the Bayesian filtering equations close and that a Gaussian remains Gaussian under the system we're considering. To show this, we will leverage Ho's equation and the integral form of Ho's equation. We start with the assumption that the posterior pdf at $k - 1$ is Gaussian, i.e.,

$$p(\mathbf{x}_{k-1} | \mathbf{z}_{1:k-1}) = p_g(\mathbf{x}_{k-1}; \mathbf{m}_{x,k-1}^+, \mathbf{P}_{xx,k-1}^+)$$

We propagate this posterior to k using the Chapman-Kolmogorov equation under the assumption of a linear-Gaussian transition density, such that

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \int p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1}\mathbf{x}_{k-1}, \mathbf{P}_{ww,k-1}) p_g(\mathbf{x}_{k-1}; \mathbf{m}_{x,k-1}^+, \mathbf{P}_{xx,k-1}^+) d\mathbf{x}_{k-1}$$

The integral form of Ho's equation can be applied to find a solution to the Chapman-Kolmogorov equation by making the substitutions

$$\mathbf{z} \rightarrow \mathbf{x}_k \quad \mathbf{H}_x \rightarrow \mathbf{F}_{x,k-1} \quad \mathbf{P}_{vv} \rightarrow \mathbf{P}_{ww,k-1}$$

$$\mathbf{x} \rightarrow \mathbf{x}_{k-1} \quad \mathbf{m}_x \rightarrow \mathbf{m}_{x,k-1}^+ \quad \mathbf{P}_{xx} \rightarrow \mathbf{P}_{xx,k-1}^+$$

from which it follows that

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1} \mathbf{m}_{x,k-1}^+, \mathbf{F}_{x,k-1} \mathbf{P}_{xx,k-1}^+ \mathbf{F}_{x,k-1}^T + \mathbf{P}_{ww,k-1})$$

That is, if we write the prior density as

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^-, \mathbf{P}_{xx,k}^-)$$

the mean and covariance are propagated according to

$$\mathbf{m}_{x,k}^- = \mathbf{F}_{x,k-1} \mathbf{m}_{x,k-1}^+$$

$$\mathbf{P}_{xx,k}^- = \mathbf{F}_{x,k-1} \mathbf{P}_{xx,k-1}^+ \mathbf{F}_{x,k-1}^T + \mathbf{P}_{ww,k-1}$$

which are *precisely* the propagation equations for the (discrete) Kalman filter. A key difference is that these equations, while equivalent to the Kalman filter, are derived from the Chapman-Kolmogorov equation using the integral form of Ho's equation for linear, Gaussian models. This is a different set of assumptions and starting points than we used for the Kalman filter, but the outcome is the same. The main takeaway for us is that Gaussianity is preserved through the propagation step.

Now, we will consider the update step using Bayes' rule. We know that the prior is Gaussian

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^-, \mathbf{P}_{xx,k}^-)$$

To update the prior density, we apply the Gaussian prior and the Gaussian measurement likelihood function within Bayes' rule, yielding the numerator of Bayes' rule as

$$p(\mathbf{z}_k | \mathbf{x}_k)p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = p_g(\mathbf{z}_k; \mathbf{H}_{x,k}\mathbf{x}_k, \mathbf{P}_{vv,k})p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^-, \mathbf{P}_{xx,k}^-)$$

This product of Gaussians is exactly in the form of Ho's equation with

$$\mathbf{z} \rightarrow \mathbf{z}_k \quad \mathbf{H}_x \rightarrow \mathbf{H}_{x,k} \quad \mathbf{P}_{vv} \rightarrow \mathbf{P}_{vv,k}$$

$$\mathbf{x} \rightarrow \mathbf{x}_k \quad \mathbf{m}_x \rightarrow \mathbf{m}_{x,k}^- \quad \mathbf{P}_{xx} \rightarrow \mathbf{P}_{xx,k}^-$$

such that the numerator of Bayes' rule becomes

$$p(\mathbf{z}_k | \mathbf{x}_k)p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = p_g(\mathbf{z}_k; \mathbf{H}_{x,k}\mathbf{m}_{x,k}^-, \mathbf{H}_{x,k}\mathbf{P}_{xx,k}^-\mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k})p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^+, \mathbf{P}_{xx,k}^+)$$

where

$$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_{x,k}\mathbf{m}_{x,k}^-)$$

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{K}_k \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^-$$

$$\mathbf{K}_k = \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T (\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k})^{-1}$$

To complete Bayes' rule, we need to also produce an expression for the denominator of Bayes' rule. Here, we can either apply the integral form of Ho's equation or we can simply integrate the result we just obtained for the numerator of Bayes' rule. Following this latter path, it directly follows that

$$\int p(\mathbf{z}_k \mid \boldsymbol{\xi}_k) p(\boldsymbol{\xi}_k \mid \mathbf{z}_{1:k-1}) d\boldsymbol{\xi}_k = p_g(\mathbf{z}_k; \mathbf{H}_{x,k}\mathbf{m}_{x,k}^-, \mathbf{H}_{x,k}\mathbf{P}_{xx,k}^-\mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k})$$

This result warrants a bit of consideration before continuing. At first glance, and in the context of Ho's equation, this is a pdf. It is a density in \mathbf{z}_k with some mean and covariance. The mean and covariance are computed based on the measurement model, measurement noise statistics, and the prior state mean and covariance. When put in the context of a measurement update, however, this pdf becomes a function evaluated at the

received measurement, \mathbf{z}_k . Therefore, we should not think about this as a pdf when we process a measurement; instead, it will just be a real-valued number.

Given the numerator and denominator of Bayes' rule, we can now formulate the posterior or filtering density as

$$\begin{aligned} p(\mathbf{x}_k | \mathbf{z}_{1:k}) &= \frac{p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{m}_{x,k}^-, \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k}) p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^+, \mathbf{P}_{xx,k}^+)}{p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{m}_{x,k}^-, \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k})} \\ &= p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^+, \mathbf{P}_{xx,k}^+) \end{aligned}$$

from which we conclude that the posterior distribution is Gaussian. We have again arrived at an algorithmic equivalent to the update stage of the Kalman filter. Once again, it is important to recognize that we have done so through entirely different means and using different assumptions.

What we have now shown is that if we start with a Gaussian posterior at $k - 1$, the prior at k , which is obtained using the Chapman-Kolmogorov equation, is also Gaussian. Bayes' rule ingests this Gaussian prior and returns a Gaussian posterior at k . This closes our recursion, so we can keep going through subsequent prediction and correction steps

knowing that the density will remain Gaussian.

To summarize:

- Given the linear system as the state-space model

$$\mathbf{x}_k = \mathbf{F}_{x,k-1} \mathbf{x}_{k-1} + \mathbf{w}_{k-1}$$

$$\mathbf{z}_k = \mathbf{H}_{x,k} \mathbf{x}_k + \mathbf{v}_k$$

make the assumption that the noises are zero-mean and Gaussian and transform the model to the probabilistic model

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}) = p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1} \mathbf{x}_{k-1}, \mathbf{P}_{ww,k-1})$$

$$p(\mathbf{z}_k | \mathbf{x}_k) = p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{x}_k, \mathbf{P}_{vv,k})$$

- Initialize a Gaussian pdf with mean and covariance

$$\mathbf{m}_{x,k-1}^+ = \mathbf{m}_{x,0} \quad \text{and} \quad \mathbf{P}_{xx,k-1}^+ = \mathbf{P}_{xx,0}$$

- Compute the predicted pdf using the Chapman-Kolmogorov equation

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^-, \mathbf{P}_{xx,k}^-)$$

where

$$\mathbf{m}_{x,k}^- = \mathbf{F}_{x,k-1} \mathbf{m}_{x,k-1}^+$$

$$\mathbf{P}_{xx,k}^- = \mathbf{F}_{x,k-1} \mathbf{P}_{xx,k-1}^+ \mathbf{F}_{x,k-1}^T + \mathbf{P}_{ww,k-1}$$

- Compute the filtered pdf using Bayes' rule

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) = p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^+, \mathbf{P}_{xx,k}^+)$$

where

$$\mathbf{m}_{x,k}^+ = \mathbf{m}_{x,k}^- + \mathbf{K}_k (\mathbf{z}_k - \mathbf{H}_{x,k} \mathbf{m}_{x,k}^-)$$

$$\mathbf{P}_{xx,k}^+ = \mathbf{P}_{xx,k}^- - \mathbf{K}_k \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^-$$

$$\mathbf{K}_k = \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T (\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^- \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k})^{-1}$$

6.4 The Gaussian Mixture Filter

We have a powerful framework with Bayesian filtering. Instead of working with mean and covariance as we did under the Kalman filtering framework, we can now work with the entire distribution. Unfortunately, being able to work with the entire distribution means that we have to have the means by which to work with the entire distribution.

A naive approach would be to grid the state space and look at the pdf at each grid point within the state space. This often works with one-dimensional problems, but it gets much too computationally complex for higher dimensional systems. Moreover, we do not necessarily know a bounded region for which we should compute the pdf, so we even have difficulty generating the grid. We therefore need a systematic approach for handling the entire pdf that allows us to extend into higher dimensions without requiring bounds on the state space. It would also be fantastic if we could leverage aspects of our knowledge of the Kalman filter, including various alterations and extensions.

The seminal paper by Sorenson and Alspach⁸ paved the way just 11 years after Kalman's

⁸Sorenson, H. W. and Alspach, D. L., "Recursive Bayesian Estimation Using Gaussian Sums," *Automatica*, Vol. 7, 1971, pp. 465–479.

work. The idea is to approximate the distribution by a finite number of weighted Gaussian pdfs. For instance, we could approximate the posterior density at $k - 1$ as

$$p(\mathbf{x}_{k-1} | \mathbf{z}_{1:k-1}) = \sum_{\ell=1}^{L_{x,k-1}^+} w_{x,k-1}^{(\ell)+} p_g(\mathbf{x}_{k-1}; \mathbf{m}_{x,k-1}^{(\ell)+}, \mathbf{P}_{xx,k-1}^{(\ell)+})$$

where $L_{x,k-1}^+$ is the number of components in the mixture, $w_{x,k-1}^{(\ell)+}$ represents the weight of the ℓ^{th} component, and $\mathbf{m}_{x,k-1}^{(\ell)+}$ and $\mathbf{P}_{xx,k-1}^{(\ell)+}$ denote the mean and covariance, respectively, that define the ℓ^{th} Gaussian within the mixture.

When we talk about pdfs, we have a few conditions that we need to satisfy. As such, there are a few restrictions that need to be imposed in order to ensure that the Gaussian sum (or Gaussian mixture) representation represents a pdf. It should probably not be any surprise that there are some restrictions for this to be a valid model. First of all, we need to respect any restrictions on the Gaussian components. This means that each covariance matrix should be symmetric and positive definite. Additionally, we need the entire mixture to describe a valid pdf. This means that both sides of the preceding equation need to integrate to one. On the left hand side, we have a generic pdf, so we can

conclude that this integrates to one. Therefore, focusing on the right hand side, imposing this conditions leads us to conclude that

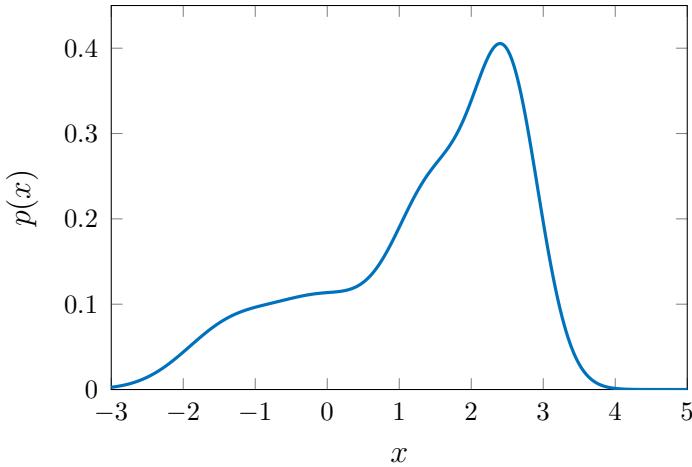
$$1 = \int \sum_{\ell=1}^{L_{x,k-1}^+} w_{x,k-1}^{(\ell)+} p_g(\mathbf{x}_{k-1}; \boldsymbol{m}_{x,k-1}^{(\ell)+}, \boldsymbol{P}_{xx,k-1}^{(\ell)+}) d\mathbf{x}_{k-1} = \sum_{\ell=1}^{L_{x,k-1}^+} w_{x,k-1}^{(\ell)+}$$

which leverages the fact that each Gaussian integrates to one and assumes that the weights are independent of the state. While it is not strictly required that the weights are independent of the state, this produces a convenient way of handling Gaussian mixture (GM) approximations. If we impose the additional constraint that all of the weights are non-negative, it also guarantees that the pdf is strictly non-negative.

The requirement that the weights sum to one is a requirement for the GM to represent a pdf, but when can we actually make the approximation? This is one of the fundamental questions addressed in the paper by Sorenson and Alspach:

“It has been shown that a probability density function can be approximated arbitrarily closely except at discontinuities by such a [G]aussian sum.”

For instance, let's consider a reasonably well-behaved (in the sense that it contains no discontinuities) pdf that is illustrated as



What Sorenson and Alspach showed is that we can make the GM approximation for a very large class of pdfs. To give an intuitive understanding to why this approximation is possible, consider an L_x -component GM approximation to the pdf $p(\mathbf{x})$, given by

$$p(\mathbf{x}) = \sum_{\ell=1}^{L_x} w_x^{(\ell)} p_g(\mathbf{x}; \mathbf{m}_x^{(\ell)}, \mathbf{P}_{xx}^{(\ell)})$$

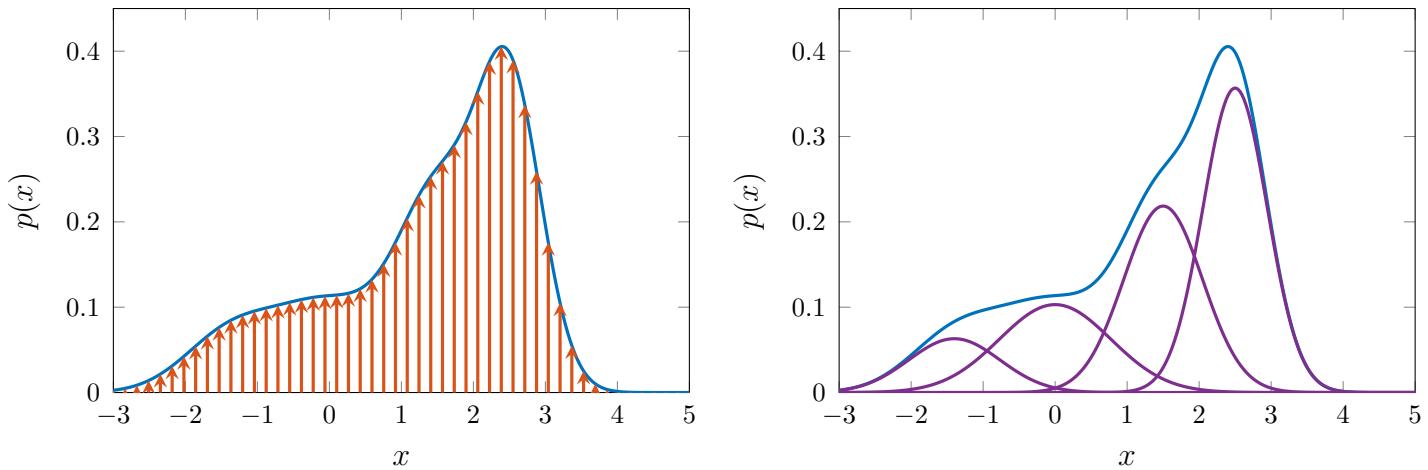
Let's make the assumption that all of the components have the same covariance, such that $\mathbf{P}_{xx}^{(\ell)} = \mathbf{P}_{xx}$. This condition of a mixture is what we call homoscedasticity. Let's further assume that $\mathbf{P}_{xx} = \sigma_x^2 \mathbf{I}_n$, which is referred to as isotropic, such that our GM approximation becomes

$$p(\mathbf{x}) = \sum_{\ell=1}^{L_x} w_x^{(\ell)} p_g(\mathbf{x}; \mathbf{m}_x^{(\ell)}, \sigma_x^2 \mathbf{I}_n)$$

As $\sigma_x \rightarrow 0^+$, the Gaussian components collapse to Dirac delta distributions; that is

$$\sum_{\ell=1}^{L_x} w_x^{(\ell)} p_g(\mathbf{x}; \mathbf{m}_x^{(\ell)}, \sigma_x^2 \mathbf{I}_n) \longrightarrow \sum_{\ell=1}^{L_x} w_x^{(\ell)} \delta(\mathbf{x} - \mathbf{m}_x^{(\ell)})$$

and we can think of the approximation as a particle-based sampling method. Intuitively, it is straightforward to fill up the area under $p(\mathbf{x})$ with a bunch of particles, and so we gain an interpretation that if we choose a sufficiently large number of components with small enough covariances, we can represent a wide variety of pdfs. An illustration for the previously-considered pdf using a set of Dirac distributions is shown in the following figure on the left.



As we add width back into the components, we can use fewer mixture elements, and in this case, as illustrated in the preceding figure on the right, we can make a very good approximation using only four Gaussian components.

We now know that it is *possible* to form a GM approximation of almost any pdf, but actually forming the approximation is rather difficult and highly problem-dependent. For the time being, we will assume that we are given an initial pdf as a Gaussian mixture and develop a mechanization for Bayesian filtering that uses the Gaussian mixture. The

resulting algorithm is what we will call the Gaussian mixture filter (GMF). To develop the GMF, we will show that a GM pdf closes the recursion formed by application of the Chapman-Kolmogorov equation and Bayes' rule for linear-Gaussian systems.

We start with the same state-space system, given by

$$\mathbf{x}_k = \mathbf{F}_{x,k-1} \mathbf{x}_{k-1} + \mathbf{w}_{k-1}$$

$$\mathbf{z}_k = \mathbf{H}_{x,k} \mathbf{x}_k + \mathbf{v}_k$$

where the process and measurement noises are both zero-mean and Gaussian, such that we have the equivalent probabilistic models

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}) = p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1} \mathbf{x}_{k-1}, \mathbf{P}_{ww,k-1})$$

$$p(\mathbf{z}_k | \mathbf{x}_k) = p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{x}_k, \mathbf{P}_{vv,k})$$

Let the filtering density at $k - 1$ be given by an $L_{x,k-1}^+$ -component GM of the form

$$p(\mathbf{x}_{k-1} | \mathbf{z}_{1:k-1}) = \sum_{\ell=1}^{L_{x,k-1}^+} w_{x,k-1}^{(\ell)+} p_g(\mathbf{x}_{k-1}; \mathbf{m}_{x,k-1}^{(\ell)+}, \mathbf{P}_{xx,k-1}^{(\ell)+})$$

To find the predicted density, we need to apply the Chapman-Kolmogorov equation

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k | \mathbf{x}_{k-1}) p(\mathbf{x}_{k-1} | \mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1}$$

We have the transition density and the previous posterior, so we can apply those known relationships to the right-hand side of the Chapman-Kolmogorov equation to yield

$$\begin{aligned} p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) &= \int \left[p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1} \mathbf{x}_{k-1}, \mathbf{P}_{ww,k-1}) \right. \\ &\quad \times \left. \sum_{\ell=1}^{L_{x,k-1}^+} w_{x,k-1}^{(\ell)+} p_g(\mathbf{x}_{k-1}; \mathbf{m}_{x,k-1}^{(\ell)+}, \mathbf{P}_{xx,k-1}^{(\ell)+}) \right] d\mathbf{x}_{k-1} \end{aligned}$$

We can rearrange this to put things into an easier form as

$$\begin{aligned} p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) &= \sum_{\ell=1}^{L_{x,k-1}^+} w_{x,k-1}^{(\ell)+} \int p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1} \mathbf{x}_{k-1}, \mathbf{P}_{ww,k-1}) \\ &\quad \times p_g(\mathbf{x}_{k-1}; \mathbf{m}_{x,k-1}^{(\ell)+}, \mathbf{P}_{xx,k-1}^{(\ell)+}) d\mathbf{x}_{k-1} \end{aligned}$$

The integral in the preceding equation fits exactly into the integral form of Ho's equation, if we make the following substitutions

$$\boldsymbol{z} \rightarrow \boldsymbol{x}_k \quad \boldsymbol{H}_x \rightarrow \boldsymbol{F}_{x,k-1} \quad \boldsymbol{P}_{vv} \rightarrow \boldsymbol{P}_{ww,k-1}$$

$$\boldsymbol{x} \rightarrow \boldsymbol{x}_{k-1} \quad \boldsymbol{m}_x \rightarrow \boldsymbol{m}_{x,k-1}^{(\ell)+} \quad \boldsymbol{P}_{xx} \rightarrow \boldsymbol{P}_{xx,k-1}^{(\ell)+}$$

from which it follows that we can express the predictive (or prior) pdf as

$$p(\boldsymbol{x}_k | \boldsymbol{z}_{1:k-1}) = \sum_{\ell=1}^{L_{x,k-1}^+} w_{x,k-1}^{(\ell)+} p_g(\boldsymbol{x}_k; \boldsymbol{F}_{x,k-1} \boldsymbol{m}_{x,k-1}^{(\ell)+}, \boldsymbol{F}_{x,k-1} \boldsymbol{P}_{xx,k-1}^{(\ell)+} \boldsymbol{F}_{x,k-1}^T + \boldsymbol{P}_{ww,k-1})$$

Alternatively, we can write this pdf as the GM

$$p(\boldsymbol{x}_k | \boldsymbol{z}_{1:k-1}) = \sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(\ell)-} p_g(\boldsymbol{x}_k; \boldsymbol{m}_{x,k}^{(\ell)-}, \boldsymbol{P}_{xx,k}^{(\ell)-})$$

where the parameters of the GM are given by

$$L_{x,k}^- = L_{x,k-1}^+$$

$$w_{x,k}^{(\ell)-} = w_{x,k-1}^{(\ell)+}$$

$$\boldsymbol{m}_{x,k}^{(\ell)-} = \boldsymbol{F}_{x,k-1} \boldsymbol{m}_{x,k-1}^{(\ell)+}$$

$$\boldsymbol{P}_{xx,k}^{(\ell)-} = \boldsymbol{F}_{x,k-1} \boldsymbol{P}_{xx,k-1}^{(\ell)+} \boldsymbol{F}_{x,k-1}^T + \boldsymbol{P}_{ww,k-1}$$

In essence, this is a set of mean and covariance propagations that looks like a set of parallel Kalman filter propagation steps, but the predictive density now represents the entire non-Gaussian pdf. In performing the propagation, the number of components is constant, and the weights of the components remain constant.

Now, we need to consider Bayes' rule, which, as a reminder, is given by

$$p(\boldsymbol{x}_k | \boldsymbol{z}_{1:k}) = \frac{p(\boldsymbol{z}_k | \boldsymbol{x}_k) p(\boldsymbol{x}_k | \boldsymbol{z}_{1:k-1})}{\int p(\boldsymbol{z}_k | \boldsymbol{\xi}_k) p(\boldsymbol{\xi}_k | \boldsymbol{z}_{1:k-1}) d\boldsymbol{\xi}_k}$$

when we have a Gaussian likelihood and a GM prior pdf. Starting with the numerator,

$$p(\boldsymbol{z}_k | \boldsymbol{x}_k) p(\boldsymbol{x}_k | \boldsymbol{z}_{1:k-1}) = p_g(\boldsymbol{z}_k; \boldsymbol{H}_{x,k} \boldsymbol{x}_k, \boldsymbol{P}_{vv,k}) \sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(\ell)-} p_g(\boldsymbol{x}_k; \boldsymbol{m}_{x,k}^{\ell-}, \boldsymbol{P}_{xx,k}^{\ell-})$$

After a simple rearrangement, the product can be written as

$$p(\mathbf{z}_k | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(\ell)-} p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{x}_k, \mathbf{P}_{vv,k}) p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{\ell-}, \mathbf{P}_{xx,k}^{\ell-})$$

This is exactly in the form of Ho's equation with

$$\begin{aligned} \mathbf{z} &\rightarrow \mathbf{z}_k & \mathbf{H}_x &\rightarrow \mathbf{H}_{x,k} & \mathbf{P}_{vv} &\rightarrow \mathbf{P}_{vv,k} \\ \mathbf{x} &\rightarrow \mathbf{x}_k & \mathbf{m}_x &\rightarrow \mathbf{m}_{x,k}^{(\ell)-} & \mathbf{P}_{xx} &\rightarrow \mathbf{P}_{xx,k}^{(\ell)-} \end{aligned}$$

such that the numerator of Bayes' rule becomes

$$\begin{aligned} p(\mathbf{z}_k | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) &= \sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(\ell)-} p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-}, \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k}) \\ &\quad \times p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(\ell)+}, \mathbf{P}_{xx,k}^{(\ell)+}) \end{aligned}$$

where

$$\mathbf{m}_{x,k}^{(\ell)+} = \mathbf{m}_{x,k}^{(\ell)-} + \mathbf{K}_k^{(\ell)} (\mathbf{z}_k - \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-})$$

$$\mathbf{P}_{xx,k}^{(\ell)+} = \mathbf{P}_{xx,k}^{(\ell)-} - \mathbf{K}_k^{(\ell)} \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-}$$

$$\mathbf{K}_k^{(\ell)} = \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T (\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k})^{-1}$$

Next, we consider the denominator of Bayes' rule, which, as with the Bayesian Kalman filter, can either be computed using the integral form of Ho's equation or by integrating the numerator of Bayes' rule. Following the latter option, it follows that

$$\int p(\mathbf{z}_k | \boldsymbol{\xi}_k) p(\boldsymbol{\xi}_k | \mathbf{z}_{1:k-1}) d\boldsymbol{\xi}_k = \sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(\ell)-} p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-}, \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k})$$

Assembling the pieces of Bayes' rule produces the posterior (or filtering) pdf as

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) = \frac{\sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(\ell)-} p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-}, \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k}) p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(\ell)+}, \mathbf{P}_{xx,k}^{(\ell)+})}{\sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(\ell)-} p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-}, \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k})}$$

Unlike the Bayesian Kalman filter, we cannot cancel the $p_g(\mathbf{z}_k; \cdot, \cdot)$ terms that appear in both the numerator and denominator. This is because they depend on the components of the GM and “tie together” the components in some way. To see how this gets enacted, it is useful to define some new terms to simplify notation; as such, let

$$k_k^{(\ell)} = p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-}, \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k})$$

which we will refer to as the weight gain. Note that when we actually receive and process a measurement, this is just a number and not a pdf. With the definition of $k_k^{(\ell)}$, the filtering density becomes

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) = \frac{\sum_{\ell=1}^{L_{x,k}} w_{x,k}^{(\ell)-} k_k^{(\ell)} p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(\ell)+}, \mathbf{P}_{xx,k}^{(\ell)+})}{\sum_{\ell=1}^{L_{x,k}} w_{x,k}^{(\ell)-} k_k^{(\ell)}}$$

If we define a set of posterior weights as

$$w_{x,k}^{(\ell)+} = \frac{k_k^{(\ell)} w_{x,k}^{(\ell)-}}{\sum_{i=1}^{L_{x,k}} k_k^{(i)} w_{x,k}^{(i)-}}$$

then it follows that the filtering density can be written in the form

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) = \sum_{\ell=1}^{L_{x,k}^+} w_{x,k}^{(\ell)+} p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(\ell)+}, \mathbf{P}_{xx,k}^{(\ell)+})$$

where $L_{x,k}^+ = L_{x,k}^-$.

As with the prediction step, we see a remarkable resemblance to a set of parallel operating Kalman filters. The key difference is that these parallel Kalman filters are “tied” together using the likelihood-based update for the component weights. It is still very important to keep in mind that these components work together to form a single pdf; as such, this is not actually a bank of Kalman filters. It is a much more general structure.

We should also not confuse the GM filter with other similar structures, like the interacting multiple model (IMM) approach or multiple hypothesis filter (MHF) approaches. These different techniques can have some similarities in how they treat elemental pieces of their construction, but the overarching premise of each method is completely different.

What we have now shown is that if we start with a posterior GM at $k - 1$, the ap-

plication of the transition dynamics via the Chapman-Kolmogorov equation produces a prior GM at k , and the application of the measurement likelihood via Bayes' rule produces a posterior GM at k . We can therefore conclude that the recursion closes, such that if we start with a GM representation of the pdf at the initial step, we can rest assured that it will remain a GM pdf through all of the steps. In fact, for the linear-Gaussian model that we are using, the GM will even retain the same number of components at every step of the filter implementation.

To summarize:

- Given the linear system as the state-space model

$$\boldsymbol{x}_k = \boldsymbol{F}_{x,k-1} \boldsymbol{x}_{k-1} + \boldsymbol{w}_{k-1}$$

$$\boldsymbol{z}_k = \boldsymbol{H}_{x,k} \boldsymbol{x}_k + \boldsymbol{v}_k$$

make the assumption that the noises are zero-mean and Gaussian and transform the model to the probabilistic model

$$p(\boldsymbol{x}_k | \boldsymbol{x}_{k-1}) = p_g(\boldsymbol{x}_k; \boldsymbol{F}_{x,k-1} \boldsymbol{x}_{k-1}, \boldsymbol{P}_{ww,k-1})$$

$$p(\boldsymbol{z}_k | \boldsymbol{x}_k) = p_g(\boldsymbol{z}_k; \boldsymbol{H}_{x,k} \boldsymbol{x}_k, \boldsymbol{P}_{vv,k})$$

- Initialize an L_0 -component GM pdf with weights, means, and covariances

$$L_{x,k-1}^+ = L_0, \quad w_{x,k-1}^{(\ell)+} = w_0^{(\ell)}, \quad \boldsymbol{m}_{x,k-1}^{(\ell)+} = \boldsymbol{m}_{x,0}^{(\ell)}, \quad \boldsymbol{P}_{xx,k-1}^{(\ell)+} = \boldsymbol{P}_{xx,0}^{(\ell)}$$

- Compute the predicted pdf using the Chapman-Kolmogorov equation

$$p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1}) = \sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(\ell)-} p_g(\boldsymbol{x}_k; \boldsymbol{m}_{x,k}^{(\ell)-}, \boldsymbol{P}_{xx,k}^{(\ell)-})$$

where

$$L_{x,k}^- = L_{x,k-1}^+$$

$$w_{x,k}^{(\ell)-} = w_{x,k-1}^{(\ell)+}$$

$$\boldsymbol{m}_{x,k}^{(\ell)-} = \boldsymbol{F}_{x,k-1} \boldsymbol{m}_{x,k-1}^{(\ell)+}$$

$$\boldsymbol{P}_{xx,k}^{(\ell)-} = \boldsymbol{F}_{x,k-1} \boldsymbol{P}_{xx,k-1}^{(\ell)+} \boldsymbol{F}_{x,k-1}^T + \boldsymbol{P}_{ww,k-1}$$

- Compute the filtered pdf using Bayes' rule

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) = \sum_{\ell=1}^{L_{x,k}^+} w_{x,k}^{(\ell)+} p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(\ell)+}, \mathbf{P}_{xx,k}^{(\ell)+})$$

where

$$L_{x,k}^+ = L_{x,k}^-$$

$$w_{x,k}^{(\ell)+} = k_k^{(\ell)} w_{x,k}^{(\ell)-} / \sum_{i=1}^{L_{x,k}^-} k_k^{(i)} w_k^{(i)-}$$

$$\mathbf{m}_{x,k}^{(\ell)+} = \mathbf{m}_{x,k}^{(\ell)-} + \mathbf{K}_k^{(\ell)} (\mathbf{z}_k - \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-})$$

$$\mathbf{P}_{xx,k}^{(\ell)+} = \mathbf{P}_{xx,k}^{(\ell)-} - \mathbf{K}_k^{(\ell)} \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-}$$

$$k_k^{(\ell)} = p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-}, \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k})$$

$$\mathbf{K}_k^{(\ell)} = \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T (\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv})^{-1}$$

It is worth noting that we have used the “textbook” form of the covariance update

throughout our dealing with the Bayesian Kalman filter and the Gaussian mixture filter. We already know that this form of the covariance update is equivalent to many other forms algebraically, but that they do not all have the same numerical performance. The textbook form is used here to be consistent with literature, but there is no reason why one cannot substitute any of the other forms of the update and make use of them directly.

There's also something really neat about the GMF, other than the fact that we can now work with pdfs. It has a form of built-in data association. The $k_k^{(\ell)}$ terms tell us, relatively speaking, how likely it is that the ℓ^{th} component generated the data \mathbf{z}_k . This is very convenient because it means that components with very low likelihood get downweighted, and the number of mixands can usually be reduced to simplify the computational burden.

At the same time, this is also dangerous. If we recall our discussion on measurement editing, we had to edit out data that didn't agree with our filtering solution because the Kalman filter doesn't have a mechanism for detecting bad data. The GMF has a mechanism, but we need to be careful. If we receive a measurement that does not agree with any of the components in the mixture, every weight gain will be zero (more precisely, they will be very small, and may become zero numerically). Therefore, the unnormalized posterior weights will end up being zero, and, if we're not careful, we could end up di-

viding by zero and losing our filtering solution completely. As such, we need to be smart and if there is no agreement (each unnormalized posterior weight is zero), we reject the measurement and take the prior as the posterior.

Another topic that requires some discussion is how to use the GM pdf to produce an estimate of the state. When we work with Kalman filters, we produce an estimated state and a measure of our confidence/uncertainty in the estimated state, which is the estimation error covariance matrix. For the GMF, we have an entire pdf, and this means that there are different quantities that we can define as our estimates. For instance, if the pdf we want to extract an estimate from is given by $p(\mathbf{x} | \mathbf{z})$, we could

1. maximize the probability

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x}} p(\mathbf{x} | \mathbf{z}) \quad \rightarrow \quad \hat{\mathbf{x}} = \text{mode of } p(\mathbf{x} | \mathbf{z})$$

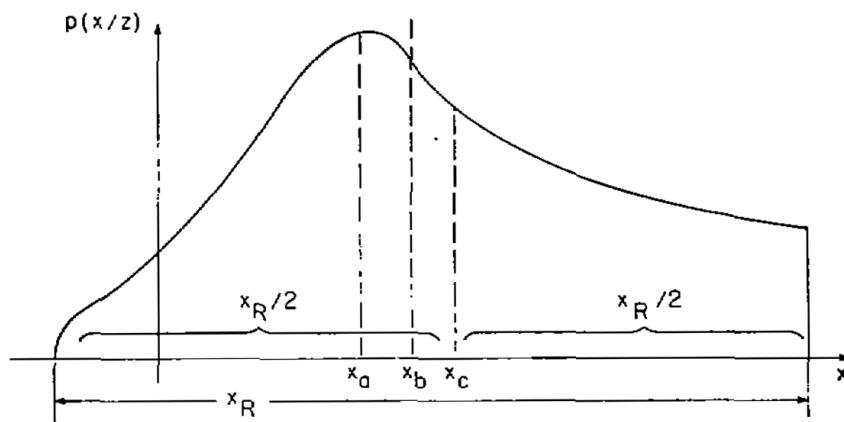
2. minimize the mean square error

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} \int \|\mathbf{x} - \hat{\mathbf{x}}\|^2 p(\mathbf{x} | \mathbf{z}) d\mathbf{x} \quad \rightarrow \quad \hat{\mathbf{x}} = \text{mean of } p(\mathbf{x} | \mathbf{z}) = \mathbb{E}\{\mathbf{x} | \mathbf{z}\}$$

3. minimize the maximum deviation

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} \max | \mathbf{x} - \hat{\mathbf{x}} | \quad \rightarrow \quad \hat{\mathbf{x}} = \text{median of } p(\mathbf{x} | \mathbf{z})$$

Each of these options is illustrated as



x_a - Most Probable Estimate

x_b - Conditional Mean Estimate

x_c - Minimax Estimate

The mode can be found, and it is arguably the most useful estimate. Unfortunately, there is no known closed-form solution for the mode of a GM, which means that we are left using numerical optimization methods. As you might imagine, for the general case, this can require a bit of computational power, so the mode is not often used.

In comparison, the mean of a GM pdf is actually quite easy to compute, so we usually consider that to be our estimate. The pdf in question can either be the prior or the posterior pdf, so we will just write it in general as

$$p(\mathbf{x} | \mathbf{z}) = \sum_{\ell=1}^{L_x} w_x^{(\ell)} p_g(\mathbf{x}; \mathbf{m}_x^{(\ell)}, \mathbf{P}_{xx}^{(\ell)})$$

From the condition for this to be a valid pdf, we know that

$$\int p(\mathbf{x} | \mathbf{z}) d\mathbf{x} = 1 \quad \rightarrow \quad \sum_{\ell=1}^{L_x} w_x^{(\ell)} = 1$$

where we also usually impose the additional constraint that $w_x^{(\ell)} \geq 0$ to ensure that the pdf remains nonnegative everywhere. This can be thought of as the zeroth moment of

the distribution.

The first moment of the distribution, or the mean, is given by

$$\mathbf{m}_x = \text{E}\{\mathbf{x}|\mathbf{z}\} = \int \mathbf{x} p(\mathbf{x}|\mathbf{z}) d\mathbf{x}$$

such that substituting for the form of the conditional pdf as a GM yields

$$\mathbf{m}_x = \sum_{\ell=1}^{L_x} w_x^{(\ell)} \int \mathbf{x} p_g(\mathbf{x}; \mathbf{m}_x^{(\ell)}, \mathbf{P}_{xx}^{(\ell)}) d\mathbf{x}$$

where we have brought the integral inside the summation and taken the weight to be state-independent. By definition, the integral evaluates to the ℓ^{th} mean, yielding

$$\mathbf{m}_x = \sum_{\ell=1}^{L_x} w_x^{(\ell)} \mathbf{m}_x^{(\ell)}$$

That is, the mean of the *entire* GM pdf is given by a weighted sum of the means of each of the components that comprise the GM.

While we're computing moments, we could also think about the second moment of the distribution, or the conditional covariance of the GM representation of the pdf. Dropping the conditional dependence on \mathbf{z} for notational simplicity, the covariance is defined as

$$\mathbf{P}_{xx} = \text{E}\{(\mathbf{x} - \mathbf{m}_x)(\mathbf{x} - \mathbf{m}_x)^T\} = \text{E}\{\mathbf{x}\mathbf{x}^T\} - \mathbf{m}_x\mathbf{m}_x^T$$

where \mathbf{m}_x is the overall mean of the conditional pdf. In terms of the expectation integral, the covariance is given by

$$\mathbf{P}_{xx} = \int \mathbf{x}\mathbf{x}^T p(\mathbf{x} | \mathbf{z}) d\mathbf{x} - \mathbf{m}_x\mathbf{m}_x^T$$

which, making use of the GM form of the conditional density, yields

$$\mathbf{P}_{xx} = \sum_{\ell=1}^{L_x} w_x^{(\ell)} \int \mathbf{x}\mathbf{x}^T p_g(\mathbf{x}; \mathbf{m}_x^{(\ell)}, \mathbf{P}_{xx}^{(\ell)}) d\mathbf{x} - \mathbf{m}_x\mathbf{m}_x^T$$

The integral evaluates to the raw second moment of the component-wise Gaussians, or

$$\int \mathbf{x}\mathbf{x}^T p_g(\mathbf{x}; \mathbf{m}_x^{(\ell)}, \mathbf{P}_{xx}^{(\ell)}) d\mathbf{x} = \mathbf{P}_{xx}^{(\ell)} + \mathbf{m}_x^{(\ell)}(\mathbf{m}_x^{(\ell)})^T$$

such that the conditional covariance is

$$\mathbf{P}_{xx} = \sum_{\ell=1}^{L_x} w_x^{(\ell)} (\mathbf{P}_{xx}^{(\ell)} + \mathbf{m}_x^{(\ell)} (\mathbf{m}_x^{(\ell)})^T) - \mathbf{m}_x \mathbf{m}_x^T$$

This can also be equivalently expressed as

$$\mathbf{P}_{xx} = \sum_{\ell=1}^{L_x} w_x^{(\ell)} (\mathbf{P}_{xx}^{(\ell)} + (\mathbf{m}_x^{(\ell)} - \mathbf{m}_x)(\mathbf{m}_x^{(\ell)} - \mathbf{m}_x)^T)$$

Thus, given the conditional pdf, which can either be prior to or after the inclusion of measurement data, the mean and covariance can easily be determined in order to be used as an estimate and a measure of the confidence in the estimate. Typically, the conditional mean of the GM pdf, \mathbf{m}_x , is used to determine the estimation error, if the true state is known. Likewise, the conditional covariance of the GM pdf, \mathbf{P}_{xx} , is used to establish confidence intervals. In this way, the manner of depicting filter performance is very similar to the methods that we have used with the various forms of the Kalman filter.

6.4.1 Example of the GMF

Let's take a look at an example of the GMF, where the dynamics of the system are taken to be given by

$$\boldsymbol{x}_k = \boldsymbol{F}_{x,k-1} \boldsymbol{x}_{k-1} + \boldsymbol{w}_{k-1}$$

$$\boldsymbol{z}_k = \boldsymbol{H}_{x,k} \boldsymbol{x}_k + v_k$$

with

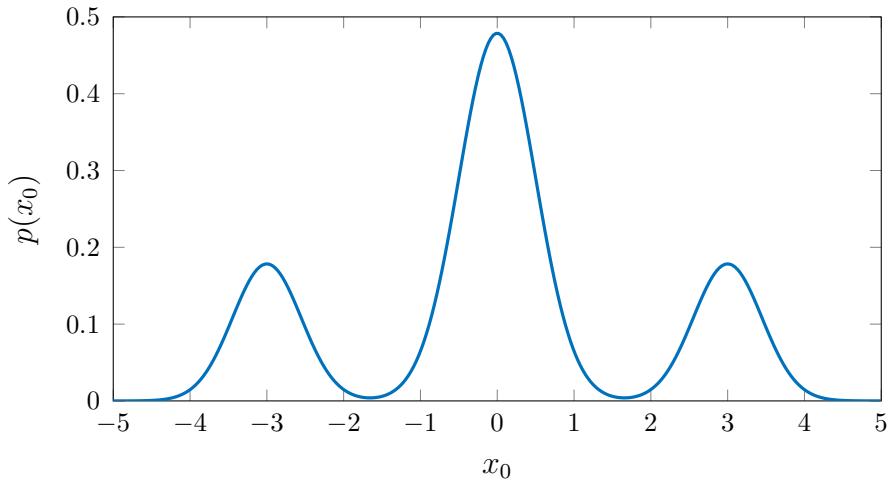
$$\boldsymbol{F}_{x,k-1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad \boldsymbol{H}_{x,k} = [1 \ 0]$$

which is, essentially, a constant velocity model with position-only measurements. That is, we interpret the first state as the position and the second state as the velocity.

To keep things relatively interesting, we take the initial pdf to be a GM of the form

$$p(\boldsymbol{x}_0) = \sum_{\ell=1}^{L_{x,0}} w_{x,0}^{(\ell)} p_g(\boldsymbol{x}_0; \boldsymbol{m}_{x,0}^{(\ell)}, \boldsymbol{P}_{xx,0}^{(\ell)})$$

The initial pdf in the position state is illustrated in the following figure.



This pdf is an $L_{x,0} = 3$ component GM with weights, means, and covariances of the Gaussian components that are given by

$$w_{x,0}^{(1)} = 0.2, \quad w_{x,0}^{(2)} = 0.6, \quad w_{x,0}^{(3)} = 0.2$$

$$\mathbf{m}_{x,0}^{(1)} = \begin{bmatrix} -3 \\ 0.15 \end{bmatrix}, \quad \mathbf{m}_{x,0}^{(2)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \mathbf{m}_{x,0}^{(3)} = \begin{bmatrix} 3 \\ -0.15 \end{bmatrix}$$

$$\boldsymbol{P}_{xx,0}^{(1)} = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.01 \end{bmatrix}, \quad \boldsymbol{P}_{xx,0}^{(2)} = \begin{bmatrix} 0.25 & 0 \\ 0 & 0.01 \end{bmatrix}, \quad \boldsymbol{P}_{xx,0}^{(3)} = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.01 \end{bmatrix}$$

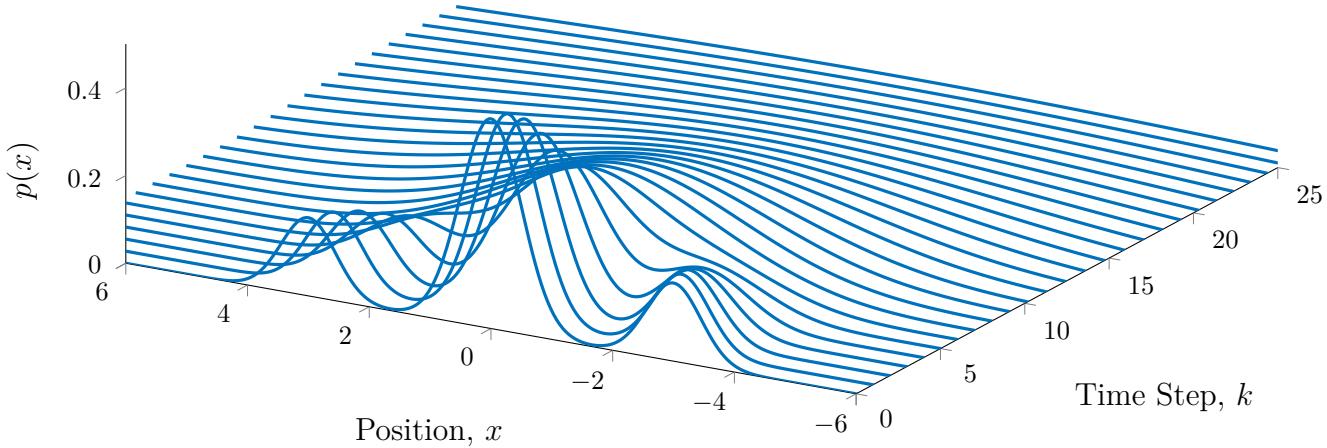
For the GMF, the system model has to be linear and Gaussian, which means that both the process noise and measurement noise are modeled as Gaussian random variables. As such, we take the process noise and measurement noise pdfs to be Gaussians of the form

$$p(\boldsymbol{w}_{k-1}) = p_g(\boldsymbol{w}_{k-1}; \mathbf{0}_w, \boldsymbol{P}_{ww,k-1}) \quad \text{and} \quad p(v_k) = p_g(v_k; 0, P_{vv,k})$$

where

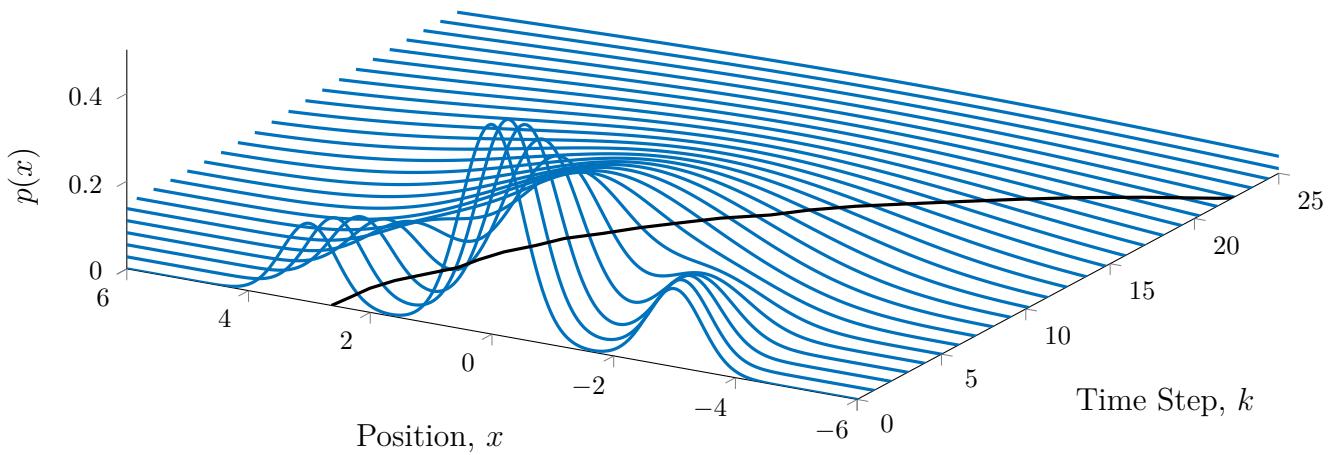
$$\boldsymbol{P}_{ww,k-1} = \begin{bmatrix} 0 & 0 \\ 0 & 0.01 \end{bmatrix} \quad \text{and} \quad P_{vv,k} = 0.5$$

To gain a bit of understanding of how GM pdfs behave and evolve, let's first consider the case where we simply propagate the GM pdf for a series of 30 time steps. The results of this propagation-only implementation are provided in the following figure for the position state. The initial GM pdf is clearly visible in this depiction, but it is also clear that the three-component structure quickly fades away. This is partially due to the injection of process noise and partially due to the velocity uncertainty leading to position uncertainty. Even though the components become indistinguishable, the pdf is always described by a three-component GM.



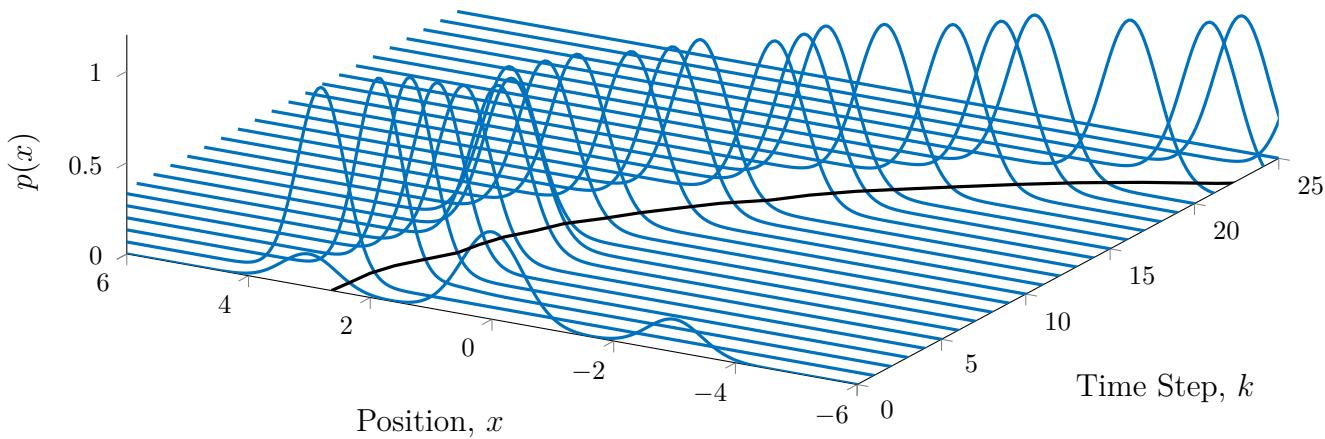
We can also generate a sample of the truth by sampling the initial GM pdf, and we can propagate this truth according to the true system dynamics. The first part of this process, sampling the initial GM pdf, is new to us. We are well-acquainted with sampling a Gaussian pdf, so we will try to leverage that knowledge. Sampling a GM pdf first requires selecting a component of the GM. The weights of the GM can be viewed as a probability mass function (pmf), and sampling a standard uniform random number can be used to generate the component by looking at where the random uniform number falls with respect to the cumulative distribution function of the pmf. Once a component is

selected, the sample from the GM pdf is generated by creating a Gaussian sample of that component. If we draw a sample of the initial pdf and propagate it according to the true system dynamics (including process noise), we get the result of the following figure.

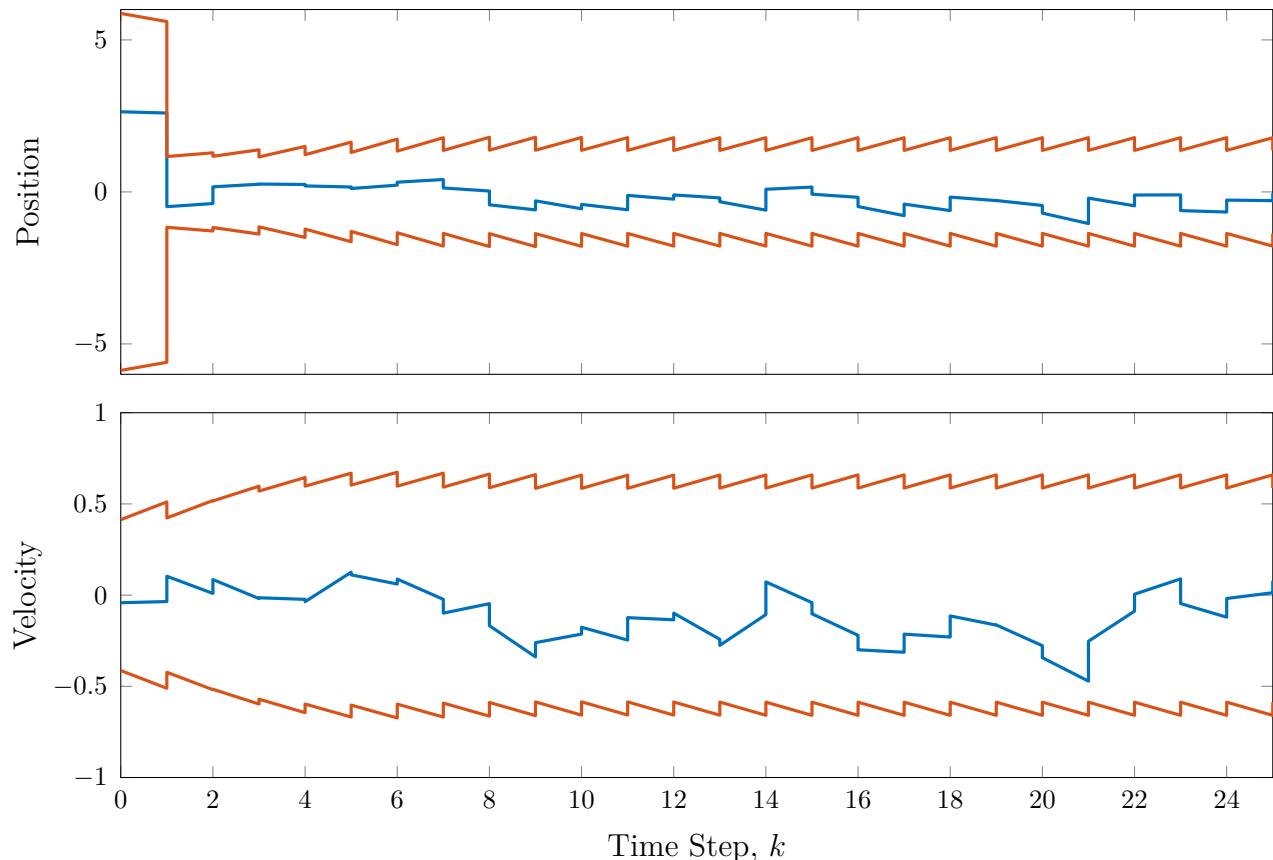


Given the sequence of true states, we can generate a set of true measurements for $1 \leq k \leq 25$ according to the true measurement model. This requires a mapping of the true states, and it also includes generation of random measurement noise, which is drawn from a Gaussian distribution. Given these measurements, the prior GM pdf, and all of

the system models, we implement a GMF. The posterior results of the GMF are shown as pdfs in the position state in the following figure.



Even though we have a GMF that *always* has three components, one component quickly dominates. The component that dominants is largely dictated by the true path of the state. The posterior pdf looks very much like a Gaussian pdf, which is due to the facts that: 1) we are processing Gaussian data and 2) the measurements are of the position state. We can also compute the conditional mean and covariance in order to compute and plot estimation errors and 3σ intervals; these are illustrated in the following figure.



We see that this filter does a good job of tracking the position of the object. The position-only measurements, however, are not precise enough to improve our initial understanding of the velocity of the object. Instead, the position-only measurement prevent the process noise from dominating the response over time.

There is an interesting theoretical perspective to add to this discussion. If we think about the linear-Gaussian system from a Kalman filtering perspective, the evolution of the covariance is deterministic. At no point do any random numbers enter into the determination of the covariance matrix for the Kalman filter (again, focusing on linear system models). For the GMF, on the other hand, the covariance matrix (found as the overall covariance of the GM) is not deterministic. The covariances of each of the GM components are deterministic, but the weights are stochastic. This means that when we compute the overall covariance, it is also stochastic. This is a very interesting difference between the Kalman filter and the Gaussian mixture filter.

6.5 Non-Gaussian Extension to the GMF

At this point, we have solutions to the Bayesian paradigm whenever the system is modeled by linear dynamics and measurements that are subjected to Gaussian process and measurement noises. If the initial pdf is a Gaussian, we get a Bayesian interpretation of the Kalman filter, and if the pdf is a GM, we get the Gaussian mixture filter. What happens whenever the process and measurement noises are non-Gaussian? In this case, we can “lift” our results for the GMF to the situation where these noises are modeled as GMs. Even though this produces a more general filter, we still refer to it as the GMF.

We need one result before we proceed, and that is a generalization of Ho’s equation that is given by

$$p_g(\mathbf{z}; \mathbf{Ax} + \mathbf{b}, \mathbf{C})p_g(\mathbf{x}; \mathbf{m}, \mathbf{P}) = p_g(\mathbf{z}; \mathbf{Am} + \mathbf{b}, \mathbf{A}\mathbf{P}\mathbf{A}^T + \mathbf{C})p_g(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Pi})$$

where

$$\boldsymbol{\mu} = \mathbf{m} + \mathbf{K}(\mathbf{z} - \mathbf{Am} - \mathbf{b})$$

$$\boldsymbol{\Pi} = \mathbf{P} - \mathbf{K}\mathbf{A}\mathbf{P}$$

$$\mathbf{K} = \mathbf{P} \mathbf{A}^T (\mathbf{A} \mathbf{P} \mathbf{A}^T + \mathbf{C})^{-1}$$

We call this the generalized Ho's equation, but the proof is omitted here. As with the original form of Ho's equation, you can also integrate over \mathbf{x} , which, in this case, yields

$$\int p_g(\mathbf{z}; \mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{C}) p_g(\mathbf{x}; \mathbf{m}, \mathbf{P}) d\mathbf{x} = p_g(\mathbf{z}; \mathbf{A}\mathbf{m} + \mathbf{b}, \mathbf{A} \mathbf{P} \mathbf{A}^T + \mathbf{C})$$

For either relationship, it is important to note that the parameter \mathbf{b} is taken to be deterministic.

Now, let's move on to developing a non-Gaussian extension to the GKF. As usual, we take the state-space models to be

$$\mathbf{x}_k = \mathbf{F}_{x,k-1} \mathbf{x}_{k-1} + \mathbf{w}_{k-1}$$

$$\mathbf{z}_k = \mathbf{H}_{x,k} \mathbf{x}_k + \mathbf{v}_k$$

which can be solved for the noises to yield

$$\mathbf{w}_{k-1} = \mathbf{x}_k - \mathbf{F}_{x,k-1} \mathbf{x}_{k-1}$$

$$\mathbf{v}_k = \mathbf{z}_k - \mathbf{H}_{x,k} \mathbf{x}_k$$

Now, however, we take the pdfs of the noises to be GMs, such that

$$p(\boldsymbol{w}_{k-1}) = \sum_{i=1}^{L_{w,k-1}} w_{w,k-1}^{(i)} p_g(\boldsymbol{w}_{k-1}; \boldsymbol{m}_{w,k-1}^{(i)}, \boldsymbol{P}_{ww,k-1}^{(i)})$$

$$p(\boldsymbol{v}_k) = \sum_{j=1}^{L_{v,k}} w_{v,k}^{(j)} p_g(\boldsymbol{v}_k; \boldsymbol{m}_{v,k}^{(j)}, \boldsymbol{P}_{vv,k}^{(j)})$$

Following the same process that we used with the Bayesian Kalman filter and with the Gaussian mixture filter, we can solve for the transition density and the measurement likelihood to give

$$p(\boldsymbol{x}_k | \boldsymbol{x}_{k-1}) = \sum_{i=1}^{L_{w,k-1}} w_{w,k-1}^{(i)} p_g(\boldsymbol{x}_k; \boldsymbol{F}_{x,k-1} \boldsymbol{x}_{k-1} + \boldsymbol{m}_{w,k-1}^{(i)}, \boldsymbol{P}_{ww,k-1}^{(i)})$$

$$p(\boldsymbol{z}_k | \boldsymbol{x}_k) = \sum_{j=1}^{L_{v,k}} w_{v,k}^{(j)} p_g(\boldsymbol{z}_k; \boldsymbol{H}_{x,k} \boldsymbol{x}_k + \boldsymbol{m}_{v,k}^{(j)}, \boldsymbol{P}_{vv,k}^{(j)})$$

where, unlike our previous approaches, we now have GM representations for the probabilistic state space model. Now, we will work on determining the form of the predicted density. As usual, our starting point is the Chapman-Kolmogorov equation, given by

$$p(\mathbf{x}_k \mid \mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k \mid \mathbf{x}_{k-1}) p(\mathbf{x}_{k-1} \mid \mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1}$$

In keeping with the spirit of the GMF, we take the posterior at $k - 1$ to be of the form

$$p(\mathbf{x}_{k-1} \mid \mathbf{z}_{1:k-1}) = \sum_{\ell=1}^{L_{x,k-1}^+} w_{x,k-1}^{(\ell)+} p_g(\mathbf{x}_{k-1}; \mathbf{m}_{x,k-1}^{(\ell)+}, \mathbf{P}_{xx,k-1}^{(\ell)+})$$

Substituting into the Chapman-Kolmogorov equation for the posterior density and the transition density yields

$$\begin{aligned} p(\mathbf{x}_k \mid \mathbf{z}_{1:k-1}) &= \int \left[\sum_{i=1}^{L_{w,k-1}} w_{w,k-1}^{(i)} p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1} \mathbf{x}_{k-1} + \mathbf{m}_{w,k-1}^{(i)}, \mathbf{P}_{ww,k-1}^{(i)}) \right] \\ &\quad \times \left[\sum_{\ell=1}^{L_{x,k-1}^+} w_{x,k-1}^{(\ell)+} p_g(\mathbf{x}_{k-1}; \mathbf{m}_{x,k-1}^{(\ell)+}, \mathbf{P}_{xx,k-1}^{(\ell)+}) \right] d\mathbf{x}_{k-1} \end{aligned}$$

which can be rearranged as

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \sum_{i=1}^{L_{w,k-1}} \sum_{\ell=1}^{L_{x,k-1}^+} w_{w,k-1}^{(i)} w_{x,k-1}^{(\ell)+} \int p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1} \mathbf{x}_{k-1} + \mathbf{m}_{w,k-1}^{(i)}, \mathbf{P}_{ww,k-1}^{(i)}) \\ \times p_g(\mathbf{x}_{k-1}; \mathbf{m}_{x,k-1}^{(\ell)+}, \mathbf{P}_{xx,k-1}^{(\ell)+}) d\mathbf{x}_{k-1}$$

Finally, we can apply the integral form of the generalized Ho's equation to find the predicted, or *a priori*, density as

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \sum_{i=1}^{L_{w,k-1}} \sum_{\ell=1}^{L_{x,k-1}^+} w_{w,k-1}^{(i)} w_{x,k-1}^{(\ell)+} \\ \times p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1} \mathbf{m}_{x,k-1}^{(\ell)+} + \mathbf{m}_{w,k-1}^{(i)}, \mathbf{F}_{x,k-1} \mathbf{P}_{xx,k-1}^{(\ell)+} \mathbf{F}_{x,k-1}^T + \mathbf{P}_{ww,k-1}^{(i)})$$

At first glance, this might not look like our standard form of a GM pdf, and that's because it isn't; nevertheless, this is a Gaussian mixture. When the process noise pdf is a GM with more than one component, the Chapman-Kolmogorov equation produces a predicted density that is a GM, but with an increased number of components. This

occurs because we have to take every combination of the posterior GM components with the process noise GM components. In fact, the predicted density can be written as

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(\ell)-} p_g(\mathbf{x}_k | \mathbf{m}_{x,k}^{(\ell)-}, \mathbf{P}_{x,k}^{(\ell)-})$$

where $L_{x,k}^- = L_{x,k-1}^+ \cdot L_{w,k-1}$ is the number of propagated components, by making an appropriate selection of the weights, means, and covariances.

Now, we consider the update step, which starts from Bayes' rule as

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) \propto p(\mathbf{z}_k | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{z}_{1:k-1})$$

where proportionality becomes equality when we normalize the result to ensure a valid posterior pdf. As such, we focus on the numerator of Bayes' rule and substitute for the measurement likelihood and the prior pdf, both as GMs, to yield

$$p(\mathbf{z}_k | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \left[\sum_{j=1}^{L_{v,k}} w_{v,k}^{(j)} p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{x}_k + \mathbf{m}_{v,k}^{(j)}, \mathbf{P}_{vv,k}^{(j)}) \right]$$

$$\times \left[\sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(\ell)-} p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(\ell)-}, \mathbf{P}_{xx,k}^{(\ell)-}) \right]$$

Rearranging terms allows us to write the numerator of Bayes' rule as

$$p(\mathbf{z}_k | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) \\ = \sum_{j=1}^{L_{v,k}} \sum_{\ell=1}^{L_{x,k}^-} w_{v,k}^{(j)} w_{x,k}^{(\ell)-} p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{x}_k + \mathbf{m}_{v,k}^{(j)}, \mathbf{P}_{vv,k}^{(j)}) p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(\ell)-}, \mathbf{P}_{xx,k}^{(\ell)-})$$

The summand matches up with the form of the generalized Ho's equation, such that

$$p(\mathbf{z}_k | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \sum_{j=1}^{L_{v,k}} \sum_{\ell=1}^{L_{x,k}^-} w_{v,k}^{(j)} w_{x,k}^{(\ell)-} \\ \times p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-} + \mathbf{m}_{v,k}^{(j)}, \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k}^{(j)}) p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(j,\ell)+}, \mathbf{P}_{xx,k}^{(j,\ell)+})$$

where

$$\mathbf{m}_{x,k}^{(j,\ell)+} = \mathbf{m}_{x,k}^{(\ell)-} + \mathbf{K}_k^{(j,\ell)} (\mathbf{z}_k - \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-} - \mathbf{m}_{v,k}^{(j)})$$

$$\mathbf{P}_{xx,k}^{(j,\ell)+} = \mathbf{P}_{xx,k}^{(\ell)-} - \mathbf{K}_k^{(j,\ell)} \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-}$$

$$\mathbf{K}_k^{(j,\ell)} = \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T (\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k}^{(j)})^{-1}$$

The denominator of Bayes' rule (alternatively viewed as the normalization factor) can be determined by integrating the numerator of Bayes' rule, which produces

$$\int p(\mathbf{z}_k | \boldsymbol{\xi}_k) p(\boldsymbol{\xi}_k | \mathbf{z}_{1:k-1}) d\boldsymbol{\xi}_k$$

$$= \sum_{j=1}^{L_{v,k}} \sum_{\ell=1}^{L_{x,k}^-} w_{v,k}^{(j)} w_{x,k}^{(\ell)-} p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-} + \mathbf{m}_{v,k}^{(j)}, \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k}^{(j)})$$

Now, we just need to put together the numerator and denominator to complete our development of Bayes' rule and arrive at the final result that we need for the filtering, or *a posteriori*, density. First, define the gain

$$k_k^{(j,\ell)} = p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-} + \mathbf{m}_{v,k}^{(j)}, \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k}^{(j)})$$

which, as a reminder, is not actually a function when we process a measurement. Whenever we acquire a measurement and process it, this evaluates as a number. Then, from

our preceding developments, the numerator and denominator are, respectively, given by

$$p(\boldsymbol{z}_k \mid \boldsymbol{x}_k) p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1}) = \sum_{j=1}^{L_{v,k}} \sum_{\ell=1}^{L_{x,k}^-} w_{v,k}^{(j)} w_{x,k}^{(\ell)-} k_k^{(j,\ell)} p_g(\boldsymbol{x}_k; \boldsymbol{m}_{x,k}^{(j,\ell)+}, \boldsymbol{P}_{xx,k}^{(j,\ell)+})$$

$$\int p(\boldsymbol{z}_k \mid \boldsymbol{\xi}_k) p(\boldsymbol{\xi}_k \mid \boldsymbol{z}_{1:k-1}) d\boldsymbol{\xi}_k = \sum_{j=1}^{L_{v,k}} \sum_{\ell=1}^{L_{x,k}^-} w_{v,k}^{(j)} w_{x,k}^{(\ell)-} k_k^{(j,\ell)}$$

which, when put together, yield the filtering distribution as

$$p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k}) = \sum_{j=1}^{L_{v,k}} \sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(j,\ell)+} p_g(\boldsymbol{x}_k; \boldsymbol{m}_{x,k}^{(j,\ell)+}, \boldsymbol{P}_{xx,k}^{(j,\ell)+})$$

where

$$w_{x,k}^{(j,\ell)+} = \frac{w_{v,k}^{(j)} w_{x,k}^{(\ell)-} k_k^{(j,\ell)}}{\sum_{j'=1}^{L_{v,k}} \sum_{\ell'=1}^{L_{x,k}^-} w_{v,k}^{(j')} w_{x,k}^{(\ell')-} k_k^{(j',\ell')}}$$

As with the predicted distribution, the posterior distribution can be expressed as a single sum in the form

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) = \sum_{\ell=1}^{L_{x,k}^+} w_{x,k}^{(\ell)+} p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(\ell)+}, \mathbf{P}_{xx,k}^{(\ell)+})$$

where $L_{x,k}^+ = L_{x,k}^- \cdot L_{v,k}$ is the number of posterior components, which is greater than the number of prior components whenever $L_{v,k} > 1$. The increase in the number of components is, again, due to consideration of every combination of prior and measurement noise components.

We have therefore illustrated that the application of a linear, Gaussian mixture model for the process noise and the measurement noise leads to Gaussian mixture predicted and filtering densities. Therefore, if the initial distribution is a Gaussian mixture, the density will remain a Gaussian mixture, and the recursion closes; however, unlike the original form of the GMF, the number of components grows with every propagation and every update unless the noise distributions are taken to be Gaussian. It is also worth noting that this GMF recovers our original GMF whenever the process and measurement noises are zero-mean Gaussian pdfs.

To summarize:

- Given the state-space model to be a linear system of the form

$$\mathbf{x}_k = \mathbf{F}_{x,k-1} \mathbf{x}_{k-1} + \mathbf{w}_{k-1}$$

$$\mathbf{z}_k = \mathbf{H}_{x,k} \mathbf{x}_k + \mathbf{v}_k$$

where the process and measurement noises are Gaussian mixture pdfs, transform the state-space model to the probabilistic model

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}) = \sum_{i=1}^{L_{w,k-1}} w_{w,k-1}^{(i)} p_g(\mathbf{x}_k; \mathbf{F}_{x,k-1} \mathbf{x}_{k-1} + \mathbf{m}_{w,k-1}^{(i)}, \mathbf{P}_{ww,k-1}^{(i)})$$

$$p(\mathbf{z}_k | \mathbf{x}_k) = \sum_{j=1}^{L_{v,k}} w_{v,k}^{(j)} p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{x}_k + \mathbf{m}_{v,k}^{(j)}, \mathbf{P}_{vv,k}^{(j)})$$

- Initialize a Gaussian mixture pdf with weights, means, and covariances

$$L_{x,k-1}^+ = L_{x,0}, \quad w_{x,k-1}^{(\ell)+} = w_{x,0}^{(\ell)}, \quad \mathbf{m}_{x,k-1}^{(\ell)+} = \mathbf{m}_{x,0}^{(\ell)}, \quad \mathbf{P}_{xx,k-1}^{(\ell)+} = \mathbf{P}_{xx,0}^{(\ell)}$$

- Compute the predicted pdf using the Chapman-Kolmogorov equation

$$p(\mathbf{x}_k \mid \mathbf{z}_{1:k-1}) = \sum_{i=1}^{L_{w,k-1}} \sum_{\ell=1}^{L_{x,k-1}^+} w_{x,k}^{(i,\ell)-} p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(i,\ell)-}, \mathbf{P}_{xx,k}^{(i,\ell)-})$$

where $L_{x,k}^- = L_{w,k-1} \cdot L_{x,k-1}^+$ and

$$w_{x,k}^{(i,\ell)-} = w_{w,k-1}^{(i)} w_{x,k-1}^{(\ell)+}$$

$$\mathbf{m}_{x,k}^{(i,\ell)-} = \mathbf{F}_{x,k-1} \mathbf{m}_{x,k-1}^{(\ell)+} + \mathbf{m}_{w,k-1}^{(i)}$$

$$\mathbf{P}_{xx,k}^{(i,\ell)-} = \mathbf{F}_{x,k-1} \mathbf{P}_{xx,k-1}^{(\ell)+} \mathbf{F}_{x,k-1}^T + \mathbf{P}_{ww,k-1}^{(i)}$$

and form the associated $L_{x,k}^-$ -component GM.

- Compute the filtered pdf using Bayes' Rule

$$p(\mathbf{x}_k \mid \mathbf{z}_{1:k}) = \sum_{j=1}^{L_{v,k}} \sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(j,\ell)+} p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(j,\ell)+}, \mathbf{P}_{xx,k}^{(j,\ell)+})$$

where $L_{x,k}^+ = L_{v,k} \cdot L_{x,k}^-$,

$$w_{x,k}^{(j,\ell)+} = w_{v,k}^{(j)} w_{x,k}^{(\ell)-} k_k^{(j,\ell)} / \sum_{j'=1}^{L_{v,k}} \sum_{\ell'=1}^{L_{x,k}^-} w_{v,k}^{(j')} w_{x,k}^{(\ell')-} k_k^{(j',\ell')}$$

$$\mathbf{m}_{x,k}^{(j,\ell)+} = \mathbf{m}_{x,k}^{(\ell)-} + \mathbf{K}_k^{(j,\ell)} (\mathbf{z}_k - \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-} - \mathbf{m}_{v,k}^{(j)})$$

$$\mathbf{P}_{xx,k}^{(j,\ell)+} = \mathbf{P}_{xx,k}^{(\ell)-} - \mathbf{K}_k^{(j,\ell)} \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-}$$

$$k_k^{(j,\ell)} = p_g(\mathbf{z}_k; \mathbf{H}_{x,k} \mathbf{m}_{x,k}^{(\ell)-} + \mathbf{m}_{v,k}^{(j)}, \mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k}^{(j)})$$

$$\mathbf{K}_k^{(j,\ell)} = \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T (\mathbf{H}_{x,k} \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_{x,k}^T + \mathbf{P}_{vv,k}^{(j)})^{-1}$$

then form the associated $L_{x,k}^+$ -component GM.

6.5.1 Another Example of the GMF

Let's look at another example of a GMF, but this time with non-Gaussian noise. To keep things a bit easier to visualize, we will consider the scalar system

$$x_k = F_{x,k-1} x_{k-1} + w_{k-1}$$

$$z_k = H_{x,k}x_k + v_k$$

where $F_{x,k-1} = 1$ and $H_{x,k} = 1$. We will take the pdf of the initial state to be a three-component GM with weights, means, and (co)variances given by

$$w_{x,0}^{(1)} = 0.4 \quad m_{x,0}^{(1)} = -1.0 \quad P_{xx,0}^{(1)} = 0.2$$

$$w_{x,0}^{(2)} = 0.4 \quad m_{x,0}^{(2)} = 0.0 \quad P_{xx,0}^{(2)} = 0.5$$

$$w_{x,0}^{(3)} = 0.2 \quad m_{x,0}^{(3)} = 1.0 \quad P_{xx,0}^{(3)} = 0.3$$

The process noise pdf is taken to be a one-component GM (otherwise known as a Gaussian) with weight, mean, and (co)variance

$$w_{w,k-1}^{(1)} = 1.0 \quad m_{w,k-1}^{(1)} = 0.0 \quad P_{ww,k-1}^{(1)} = 0.01$$

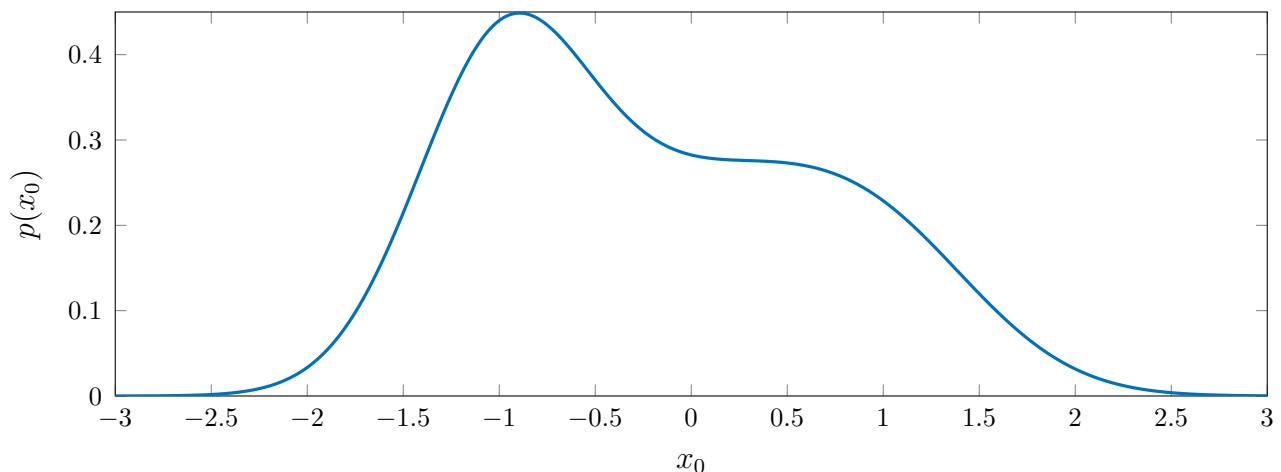
Finally, the measurement noise pdf is taken to be a three-component GM with weights, means, and (co)variances of

$$w_{v,k}^{(1)} = 0.25 \quad m_{v,k}^{(1)} = -0.9 \quad P_{vv,k}^{(1)} = 0.5$$

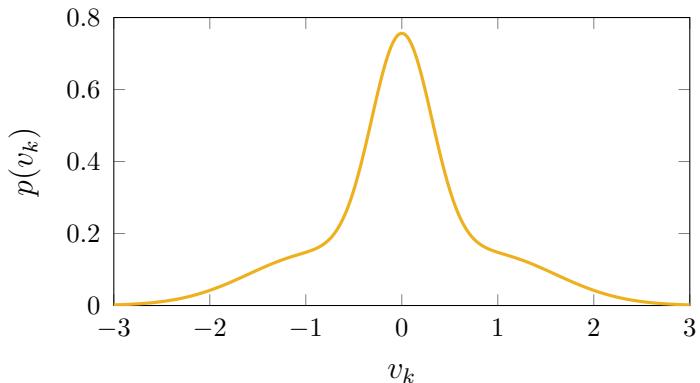
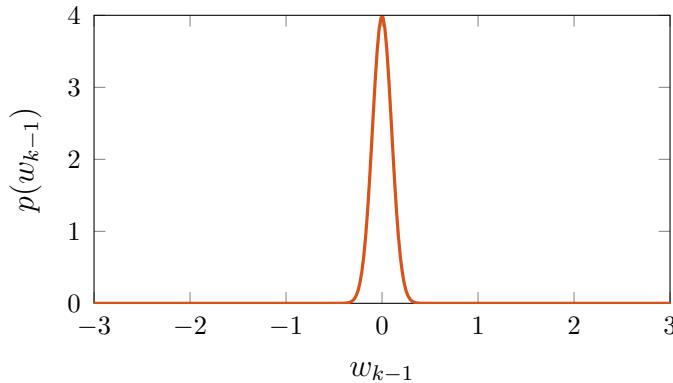
$$w_{v,k}^{(2)} = 0.5 \quad m_{v,k}^{(2)} = 0.0 \quad P_{vv,k}^{(2)} = 0.1$$

$$w_{v,k}^{(3)} = 0.25 \quad m_{v,k}^{(3)} = 0.9 \quad P_{vv,k}^{(3)} = 0.5$$

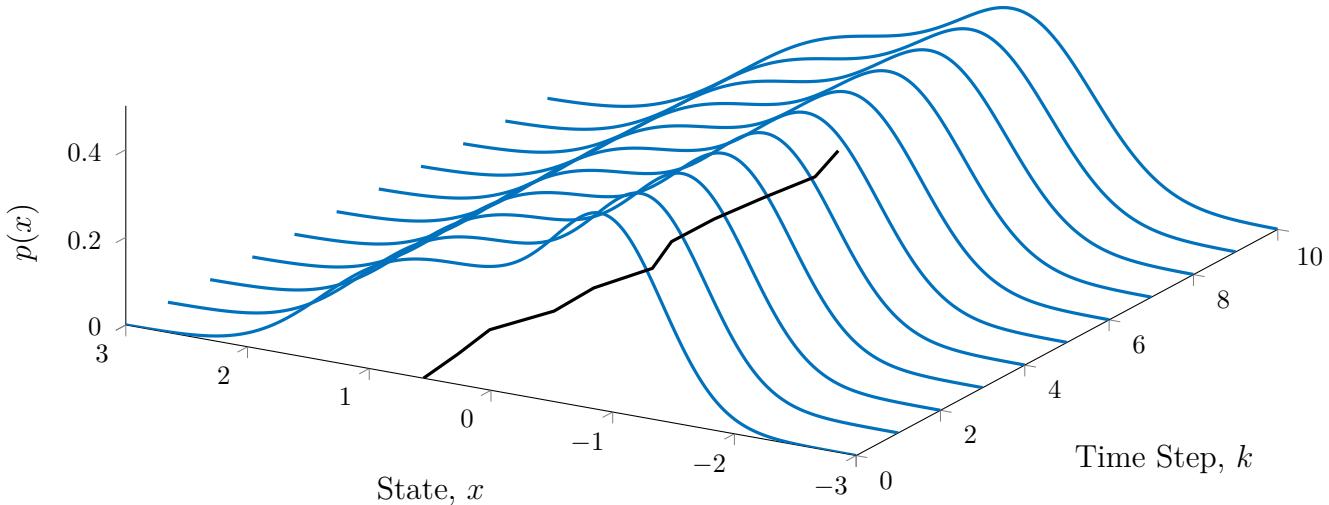
To gain a bit of insight into the system that we're working with, the pdf of the initial state is visualized as



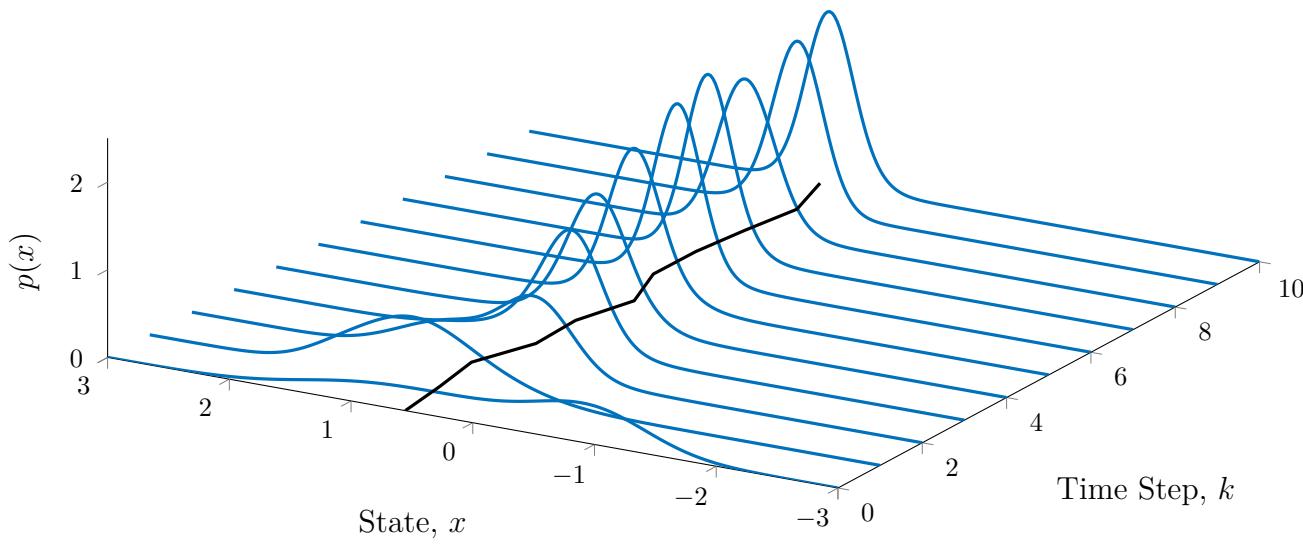
and the process noise and measurement noise pdfs are illustrated in the next figures.



We now have everything that we need in order to be able to implement a GMF. Before implementing the full GMF, however, let's investigate what happens if we only propagate uncertainty for a series of 10 time steps. The corresponding results are provided in the next figure, where it is evident that the process noise is not very dominant, such that it leaves the overall shape of the pdf relatively unchanged. Additionally, it is worth noting that, since we have a Gaussian process noise, this pdf is, at every time step, a GM with only three components.

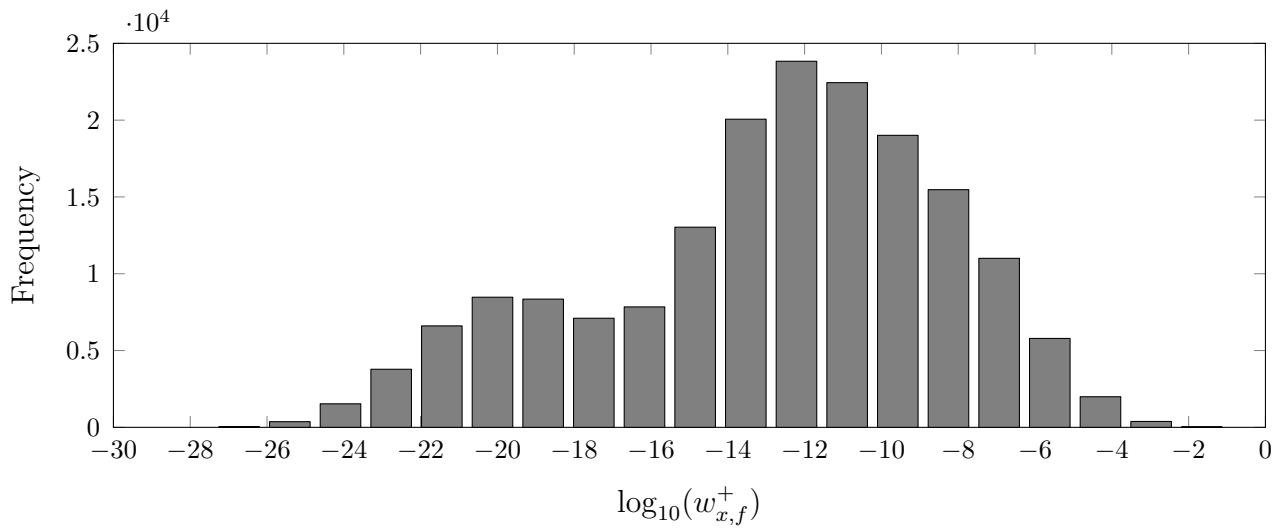


With a better understanding for the uncertainty propagation problem, let's consider the estimation problem. We will generate an initial true state, x_0 , that will be propagated according to the system dynamics. At each time step, we will generate data using the true state of the system. This data is processed using the GMF, and each time we process a measurement, we get a three-fold increase in the number of components that describe our pdf. The posterior pdf at each time step is shown in following figure.



As expected, we see a large growth in the number of components as time moves along and we keep processing measurements. In fact, $L_{x,k}^+ = 3 \cdot 3^{10} = 177,147$ at $k = 10$. Despite this large number of components, the posterior looks relatively Gaussian, even as early as $k = 4$. This makes us wonder if all of the components of the GM are really all that important. We can assess the importance of the components by looking at the weights of the components. A histogram of the weights, which is given in the next figure, indicates

that there is a significant number of components with extremely small weights.



This means that a lot of the components can be removed, but it should be noted that the filter then becomes approximate and that we have to choose a threshold for component removal.

We might also wonder how a Kalman filter (in the MMSE sense) would perform on

this problem relative to the GMF. First, we need to do a little work to translate our problem configuration into one that can be tackled by the KF. To make the translation, we need to compute the first two moments of the initial state pdf, the process noise pdf, and the measurement noise pdf. From our previous discussion on the moments of a GM pdf, it follows that

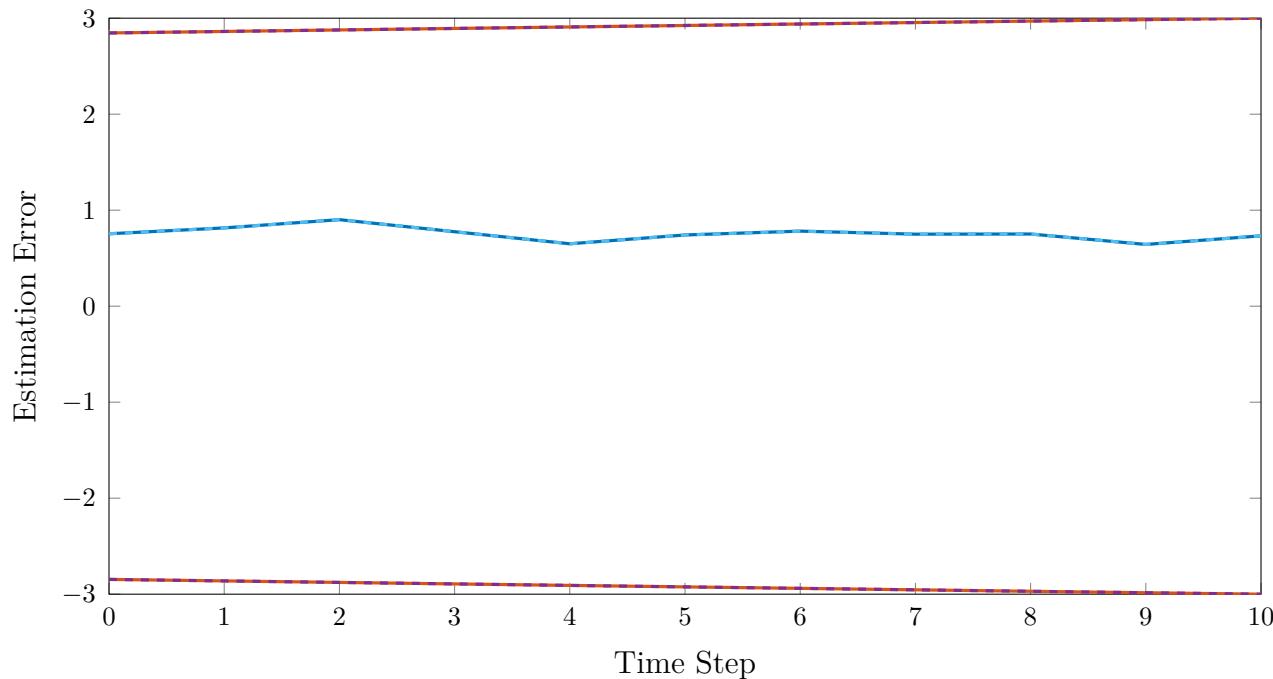
$$m_{x,0} = \sum_{\ell=1}^{L_{x,0}} w_{x,0}^{(\ell)} m_{x,0}^{(\ell)} = -0.2 \quad \text{and} \quad P_{xx,0} = \sum_{\ell=1}^{L_{x,0}} w_{x,0}^{(\ell)} (P_{x,0}^{(\ell)} + (m_{x,0}^{(\ell)} - m_{x,0})^2) = 0.9$$

As the process noise is already Gaussian, it is straightforward to see that it is zero mean, with (co)variance $P_{ww,k-1} = 0.01$. Finally, the measurement noise pdf is symmetric, so it has a mean of zero, and the (co)variance is

$$P_{vv,k} = \sum_{j=1}^{L_{v,k}} w_{v,k}^{(j)} (P_{vv,k}^{(j)} + (m_{v,k}^{(j)})^2) = 0.705$$

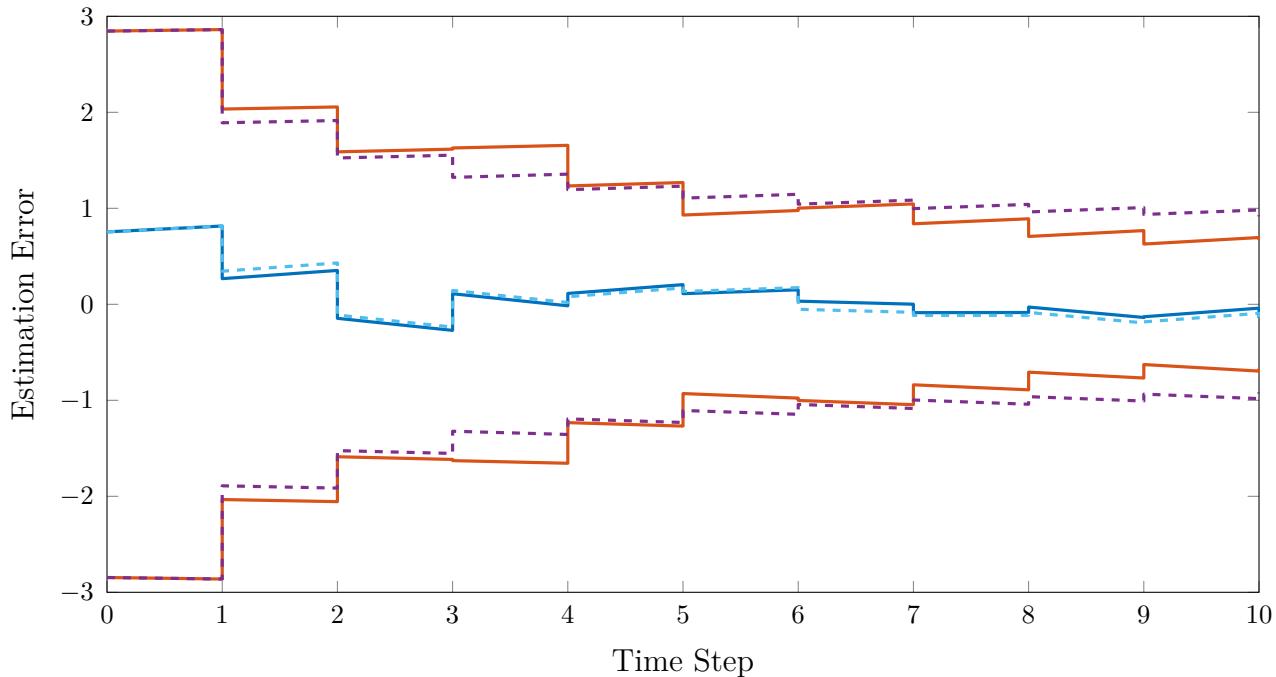
This is everything that we need to set up the Kalman filter. The first thing that we'll do is just run a propagation of the uncertainty using both the GMF and the KF. We know that the GMF is going to yield a non-Gaussian pdf, so for comparison, we'll compute the

mean and covariance from the GM and compare that to the mean and covariance of the Kalman filter. We do this by computing and plotting estimation errors and 3σ intervals for both filters, which are illustrated in the next figure.



They're identical! Propagating a GM pdf using Chapman-Kolmogorov and computing the mean and covariance yields the same result as the Kalman filter. This result is provable, so it's not simply a coincidence that it occurs in this problem. It does rely on the process noise being zero-mean, however. That limitation can be resolved by equipping the Kalman filter with a non-zero-mean process noise, as well.

Now, let's take a look at what happens when we process data in both of these filters. To make a fair comparison, it is critical that the exact same data is processed by both the GMF and the KF. We take the GM representation of the system (i.e. the GM representations of the initial pdf and the noise pdfs) to be the true description of the system. This information is used to generate the state history and the measurement history. Keep in mind that, while the propagation stages of each filter yielded the same result, the Kalman filter is a linear filter and the GMF is a nonlinear filter (thanks to Bayes' rule). Once again, we'll compute the GM pdf using the GMF, then determine the mean and covariance to compare against the KF's mean and covariance (prior and posterior). The estimation errors and corresponding 3σ intervals are shown in the following figure, where the GMF is shown in solid lines, and the KF is shown in dashed lines.



We clearly observe different behaviors from the two filters. This is the effect of linear vs nonlinear filters. In this case, the GMF obtains a more precise estimate of the truth, even though both estimates are very similar to one another. One thing to keep in mind is that

the 3σ interval of the GMF is stochastic, while the 3σ interval of the KF is deterministic. This means that a different measurement history might lead to the KF achieving a more precise estimate than the GMF.

There is something else interesting to consider here. What happens to the 3σ value of the GMF at the update for $k = 3$ and for $k = 6$? At both updates, the uncertainty grows a little bit. Is this allowable?

6.6 Nonlinear Extensions to the GMF

As with the Kalman filter, we can also handle nonlinear systems by any of a number of methods:

- linearization – EGMF
- unscented transform – UGMF
- quadrature – GHGMF
- cubature – CGMF

The additive, nonlinear state-space models are given by

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k$$

We can still make the assumption that the process and measurement noises are Gaussian, which leads us to the probabilistic models

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}) = p_g(\mathbf{x}_k; \mathbf{f}(\mathbf{x}_{k-1}), \mathbf{P}_{ww,k-1})$$

$$p(\mathbf{z}_k | \mathbf{x}_k) = p_g(\mathbf{z}_k; \mathbf{h}(\mathbf{x}_k), \mathbf{P}_{vv,k})$$

It is also possible to take the process and measurement noises to be represented by GMs in much the same way as we did previously, and this is, as we have seen, a pretty straightforward extension.

Our objective is to use the Bayesian paradigm of propagating the pdf via the Chapman-Kolmogorov equation and updating the pdf via Bayes' rule. The difference is that we now need to account for the appearance of the nonlinear systems in the application of our filtering equations.

Unfortunately, the form of our probabilistic state space model no longer conforms with Ho's equation anymore. As such, we need to figure out a way to overcome this challenge. First, let's apply a first-order Taylor series expansion to both of our nonlinear functions. For the transition density, we will expand about the previous posterior, and for the measurement likelihood, we will expand about the current prior. This process yields

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}) = p_g(\mathbf{x}_k; \mathbf{f}(\mathbf{m}_{x,k-1}^+) + \mathbf{F}_x(\mathbf{m}_{x,k-1}^+) [\mathbf{x}_{k-1} - \mathbf{m}_{x,k-1}^+], \mathbf{P}_{ww,k-1})$$

$$p(\mathbf{z}_k | \mathbf{x}_k) = p_g(\mathbf{z}_k; \mathbf{h}(\mathbf{m}_{x,k}^-) + \mathbf{H}_x(\mathbf{m}_{x,k}^-) [\mathbf{x}_k - \mathbf{m}_k^-], \mathbf{P}_{vv,k})$$

Now, let's rearrange terms to bring out the state-dependence by itself, giving

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}) = p_g(\mathbf{x}_k; \mathbf{F}_x(\mathbf{m}_{x,k-1}^+) \mathbf{x}_{k-1} + [\mathbf{f}(\mathbf{m}_{x,k-1}^+) - \mathbf{F}_x(\mathbf{m}_{x,k-1}^+) \mathbf{m}_{x,k-1}^+], \mathbf{P}_{ww,k-1})$$

$$p(\mathbf{z}_k | \mathbf{x}_k) = p_g(\mathbf{z}_k; \mathbf{H}_x(\mathbf{m}_{x,k}^-) \mathbf{x}_k + [\mathbf{h}(\mathbf{m}_{x,k}^-) - \mathbf{H}_x(\mathbf{m}_{x,k}^-) \mathbf{m}_{x,k}^-], \mathbf{P}_{vv,k})$$

Looking at both of these conditional pdfs, it is clear that they are both of the form

$$p_g(\mathbf{z}; \mathbf{Ax} + \mathbf{b}, \mathbf{C})$$

where the term \mathbf{b} is given by

$$\mathbf{b} = \mathbf{f}(\mathbf{m}_{x,k-1}^+) - \mathbf{F}_x(\mathbf{m}_{x,k-1}^+) \mathbf{m}_{x,k-1}^+$$

$$\mathbf{b} = \mathbf{h}(\mathbf{m}_{x,k}^-) - \mathbf{H}_x(\mathbf{m}_{x,k}^-) \mathbf{m}_{x,k}^-$$

for the transition density and the measurement likelihood, respectively. We can express both of these terms in a general expression of the form

$$\mathbf{b} = \mathbf{g}(\mathbf{m}) - \mathbf{A}(\mathbf{m})\mathbf{m}$$

where \mathbf{g} , \mathbf{A} , and \mathbf{m} are adjusted as needed for each of the densities. Provided that we make the usual assumptions about deterministic quantities, we can apply the generalized Ho's equation from before. From the general form of \mathbf{b} , it follows that

$$\mathbf{b} + \mathbf{A}(\mathbf{m})\mathbf{m} = \mathbf{g}(\mathbf{m})$$

Therefore, making the preceding substitution in the application of the generalized Ho's equation, we arrive at

$$p_g(\mathbf{z}; \mathbf{A}(\mathbf{m})\mathbf{x} + \mathbf{b}, \mathbf{C}) p_g(\mathbf{x}; \mathbf{m}, \mathbf{P}) = p_g(\mathbf{z}; \mathbf{g}(\mathbf{m}), \mathbf{A}(\mathbf{m})\mathbf{P}\mathbf{A}^T(\mathbf{m}) + \mathbf{C}) p_g(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Pi})$$

where

$$\boldsymbol{\mu} = \mathbf{m} + \mathbf{K}(\mathbf{z} - \mathbf{g}(\mathbf{m}))$$

$$\boldsymbol{\Pi} = \mathbf{P} - \mathbf{K}\mathbf{A}(\mathbf{m})\mathbf{P}$$

$$\mathbf{K} = \mathbf{P}\mathbf{A}^T(\mathbf{m})(\mathbf{A}(\mathbf{m})\mathbf{P}\mathbf{A}^T(\mathbf{m}) + \mathbf{C})^{-1}$$

You should quite clearly see equations familiar from the EKF emerging from the Bayesian framework now. However, it is important to recognize that this is a fundamentally different approach than how we developed the EKF, and the similarity should not imply equivalence.

Of course, when we're dealing with nonlinear systems, it cannot be guaranteed that a transformed Gaussian is Gaussian. This is embedded in our approximation of the nonlinear dynamical system through the first-order Taylor series.

At this point, we have all of the tools that we need to derive an extended Gaussian mixture extended filter (EGMF). We won't go through the process explicitly, but the steps directly mirror those of the GMKF:

1. Assume that the posterior at time t_{k-1} is described by an $L_{x,k-1}^+$ -component GM.
2. Using the Chapman-Kolmogorov equation in conjunction with the transition density, a first-order Taylor series expansion, and the integral form of the generalized Ho's equation, solve for the $L_{x,k}^-$ -component prior GM at time t_k .
3. Using Bayes' rule in conjunction with the measurement likelihood, a first-order Taylor series expansion, the generalized Ho's equation, and the integral form of the generalized Ho's equation, solve for the $L_{x,k}^+$ -component posterior GM at time t_k .

Note: Whereas the GMF is closed (a GM rigorously stays a GM), the EGMF cannot be closed due to the nonlinear functions; it is only an approximate closure of the GM recursion. This does not stop us from applying the GMEKF in practice, of course, but it is something to keep in mind.

It is also important to note that if the process noise and measurement noise are taken to be Gaussian, there is no growth in component number, i.e. $L_{x,k}^+ = L_{x,k}^- = L_{x,k-1}^+$. This is not the case if we model the noises as being mixtures as well. The component growth in these situations directly mirrors that of the GMF that we have previously discussed.

One of the key uses for GM representations in nonlinear systems, besides modeling non-Gaussian pdfs, is to reduce linearization error. In general, as the number of components in a mixture is increased and the covariance of each component is decreased, the errors incurred by the linearization process can be mitigated such that the GM recursion more accurately describes the true pdf. Thus, a trade-off emerges between computational complexity and solution accuracy.

The EGMF for Gaussian noise is summarized as follows:

- Given the nonlinear system as the state-space model

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k$$

make the assumption that the noises are Gaussian and transform the model to the probabilistic model

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}) = p_g(\mathbf{x}_k; \mathbf{f}(\mathbf{x}_{k-1}), \mathbf{P}_{ww,k-1})$$

$$p(\mathbf{z}_k | \mathbf{x}_k) = p_g(\mathbf{z}_k; \mathbf{h}(\mathbf{x}_k), \mathbf{P}_{vv,k})$$

- Initialize a Gaussian mixture pdf with weights, means, and covariances

$$L_{x,k-1}^+ = L_{x,0}, \quad w_{x,k-1}^{(\ell)+} = w_{x,0}^{(\ell)}, \quad \mathbf{m}_{x,k-1}^{(\ell)+} = \mathbf{m}_{x,0}^{(\ell)}, \quad \mathbf{P}_{xx,k-1}^{(\ell)+} = \mathbf{P}_{xx,0}^{(\ell)}$$

- Compute the predicted pdf using the Chapman-Kolmogorov equation

$$p(\mathbf{x}_k | \mathbf{z}_{1:k-1}) = \sum_{\ell=1}^{L_{x,k}^-} w_{x,k}^{(\ell)-} p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(\ell)-}, \mathbf{P}_{xx,k}^{(\ell)-})$$

where

$$L_{x,k}^- = L_{x,k-1}^+$$

$$w_{x,k}^{(\ell)-} = w_{x,k-1}^{(\ell)+}$$

$$\mathbf{m}_{x,k}^{(\ell)-} = \mathbf{f}(\mathbf{m}_{x,k-1}^{(\ell)+})$$

$$\mathbf{P}_{xx,k}^{(\ell)-} = \mathbf{F}_x(\mathbf{m}_{x,k-1}^{(\ell)+}) \mathbf{P}_{xx,k-1}^{(\ell)+} \mathbf{F}_x^T(\mathbf{m}_{x,k-1}^{(\ell)+}) + \mathbf{P}_{ww,k-1}$$

- Compute the filtered pdf using Bayes' Rule

$$p(\mathbf{x}_k | \mathbf{z}_{1:k}) = \sum_{\ell=1}^{L_{x,k}^+} w_{x,k}^{(\ell)+} p_g(\mathbf{x}_k; \mathbf{m}_{x,k}^{(\ell)+}, \mathbf{P}_{xx,k}^{(\ell)+})$$

where the updated parameters of the GM are

$$L_{x,k}^+ = L_{x,k}^-$$

$$w_{x,k}^{(\ell)+} = k_k^{(\ell)} w_{x,k}^{(\ell)-} / \sum_{i=1}^{L_{x,k}^-} k_k^{(i)} w_k^{(i)-}$$

$$\mathbf{m}_{x,k}^{(\ell)+} = \mathbf{m}_{x,k}^{(\ell)-} + \mathbf{K}_k^{(\ell)} (\mathbf{z}_k - \mathbf{m}_{z,k}^{(\ell)-})$$

$$\mathbf{P}_{xx,k}^{(\ell)+} = \mathbf{P}_{xx,k}^{(\ell)-} - \mathbf{P}_{xz,k}^{(\ell)-} (\mathbf{K}_k^{(\ell)})^T - \mathbf{K}_k^{(\ell)} (\mathbf{P}_{xz,k}^{(\ell)-})^T + \mathbf{K}_k^{(\ell)} \mathbf{P}_{zz,k}^{(\ell)-} (\mathbf{K}_k^{(\ell)})^T$$

the expected values are

$$\mathbf{m}_{z,k}^{(\ell)-} = \mathbf{h}(\mathbf{m}_{x,k}^{(\ell)-})$$

$$\mathbf{P}_{xz,k}^{(\ell)-} = \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_x^T (\mathbf{m}_{x,k}^{(\ell)-})$$

$$\mathbf{P}_{zz,k}^{(\ell)-} = \mathbf{H}_x(\mathbf{m}_{x,k}^{(\ell)-}) \mathbf{P}_{xx,k}^{(\ell)-} \mathbf{H}_x^T(\mathbf{m}_{x,k}^{(\ell)-}) + \mathbf{P}_{vv,k}$$

and the gains are

$$\begin{aligned}\mathbf{K}_k^{(\ell)} &= \mathbf{P}_{xz,k}^{(\ell)-} (\mathbf{P}_{zz,k}^{(\ell)-})^{-1} \\ k_k^{(\ell)} &= p_g(\mathbf{z}_k; \mathbf{m}_{z,k}^{(\ell)-}, \mathbf{P}_{zz,k}^{(\ell)-})\end{aligned}$$

It is worth noting that, in the preceding algorithm for the EGMF, we have used the generalized Joseph formula for the covariance update. Even though this form does not directly appear through the developments from Ho's equation, we know that these covariance formulations are algebraically equivalent when we apply the linear gain that arises in Ho's equation (which is the Kalman gain in the Kalman framework). Because of the generally superior behavior of the variants of the Joseph formula, this alteration is leveraged in the preceding algorithm.

It is quite straightforward to extend this EGMF to non-Gaussian noise sequences. A similar procedure that we previously employed for the GMF can be used, and an approach similar to the one just listed out for the EGMF will be the result.

It is also possible to employ the methods we have already discussed for computing expectations to build a(n)

- unscented Gaussian mixture filter (UGMF)
- Gauss-Hermite Gaussian mixture filter (GHGMF)
- cubature Gaussian mixture filter (CGMF)

One can even assemble square root implementations to build a

- square root Gaussian mixture filter (SRGMF)
- square root extended Gaussian mixture filter (SREGMF)
- square root unscented Gaussian mixture filter (SRUGMF)
- square root Gauss-Hermite Gaussian mixture filter (SRGHGMF)
- square root cubature Gaussian mixture filter (SRCGMF)

It all boils down to how you compute expected values and how you represent covariance matrices.

6.6.1 Example of the EGMF

Let's look at an example of the EGMF for an orbit determination and object characterization problem. We're going to consider the problem of estimating the position, velocity, and a solar radiation pressure coefficient of a cannonball object using line-of-sight (right-ascension and declination) measurements.

The dynamics of the problem are

$$\dot{\mathbf{r}} = \mathbf{v}$$

$$\dot{\mathbf{v}} = -\frac{\mu}{\|\mathbf{r}\|^3} \mathbf{r} - \frac{C_R A}{m} \frac{\Phi}{c} \frac{\mathbf{AU}}{\|\mathbf{r} - \mathbf{r}_{\text{sun}}\|^2} (\mathbf{r} - \mathbf{r}_{\text{sun}})$$

$$\frac{d}{dt} \left\{ \frac{C_R A}{m} \right\} = 0$$

where

- $\mu = 3.986004418 \times 10^5 \text{ km}^3/\text{s}^2$ – gravitational parameter of Earth
- $\Phi = 1367.0 \text{ W/m}^2$ – solar constant

- $c = 299792458.0$ m/s – the speed of light
- $\text{AU} = 149597870.7$ km – the astronomical unit
- C_R – coefficient of reflectivity
- A/m – area-to-mass ratio (AMR)
- \mathbf{r}_{sun} – position of the Sun in the inertial frame
- \mathbf{r} – position of the space object in the inertial frame
- \mathbf{v} – velocity of the space object in the inertial frame

Our states in this problem are taken to be \mathbf{r} , \mathbf{v} , and $C_R(A/m)$. We must be very careful with units here, as distance units are mixed in the preceding descriptions.

We have defined a continuous-time dynamical system, but our EGMF has only been developed for discrete-time dynamical systems. This causes no issue, however, as we know that we can replace the discrete-time propagation of mean and covariance with continuous-time propagation, and this is even made more straightforward when there is

no process noise, such as in this problem.

We need to have some initial conditions on our object. To that end, we take the initial distribution of position and velocity to be Gaussian with means

$$\mathbf{m}_{r,0} = \begin{bmatrix} 42000.0 \\ 0.0 \\ 0.0 \end{bmatrix} \text{ km} \quad \text{and} \quad \mathbf{m}_{v,0} = \begin{bmatrix} 0.0 \\ 3.0807 \\ 0.0 \end{bmatrix} \text{ km/s}$$

The covariance for position and velocity is taken to be diagonal with standard deviations of 1 km in position and 0.1 m/s in velocity.

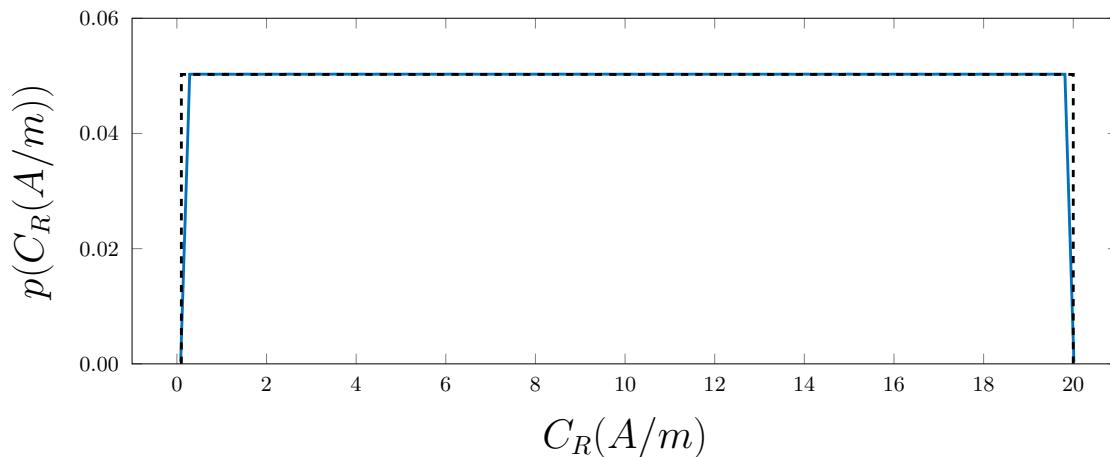
Things get much more interesting when we look at our parameter $C_R(A/m)$. This parameter represents some characterization of the object in terms of it's material properties (C_R) and it's mass and size (A/m). For our problem, we'll assume that we have some bounds on these two parameters, namely:

$$1 \leq C_R \leq 2 \quad \text{and} \quad 0.1 \text{ m}^2/\text{kg} \leq (A/m) \leq 10 \text{ m}^2/\text{kg}$$

Beyond that, we'll assume that we don't know anything about the parameter of our object. From the ranges of the individual contributors, it follows that

$$0.1 \text{ m}^2/\text{kg} \leq C_R(A/m) \leq 20 \text{ m}^2/\text{kg}$$

Since we don't know anything other than these bounds, let's assume that the prior distribution is uniform on this interval. Since we're operating with a EGMF, we can't use a uniform distribution in its natural form, but we can formulate a GM representation of the uniform distribution, such as



This GM approximation of the uniform pdf is accomplished with a set of 1343 Gaussian components, each with a standard deviation of $0.01 \text{ m}^2/\text{kg}$ and a weight of 7.446×10^{-4} .

What do we do with all of these components and our initial Gaussian in position and velocity? For every GM component of $C_R(A/m)$, we

- use the weight of the component as the GM weight for our state pdf
- append the mean of the component to the position/velocity mean
- append the covariance of the component to the position/velocity covariance, assuming that $C_R(A/m)$ is uncorrelated to position and velocity

This leads to a set of 1343 components in the full state and represents our initial GM representation of the full-state pdf.

Measurements of right-ascension and declination are simulated from the Haleakalā site over a period of 14 days. The measurement noise is taken to be zero-mean, Gaussian, with a standard deviation of $10''$. Measurements are only simulated when the solar elevation is $< -18^\circ$ (astronomical sunset) and when the object is above an elevation mask

of 30° . Measurements are limited to no more than 3 sequential observations spread apart within a tracklet by 60 seconds with a minimum spacing between tracklets of 5 hours. These conditions yield a total of 84 right-ascension/declination measurements over the span of 2 weeks (2 tracklets per night, 3 measurements per tracklet).

Now, we're ready to apply the EGMF to this problem. We have a GM representation of our initial pdf, no process noise, and Gaussian measurement noise. How does the lack of process noise influence our EGMF algorithm?

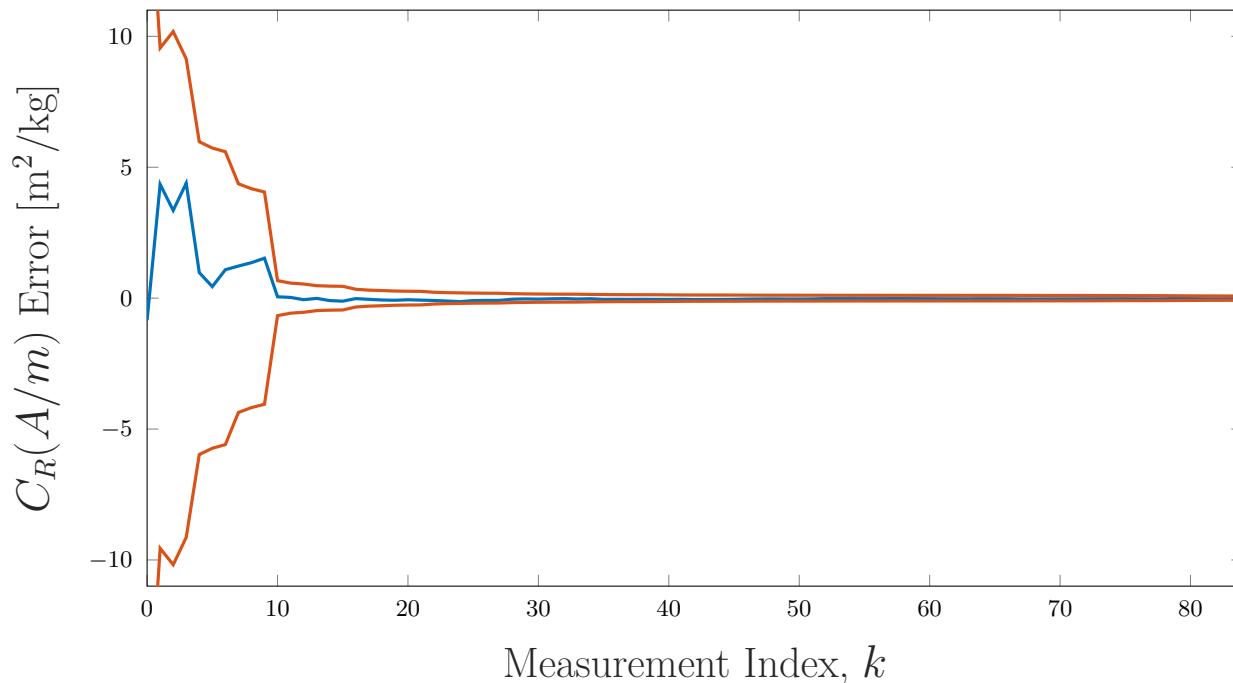
If we apply the EGMF, we'll always have 1343 components. We previously noted that we can remove components with “small” weights. If we just look for “small” weights, we might be tempted to remove components that are actually large relative to the other components; instead, we define a relative tolerance, \varkappa . Provided that the i^{th} posterior component is such that

$$w_{x,k}^{(i)+} < \varkappa \max_{\ell \in \{1, \dots, L_{x,k}^+\}} w_{x,k}^{(\ell)+}$$

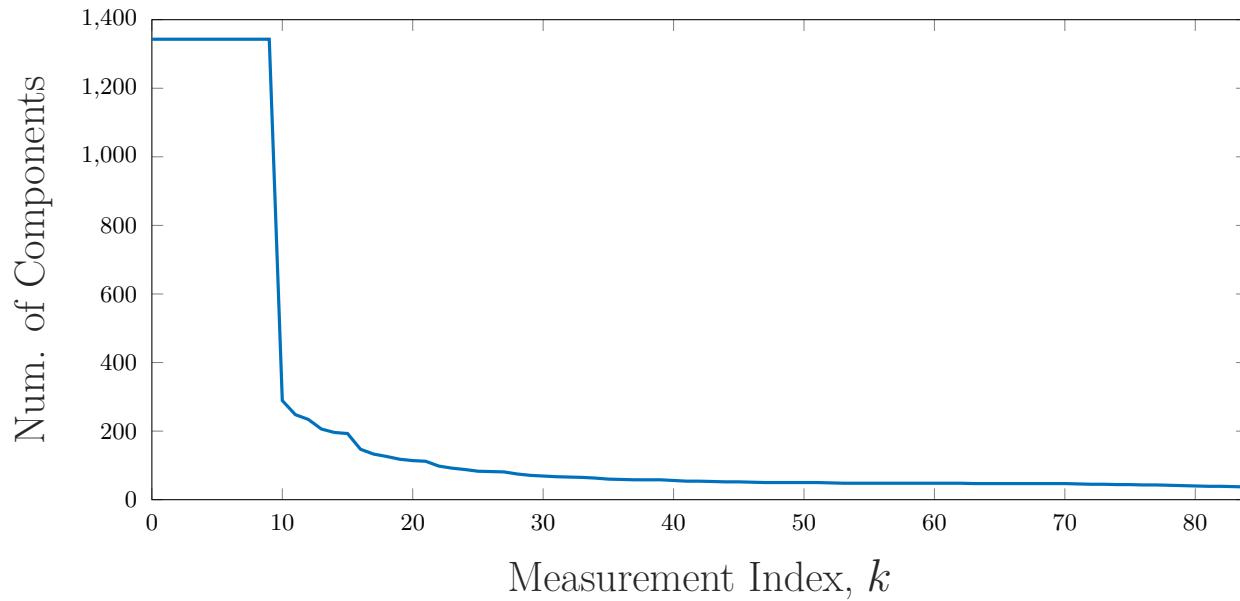
it is removed. All such components satisfying this condition are removed. For this problem, we'll use $\varkappa = 10^{-20}$, which is a pretty small relative threshold.

We will focus the analysis on the estimation of the $C_R(A/m)$ parameter, by looking at the posterior pdf of the parameter at each measurement index.

For a more conventional analysis, we can look at the error in the estimate of the parameter (taken to be the conditional mean) and a measure of our confidence in the estimate (taken to be the conditional (co)variance).



At initialization, we have a standard deviation in $C_R(A/m)$ of 5.7404 m²/kg, and after processing 84 line-of-sight measurements, we have a standard deviation of 0.0268 m²/kg, which is less than 0.5% of the initial uncertainty.



Whereas we started with 1343 components, removing components allows us to end up

with only 37 components. There is an especially steep drop in the number of components that occurs around the processing of the 10th measurement. This is also about the time that the filter “locks” on and the uncertainty drops.