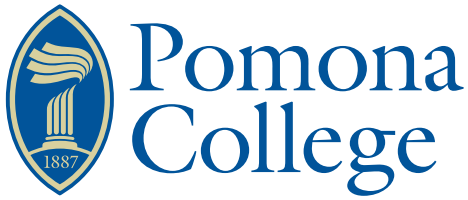


Senior Thesis in Mathematics

An Exploration of Kalman Filters

Author:
Lindsey Tam

Advisor:
Dr. Blerta Shtylla

Submitted to Pomona College in Partial Fulfillment
of the Degree of Bachelor of Arts

January 20, 2020

Abstract

Kalman Filters create predictions for linear systems by recursively generating and correcting predictions according to incoming data. This process enables estimates to become progressively more accurate without relying on large amounts of initial data. Nonlinear forms of the Kalman Filter exist and include the Extended Kalman Filter, the Unscented Kalman Filter, and the Dual Unscented Kalman Filter. This paper explores the theory behind each of these filters and implements a few examples of the Unscented Kalman Filter.

Contents

Chapter 1

Introduction

This paper explores the Kalman Filter, Extended Kalman Filter, Unscented Kalman Filter (UKF), and the Dual Unscented Kalman Filter. The Kalman Filter is a data prediction method that estimates the value of unknown parameters of linear dynamical systems. This is a recursive predictive-corrective process that allows Kalman Filters to continuously generate predictions about state variables without relying on large amounts of initial data. In the case of nonlinear systems, alternative forms of the Kalman Filter were developed, including the Extended Kalman Filter, the Unscented Kalman Filter, and the Dual Unscented Kalman Filter. The Extended Kalman filter linearizes the nonlinear system around the mean by taking the Jacobean of the nonlinear function. Navigation and signal processing are the main applications of the Extended Kalman Filter. Instead of linearizing the system around a single points, the Unscented Kalman Filter utilizes many points, known as Sigma points. The Sigma points then undergo a nonlinear transformation known as the Unscented Transform. The Dual Unscented Kalman Filter is similar to the Unscented Kalman Filter, but also estimates the value of some unknown parameter in the system. Applications of the Unscented and Dual Unscented Kalman Filters include modeling biological systems.

In addition to understanding the theory behind these algorithms, this paper also explores applications of the UKF. An example of the UKF can be applied to the Van der Pol oscillator, which is a self sustaining nonconservative oscillator.

Chapter 2

Kalman Filters

The Kalman Filter (KF) generates predictions for linear dynamical systems [6]. Assuming the given data is noisy and Gaussian [5, 7], the first predictive step assumes knowledge of initial states and the model process. The next step involves calculating the covariance in order to calculate the Kalman Gain, which is a measure of how much the estimate should be changed given actual measurements of the system. The corrective step utilizes the Kalman Gain and the measured system outputs to gauge whether to depend more on the model or on the data. This process can be done recursively, allowing the model to become progressively more accurate as more data is added. Therefore, overtime, it is expected that the model will converge with the actual system measurements.

The Kalman Filter had many aeronautical applications, which include long-distance flight, autopilots, space exploration and military defense systems. In fact, was even used in the Apollo Program [**?**].

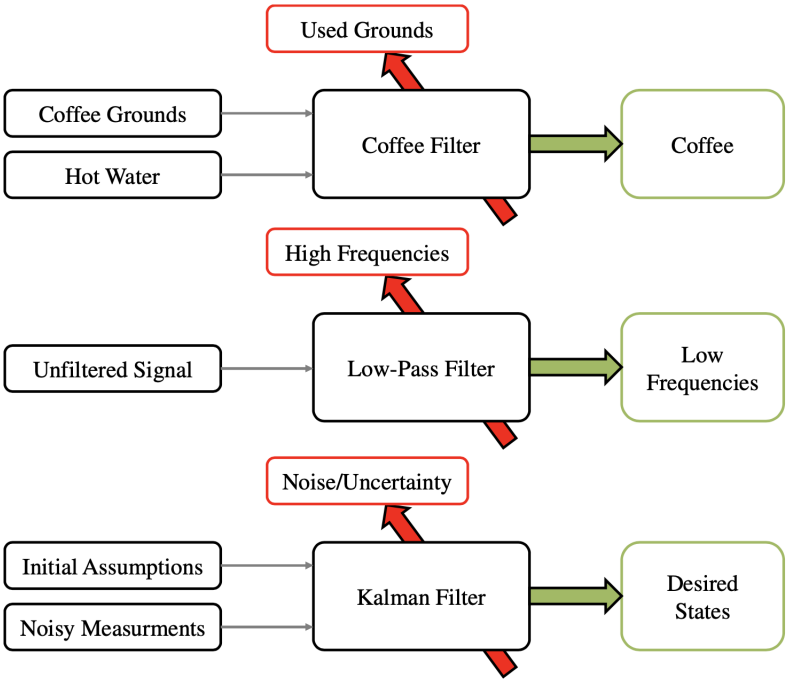


Figure 2.1: A screen-shot taken from a paper by Rhudy et al. comparing the Kalman Filter to a coffee filter [5].

The Kalman Filter is named after its developer, Rudolph Emil Kalman. Kalman was born on May 19, 1930 in Budapest, Hungary. After arriving in the United States, Kalman completed his undergraduate studies and masters degree in electrical engineering from Massachusetts Institute of Technology and completed his doctoral degree in Columbia University. He would spend the next years of his life teaching. In the 1960s to 1970s he became a professor at Stanford university [**?**]. In the 1970s and 1990s, Kalman spent time as professor of engineering at the University of Florida. Kalman is most known for his work on the Kalman Filter, which was developed in the late 1950s. The Kalman Filter greatly aided the United States' military projects, resulting in formed President Obama to award Kalman the National Medal of Science in 2009. In addition, in 1985, Kalman was awarded the Kyoto Prize, which is the Japanese version of the Nobel Peace Prize in the united States. Kalman passed away July 2, 2016 at the age of 86 and is survived by his wife and two children [**?**].

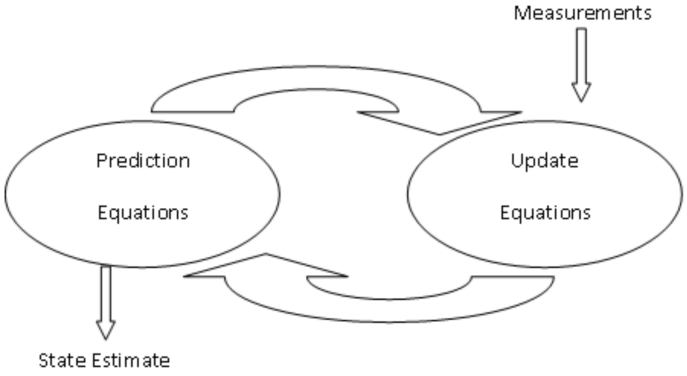


Figure 2.2: A basic diagram of the Kalman Filter [3].

2.1 Kalman Filter Algorithm

The Kalman Filter predicts the system's state variable at the next time step given knowledge of the system. This is modeled by the formula.

$$x_{k+1} = Fx_k + w_k,$$

where x_{k+1} is the state system at the next time step, F is the state function, and w_k is process noise. Here, measurement noise is normally distributed with a mean of 0.

To initialize the model, it is assumed that the state system at step 0 is normally distributed, or

$$x_0 = N(m_0, c_0),$$

where m_0 is the mean of the data and c_0 is the covariance.

1. Begin by initializing the state estimate and the initial state covariance matrix. The state estimate, x_0, can be found by taking the expected value of the data where data is normally distributed which is denoted by:

$$\hat{x}_0 = \mathbb{E}[x_0] = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}.$$

The state covariance matrix, P_0, is a square matrix whose contents are the covariance of the pairwise elements [1] , or

$$P_0 = \begin{bmatrix} var(x_1) & \dots & cov(x_1, x_n) \\ \vdots & \ddots & \vdots \\ cov(x_n, x_1) & \dots & var(x_n) \end{bmatrix}.$$

[1] Recall that the covariance of a variable with itself is the variance of the variable

Enough should be known about the modeled system to generate these values. In general, $\hat{x}_k$ denotes the model's estimate of the state variables. When initializing these values, we use x_0, which is the actual initial measurements of the system.

A state covariance matrix is a symmetrical positive semi-definite square matrix whose diagonals correspond to the variance of a variable at location i and elsewhere is the covariance of the pairwise elements. In practice, covariance matrices help us better understand the spread of data. For the case of the KF, calculating the state covariance is necessary for computing Kalman Gain.

2. After initializing the state estimate and state covariance, a prediction can be generated. The estimate of the system at the next time step, x_{k+1}, is given by

$$x_{k+1} = Fx_k + Gu_k + w_k,$$

where F is the state matrix, G is the F and input matrix, u_k is an input vector and w_k is the process noise vector. Every state variable contained in x_k is defined by a linear differential equation. These linear differential equations can be used to generate the F and G matrices. Therefore, F should be a square matrix whose dimension is equal to the number of states variables and G can be a matrix (or in some cases a vector), depending on the dimension of input vector u.

u_k is an input vector, which is a measurable value that helps define the system, but is not contained in the state vector. Depending on how the system is defined, u_k can be a constant value or it can be a value dependent on time step k.

w_k is the process noise vector at time k. Process noise can be thought of as the model's accuracy. When process noise is 0, it implies that the model is perfectly accurate and does not have to correct for incoming system measurements. On the other hand, high process noise will essentially restate the system based on incoming measurements. w_k has the same dimensions as x_k, allowing us to identify whether or not to adjust the equations for the state variables. When all values of w_k equal 0, it implies that the linear differential equations we are using to define the state variables have no error.

3. Next, the state covariance matrix to calculate Kalman Gain. The state covariance matrix at time step k given the last time step, is

$$P_{k|k-1} = FP_{k-1}F^T + Q_{k-1},$$

where F^T is the transpose of F, and Q_k is the process noise covariance of

w_k. The Kalman Gain at time step k, is given by:

$$K_k = P_{k|k-1}H_K^T(H_k P_{k|k-1}H_K^T + R_k)^{-1},$$

where H is the observation matrix and R is measurement noise covariance. H enables the state variables to be linearly transformed to match the outputs of the system. The dimensions of H reflects which state variables have measurable values in the system. It is not assumed that every state variable is measurable, so H allows us to compare the measurable state variables to x_k. Simple applications of H include creating matrices with 0's and 1's, with 1's denoting that a state variable is measurable and a 0's representing non-measurable states. In other cases, H is an integer used as a scaling factor.

The Kalman Gain is the main component of the corrective aspect of the KF. The Kalman Gain is a measure of how much to change the model based on incoming data. Low values of the Kalman Gain imply the model is accurate while higher values indicate the model should adjust based on the incoming data.

c From this equation, one can see that balancing Q and R is critical for model performance. Larger values of Q indicate higher modeling error, which leads to a higher Kalman Gain and increased model correction. On the other hand, large values of R imply high measurement error, leading to a lower Kalman Gain and less model correction.

4. Next, calculate the transformed output vector and correct the prediction. The transformed output vector, $\hat{y}_k$, is given by

$$\hat{y}_k = H_k x_k + v_k,$$

where v_k is measurement noise, which is added to account for measurement error. The corrected prediction, x_k, is given by:

$$x_k = x_{k-1} + K_k(y_k - \hat{y}_k),$$

where y_k is the actual measurements of the system, K_k is the Kalman Gain, and x_{k-1} are the values of the state variable at the last time step. $\hat{y}_k$ is a way of transforming the prediction (using H) into a format that can be compared with y_k. Subtracting the actual measurements from the predicted measurements is necessary, because y_k does not always contain measurements for all variables in state vector x_k. The quantity $y_k - \hat{y}_k$ is also known as measurement residual or innovation.

5. The final step is to update the state covariance matrix, P_k, through the equation:

$$P_k = (I - K_k H_k)P_{k|k-1},$$

where I is the identity matrix, K_k is the Kalman Gain, H_k is the obersevation function, and $P_{k|k-1}$ is the state covariance at time step k given the last time step. P_k will be used in the next iteration of the filter.

Table 2.1: Description of all variables in the Kalman Filter

Variables in the Kalman Filter		
Variable	Description	Dimensions
x	State variables	$d_x \times 1$
y	Output vector	$d_y \times 1$
u	System inputs	$d_u \times 1$
v	Measurement noise	$d_y \times 1$
w	Process noise	$d_x \times 1$
F	State function	$d_x \times d_x$
H	Observation function	$d_y \times d_x$
G	Input function	$d_x \times d_u$
K	Kalman Gain	$d_x \times d_y$
Q	Process noise covariance	$d_x \times d_x$
R	Measurement noise covariance	$d_y \times d_y$
P	Covariance matrix	$d_x \times d_x$

Chapter 3

Extended Kalman Filters

The Extended Kalman Filter (EKF) is the non-linear version of the Kalman Filter. Though it can be used for non linear equations, it is important to note that it is not an optimal estimator. For the most part, the EKF Algorithm is nearly identical to the KF algorithm. The critical difference is in linearizing the state and observation function. The EKF uses the Jacobean to linearly approximate the non-linear function around the mean of the Gaussian distribution. Skipping this step would result in the transformed data being non-Gaussian; though taking the Jacobean enables the transformation to remain Gaussian, it is not exact, resulting in some generalization. Linear approximation through a single point also makes the EKF inefficient when dealing with complex, higher order systems. Because of this, the model is highly subject to error, which can be somewhat reduced by setting accurate initial values. Though these flaws exist, the EKF performs strongly with applications of real time spatial fields, including navigation and positioning systems.

3.1 Extended Kalman Filter Algorithm

Variables in the Extended Kalman Filter		
Variable	Description	Dimensions
x	State variables	$d_x \times 1$
y	Output vector	$d_y \times 1$
u	System inputs	$d_u \times 1$
v	Measurement noise	$d_y \times 1$
w	Process noise	$d_x \times 1$
f	None linear state function	$d_x \times d_x$
F	State function	$d_x \times d_x$
h	Non linear observation function	$d_y \times d_x$
H	Linearized observation function	$d_y \times d_x$
K	Kalman Gain	$d_x \times d_y$
Q	Process noise covariance	$d_x \times d_x$
R	Measurement noise covariance	$d_y \times d_y$
P	Covariance matrix	$d_x \times d_x$

1. Initialize the state estimate (x_0) and the initial covariance matrix (P_0)

$$\hat{x}_0 = \mathbb{E}[x_0] = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$P_0 = \begin{bmatrix} var(x_1) & \dots & cov(x_1, x_n) \\ \vdots & \ddots & \vdots \\ cov(x_n, x_1) & \dots & var(x_n) \end{bmatrix}$$

   Often, P_0 will be initialized as a diagonal matrix with the diaganals being the variance of each state variance and every other value being set to 0.

2. Calculate the Jacobean

$$F = \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

$$H = \frac{\partial h}{\partial x} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_m}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_m}{\partial x_1} & \cdots & \frac{\partial h_{d_y}}{\partial x_n} \end{bmatrix}$$

   Here, both f and h have no closed form solution. f is the non-linear state function and F is the linear approximation of f with dimension d_x. Similarly, h is the non linear observation while H is linearized observation with dimension d_y.

3. Generate a prediction

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

   This equation is different from the one used in the Kalman filter

4. Use the state covariance matrix ($P_{k|k-1}$) to calculate Kalman Gain (K_k)

$$P_{k|k-1} = FP_{k-1}F^T + Q_{k-1}$$

$$K_k = P_{k|k-1}H_K^T(H_kP_{k|k-1}H_K^T + R_k)^{-1}$$

   Recall that H and F are linear approximations of f and h. Therefore, it is assumed that there is some amount of error when calculating the Kalman Gain.

5. Correct the prediction

$$\hat{y}_k = H(x_k + v_k)$$

$$x_k = x_{k-1} + K_k(y_k - \hat{y}_k)$$

6. Update the covariance matrix

$$P_k = (I - K_KH_k)P_{k|k-1}$$

Chapter 4

Unscented Kalman Filters

The Unscented Kalman Filter (UKF) is another nonlinear version of the Kalman Filter developed to address the shortcomings of the EKF. For instance, as opposed to using the Jacobean to linearly approximate around a single point, the UKF uses the Unscented Transform (UT) to approximate around multiple points, known as sigma points. The UT is a method of approximating probability distributions that have undergone a non linear transformation using limited statistics. The UT involves using these sigma points, which are represented in a Sigma Point Matrix, to represent the normal distribution of the data. The covariance and weights of these sigma points are calculated. These sigma points undergo a non-linear transformation, resulting in a posterior distribution that is not normal [6, 8] . We are able to approximate the normal distribution of the posterior distribution using the weights and covariance that were calculated prior to the transformation. This process enables the Kalman Filter to be applied to more complex non linear problems.

Unlike the Kalman Filter and the Extended Kalman Filter, the UKF also has a set of parameters. Explanations of each parameter and their default values can be found in the chart below. For the UKF, parameters are necessary for controlling the spread of sigma points. This was not needed for the EKF, since the EKF was only linearizing around the mean. expand here on how to tune parameters.

The term 'unscented' was arbitrarily coined by the developer of the UKF, Jeffrey Uhlmann. In an interview, he shares:

> "Initially I only referred to it as the "new filter." Needing a more specific name, people in my lab began referring to it as the "Uhlmann filter," which obviously isn't a name that I could use, so I had to come up with an official term. One evening everyone else in the lab was at the Royal Opera House, and as I was working I noticed someone's deodorant on a desk. The word "unscented" caught my

eye as the perfect technical term. At first people in the lab thought it was absurd—which is okay because absurdity is my guiding principle—and that it wouldn't catch on. My claim was that people simply accept technical terms as technical terms: for example, does anyone think about why a tree is called a tree?"

4.1 Unscented Kalman Filter Algorithm

Variables in the Unscented Kalman Filter		
Variable	Description	Dimensions
x	Vector containing state variables	$d_x \times 1$
χ	Sigma Point Matrix	$d_x \times (2d_x + 1)$
f	Nonlinear tate function	$d_x \times d_x$
h	Nonlinear observation function	$d_u \times d_x$
u	Input vector	$d_u \times 1$
P	Covariance matrix	$d_x \times d_x$
Q	Process noise covariance	$d_x \times d_x$
R	Measurement noise covariance	$d_y \times d_y$
$W^{(m)}$	Weight for state variable (mean)	scalar
$W^{(c)}$	Weight for covariance	scalar

Parameters in the Unscented Kalman Filter			
	Description	Bounds	Default
α	Controls spread of sigma points	$0 < \alpha \leq 1$	.001
β	Adjust sigma point weight	$\beta \geq 0$	2
κ	Sigma point weighting constant	$0 \leq \kappa \leq 3^*$	0

* Others use $\kappa = 3 - d_x$

1. Initialize state vector and covariance

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

$$P_{x_0} = \mathbb{E}[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T]$$

This step is the exact same as the first step of the KF and the EKF. We need to initialize these values so the model can begin generating and correcting predictions.

It is critical to choose initial values that are close to the actual values of the data. Failure to do causes the model to converge slowly.

2. Calculate sigma points

Sigma points characterize the distribution of the data. The number of

sigma points is deterministic and depends on the dimensions of the system. In general, a UFK will have $2 \cdot d_x + 1$ sigma points, where d_x represents the dimension of the state vector [6–8]. We use the equation below to generate a scalar value that determines how spread out the sigma points are from the mean.

$$\lambda = \alpha^2(d_x + \kappa) - d_x$$

α and κ are both parameters that control for the spread of sigma points around the mean value of the state. The spread of the sigma points is proportional to α . For both α and κ, the smaller the values are, the closer the sigma points are to the mean.

β is a parameter that uses information regarding state distribution to adjust sigma points. β has a default value of 2 if the data is Gaussian.

$$\chi_{\ 0,k-1} = \hat{x}_{k-1}$$

Since the goal is to characterize the distribution, set one of these sigma points to the mean. Half of the remaining points will be smaller than the mean and the other half will be larger than the mean.

$$\chi_{\ i,k-1} = \hat{x}_{k-1} \pm \left(\sqrt{(d_x + \lambda)P_{x_{k-1}}}\right)_i \qquad i = 1, \ldots, 2d_x + 1$$

The square root of a matrix, call it A satisfies the following condition: $A = B^2$. Note that $(\sqrt{(d_x + \lambda)P_{x_{k-1}}})$ is a matrix, and the i subscript is the i^{th} column of the matrix. Also note that $\chi_{\ i,k-1}$ is the i^{th} column of the sigma point matrix at time $k - 1$.

3. Calculate the weights for each sigma point

Weights are scalars used to calculate posterior sigma points after they have undergone a nonlinear transformation. The weights are later used to approximate Gaussian mean and covariance. Weights can have positive or negative values, but will ultimately sum to 1 [1]. The subscript of the weight indicates which sigma point the weight is for.

$$W_0^{(m)} = \frac{\lambda}{d_x + \lambda}$$

This is the weight for the mean of the 0th sigma point.

$$W_0^{(c)} = \frac{\lambda}{d_x + \lambda} + (1 - \alpha^2 + \beta)$$

This is the weight for the covariance of the 0th sigma point.

$$W_i^{(m)} = W_i^{(c)} = \frac{\lambda}{2(d_x + \lambda)} \qquad i = 1, \ldots, 2d_x$$

The rest of the weights for the means and covariances of the other $2d_x$ sigma points is calculated above.

4. Generate a prediction

$$\chi_{k|k-1} = f(\chi)$$

The equation above performs nonlinear transformation f on the sigma points. Though χ has a Gaussian distribution, $\chi_{k|k-1}$ does not because it has been transformed by the nonlinear state function f.

$$\hat{x}_{k|k-1} = \sum_{i=0}^{2d_x} W_i^{(m)} \chi_{i,k|k-1}$$

Now, the new estimate of the state variables is conditioned on the last estimate. The weights are included in this step in order to approximate the Gaussian distribution.

$$P_{x,k|k-1} = \sum_{i=0}^{2d_x} W_i^{(c)} (\chi_{i,k|k-1} - \hat{x}_{i,k|k-1})(\chi_{i,k|k-1} - \hat{x}_{i,k|k-1})^T + Q$$

The posterior covariance matrix for the state variable is necessary for updating the state covariance later on (step 9). Recall that Q is process noise, which provides the error in our model f.

5. Calculate posterior sigma points

By now, the prediction step has concluded and the model begins the process of correction. Calculation of the posterior (also called augmented) sigma points is necessary for interpreting the distribution. From the non linear transformation above, the outputs are not Gaussian. Using the weights calculated in step 3, we are able to approximate the Gaussian distribution of the transformed sigma points.

$$\chi_{0,k|k-1}^{(aug)} = \hat{x}_{k|k-1}$$

$$\chi_{i,k|k-1}^{(aug)} = \hat{x}_{k|k-1} \pm \left(\sqrt{(d_x + \lambda) P_{x_{k|k-1}}} \right)_i \qquad i = 1, \ldots, 2d_x$$

The calculation is the same process as in step 2. Recall that λ was calculated in step 2. Since it is not time dependent, we can use the same λ value used earlier.

6. Calculate transformed output

$$\mathcal{Y}_{k|k-1} = h(\chi_{k|k-1}^{(aug)})$$

$$y_{k|k-1} = \sum_{i=0}^{2d_x} W_i^{(m)} \mathcal{Y}_{i,k|k-1}$$

Similar to the KF and EKF, use observation function h to convert the sigma point matrix into ta form that can be compared to the actual measurements of the system y_k. Since this form is not Gaussian, use weights to interpret the results.

7. Calculate Kalman Gain

   Unlike previous versions of the KF, in addition to calculating the covariance of the state variables, calculations is also done for the covariance of observations P_y and for state variables with observations P_{xy}. When generating the covariance matrix for y the covariance of measurement noise is added.

$$P_{y,k|k-1} = \sum_{i=0}^{2d_x} W_i^{(c)} (\mathcal{Y}_{i,k|k-1} - y_{i,k|k-1})(\mathcal{Y}_{i,k|k-1} - y_{i,k|k-1})^T + R$$

$$P_{xy,k|k-1} = \sum_{i=0}^{2d_x} W_i^{(c)} (\chi_{i,k|k-1}^{(aug)} - \hat{x}_{i,k|k-1})(\mathcal{Y}_{i,k|k-1} - y_{i,k|k-1})^T$$

$$K_k = P_{xy,k|k-1}(P_{y,k|k-1})^{-1}$$

8. Correct the prediction

$$\hat{x}_k = \hat{x}_{k|k-1} + K_k(y_k - y_{k|k-1})$$

   Similar to the KF and EKF, the correction step follows the same equation.

9. Update the covariance matrix

$$P_{x,k} = P_{x,k|k-1} - K_k(P_{y,k|k-1}){K_k}^T$$

4.2 Van der Pol Example

Applying the UKF to the Van der Pol oscillator will be used as a simple example to demonstrate the impacts of measurement and process noise. The Van der Pol equation, so named after its developer Balthasar Van der Pol, describes a self sustaining oscillator that create energy at small amplitudes and remove energy from large amplitudes. The Van der Pol equation describes a nonconservative oscillator (also known as a relaxation oscillator). Applications of the Van der Pol oscillators include circuits, vacuums, and modeling biological systems [9]. The Van der Pol oscillator is represented by a nonlinear second order differential equation:

$$\frac{d^2y}{dt^2} + \mu(y^2 - 1)\frac{dy}{dt} = 0$$

where μ is a damping coefficient, $\frac{d^2y}{dt^2}$ is acceleration, $\frac{dy}{dt}$ is velocity, and y is position. Therefore, for all $\mu < 0$, dampening occurs and the system tends to 0. The rate at which the system converges to zero is dependent on the size of μ, with larger values taking longer to converge and smaller values converging faster. If $\mu = 0$, the system becomes a simple harmonic oscillator, where motion is periodic. Lastly, if $\mu > 0$, the system enters a limit cycle, which is an isolated closed trajectory. [2].

The UKF can be applied to this nonlinear system to determine where the system will be at a point in time. To do so, relevant state variables include position and velocity. [1]

$$x_k = \begin{bmatrix} y \\ v \end{bmatrix}$$

Transforming the Van der Pol equation from a second order differential equation to a first order differential equation makes it easier to define f, the state function of the system. By substituting v for $\frac{dy}{dt}$, and $\frac{dv}{dt}$ for $\frac{d^2y}{dt^2}$ one can rewrite the Van der Pol equation as

$$\frac{dv}{dt} + \mu(y^2 - 1)v + y = 0.$$

From this, we get the differential equations associated with the state variables to generate the nonlinear transformation function f. For the sake of simplicity, assuming $\mu = 1$, we get

$$\dot{x}_k = \begin{bmatrix} \frac{dy}{dt} \\ \\ \frac{d^2y}{dt^2} \end{bmatrix} = \begin{bmatrix} v \\ \\ \frac{dv}{dt} \end{bmatrix} = \begin{bmatrix} v \\ \\ -1(y^2 - 1)v - y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \\ 1 - y^2 - 1 - y \end{bmatrix} x_k = f.$$

[1] In Matlab, x(1) and x(2) represent position and velocity, respectively

In this particular example, the only measurement received from the system is position. Therefore, this filter is continually correcting for the position state state variable through measurement function h. Here, measurements of the system are simulated by adding noise to the position state variable. Even though there are only measurements for one state variable, we can still generate estimates of both state variables. In terms of parameters α, κ, and β, default values were used.

To simulate a Van der Pol Oscillator, an ordinary differential equation solver can be applied to state function f to generate true values of the system. Of course, systems do not perform perfectly; variations in model performance can be captured by randomly adding noise to the system.

All of this can be modeled on Matlab; all source code is complimentary of Matlab and can be referenced in Appendix A [**?**]

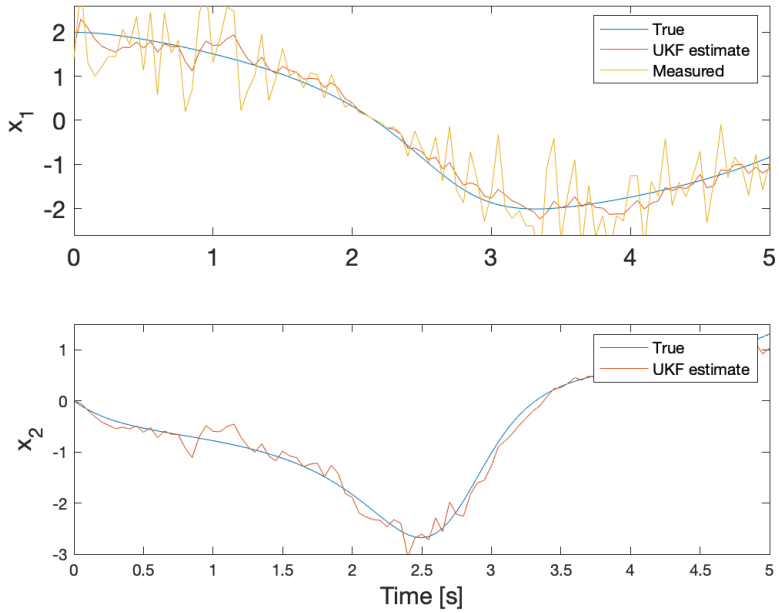


Figure 4.1: Performance of the UKF with R = 0.2, and Q = (0.02 0.1). As expected, the model converges on the true values of the system for both state variables. In this case, the only measurements we are receiving from the system are position, which is why the second state variable has no measured values.

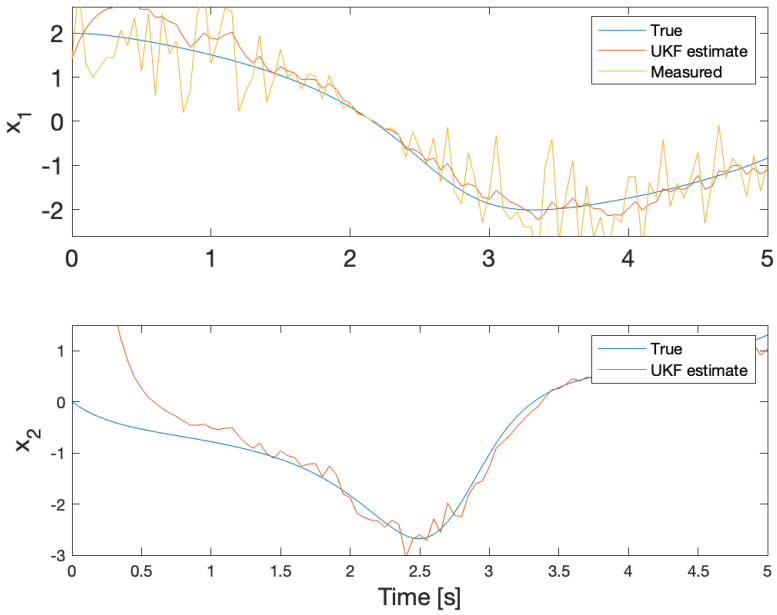
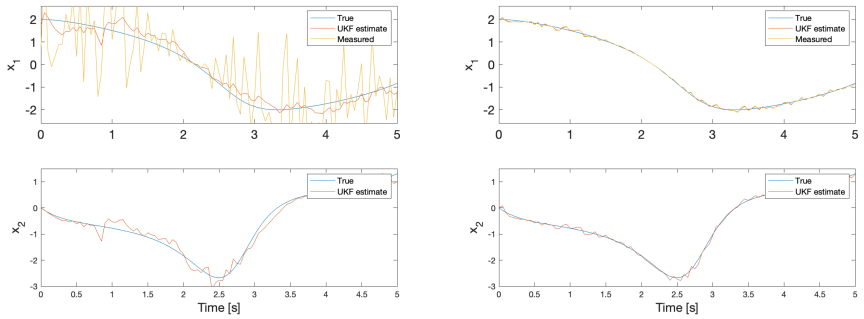
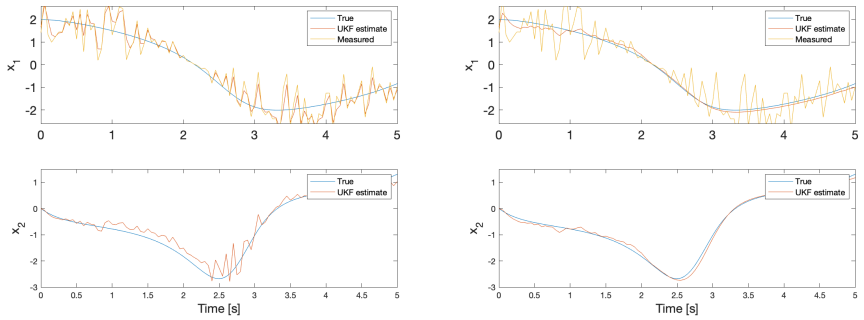


Figure 4.2: Performance of the UKF with poor initial conditions: (1, 7) instead of (2,0). Recall that inaccurate intial conditions cause convergence to take place more slowly. This seems to be the case, especially in the state variable that is not being corrected for.



(a) UKF with high measurement noise (0.9) (b) UKF with low measurement noise (0.002)

Figure 4.3: UKF on VDP oscillator with difference values of measurement noise. The model's behavior changes in response to the different values of measurement noise. Even when measurement noise is high, the UKF continues to perform well. Though, the rate of convergence appears to be slower. On the other hand, when measurement noise is low, the UKF seems to converge instantly with the measured values. The velocity state variable also quickly converges with the true value of the system.



(a) UKF with high process noise (0.9 0.8) (b) UKF with low process noise (0.0001 0.0001)

Figure 4.4: UKF on VDP oscillator with different values of process noise. Recall that process noise measures errors in the model. For the purpose of this example, process error was set to extreme high and low values. In theory, the Van der Pol equation should have small process error because it is a well used equation.

4.3 Modeling a Biological System

This example was inspired by a paper that uses the UKF to model the biological pathway of metabolites. A metabolite is small bodily structure that is the byproduct of the metabolism. Data regarding metabolites is highly influenced by noise, which is a factor that makes other approaches, such as regression and annealing, fail. The system in the model is complex, containing 18 unknown parameters and four state variables. Initial data contained information about metabolites at various time steps [4] .

In the paper, researchers had access to their own data sources and used an approach that was adapted from the UKF. Though we are not using their exact dataset, we will be simulating data using the same approach as the previous example. Unlike the paper, this example will be following UKF algorithm and then comparing the result of both methods. Ultimately, the goal of this example is to demonstrate how an UKF works on higher dimensional and more complex systems, how an UKF corrects for multiple variables, and how parameters can be adjusted to fit the model.

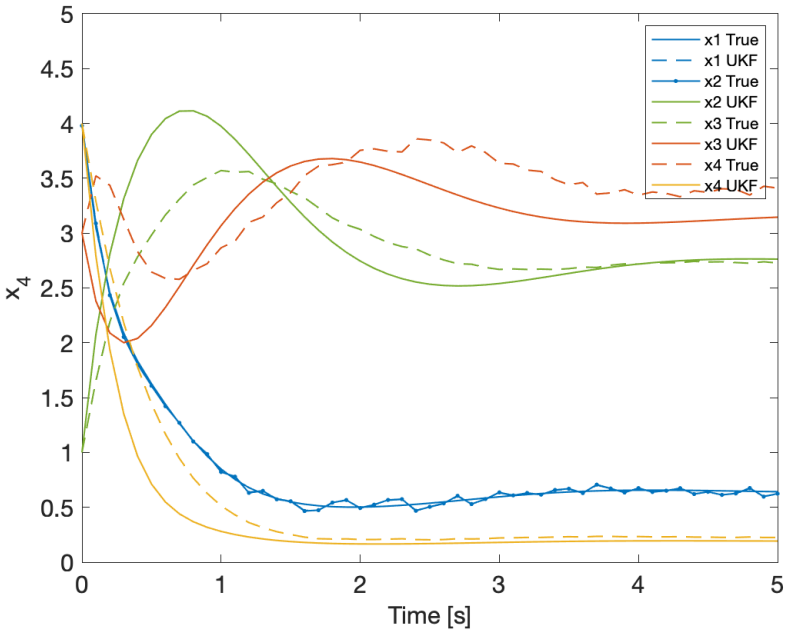


Figure 4.5: Performance of all state variables with the following conditions: R = 0.001, Q = (0.02 0.01 .03 .04) and initial values = (4, 1, 3, 4). Paramaters included:

Similar to the previous example, the system's true output is simulated using an ODE solver. We are given the differential equation for the four state variables.

The paper was able to make the model converge faster by resetting the covariance to re-excite the model. By skipping this step, the three state variables without measurements converge significantly slower.

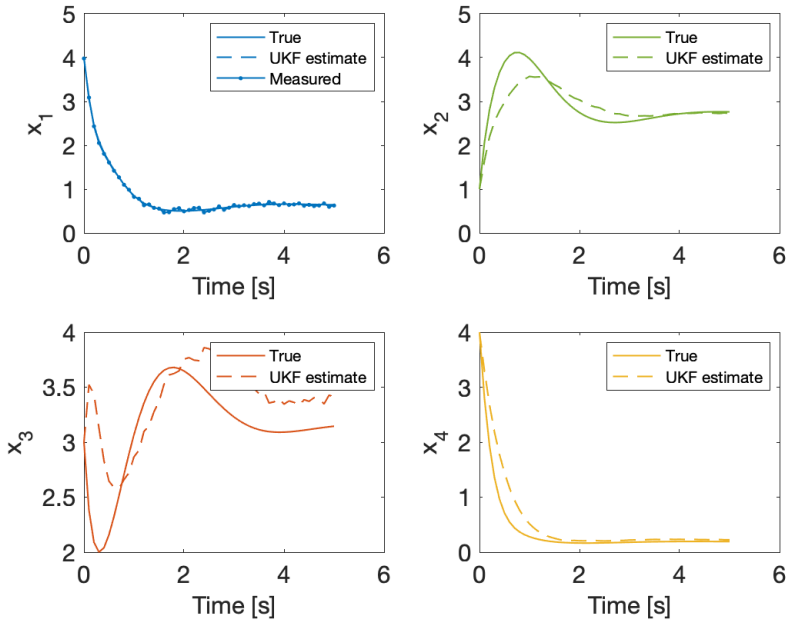


Figure 4.6: Performance of all four state variables with one corrected state. The first state variable (blue) is the only one that is receiving measurements from the system. This explains why it is more accurate and converges better with the system, as compared with the other three states.

Expand here on tweaking parameters.In theory, values of κ can be negative, but cannot be used on MATLAB.

Expand here on correcting for 1+ states

Bibliography

[1] J. Gove and DY Hollinger. Application of a dual unscented kalman filter for simultaneous state and parameter estimation in problems of surface-atmosphere exchange. *Journal of Geophysical Research*, 111:21 PP.–21 PP., 04 2006.

[2] Shuichi Kinoshita. Van der pol oscillator, 2013.

[3] David Kohanbash. Kalman filtering – a practical implementation guide (with code!), Jan 2014.

[4] Nader Meskin, Hazem Nounou, Mohamed Nounou, and Aniruddha Datta. Parameter estimation of biological phenomena: An unscented kalman filter approach. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM*, 10:537–43, 03 2013.

[5] Matthew Rhudy, Roger Salguero, and Keaton Holappa. A kalman filtering tutorial for undergraduate students. *International Journal of Computer Science & Engineering Survey*, 08:01–18, 02 2017.

[6] Esra Saatci and Aydin Akan. *Dual Unscented Kalman Filter and Its Applications to Respiratory System Modeling*, page 4. intechopen, 04 2009.

[7] Eric Wan and Ronell Merwe. The unscented kalman filter for nonlinear estimation. In *The Unscented Kalman Filter for Nonlinear Estimation*, volume 153-158, pages 153 – 158, 02 2000.

[8] Eric A. Wan and Rudolph Van Der Merwe. The unscented kalman filter. In *Kalman Filtering and Neural Networks*, pages 221–280. Wiley, 2001.

[9] Eric W Weisstein. Van der pol equation, Dec 2019.

Appendix A

Van der Pol Code

A.1 Main function

```
% Taken from https://www.mathworks.com/help/control/ug/
    nonlinear-state-estimation-using-unscented-kalman-
    filter.html

initialStateGuess = [2;0]; % xhat[k|k-1]
% Construct the filter
ukf = unscentedKalmanFilter(...
    @vdpStateFcn,... % State transition function
    @vdpMeasurementNonAdditiveNoiseFcn,... % Measurement
        function
    initialStateGuess,...
    'HasAdditiveMeasurementNoise',false);

R = 0.2; % Variance of the measurement noise v[k],
    original
%R = 0.002; % little measurement noise
%R = .9; % high measurement noise
ukf.MeasurementNoise = R;

ukf.ProcessNoise = diag([0.02  0.1]); %original
%ukf.ProcessNoise = diag([0.9  0.8]); %model is corrected
    alot
%ukf.ProcessNoise = diag([0.0001  0.0001]); %model is
    corrected little

T = 0.05; % [s] Filter sample time
timeVector = 0:T:5;
[~,xTrue]=ode45(@vdp1,timeVector,[2;0]);
```

```
23
24 rng(1); % Fix the random number generator for
       reproducible results
25 yTrue = xTrue(:,1);
26 yMeas = yTrue .* (1+sqrt(R)*randn(size(yTrue))); % sqrt(R
       ): Standard deviation of noise
27
28 Nsteps = numel(yMeas); % Number of time steps
29 xCorrectedUKF = zeros(Nsteps,2); % Corrected state
       estimates
30 PCorrected = zeros(Nsteps,2,2); % Corrected state
       estimation error covariances
31 e = zeros(Nsteps,1); % Residuals (or innovations)
32
33 for k=1:Nsteps
34     % Let k denote the current time.
35     %
36     % Residuals (or innovations): Measured output -
           Predicted output
37     e(k) = yMeas(k) - vdpMeasurementFcn(ukf.State); % ukf
           .State is x[k|k-1] at this point
38     % Incorporate the measurements at time k into the
           state estimates by
39     % using the "correct" command. This updates the State
            and StateCovariance
40     % properties of the filter to contain x[k|k] and P[k|
           k]. These values
41     % are also produced as the output of the "correct"
           command.
42     [xCorrectedUKF(k,:), PCorrected(k,:,:)] = correct(ukf
           ,yMeas(k));
43     % Predict the states at next time step, k+1. This
           updates the State and
44     % StateCovariance properties of the filter to contain
            x[k+1|k] and
45     % P[k+1|k]. These will be utilized by the filter at
           the next time step.
46     predict(ukf);
47 end
48
49
50 figure();
51 subplot(2,1,1);
52 plot(timeVector,xTrue(:,1),timeVector,xCorrectedUKF(:,1),
       timeVector,yMeas(:));
53 set(gca, 'FontSize', 15);
```

```
54 e=legend('True','UKF estimate','Measured')
55 e.FontSize = 10;
56 ylim([-2.6  2.6]);
57 ylabel('x_1', 'FontSize', 15);
58 subplot(2,1,2);%
59 plot(timeVector,xTrue(:,2),timeVector,xCorrectedUKF(:,2))
       ;
60 f=legend('True','UKF estimate')
61 f.FontSize = 10;
62 set(gca, 'FontSize', 10);
63 ylim([-3  1.5]);
64 xlabel('Time [s]', 'FontSize', 15);
65 ylabel('x_2', 'FontSize', 15);
66
67 %saveas(gcf,'\Users\lindseytam\Desktop\thesis\
       VDP_highMN_highPN.png')
```

A.2 State function

```
1  function x = vdpStateFcn(x)
2  % vdpStateFcn Discrete-time approximation to van der Pol
       ODEs for mu = 1.
3  % Sample time is 0.05s.
4  %
5  % Example state transition function for discrete-time
       nonlinear state
6  % estimators.
7  %
8  % xk1 = vdpStateFcn(xk)
9  %
10 % Inputs:
11 %    xk - States x[k]
12 %
13 % Outputs:
14 %    xk1 - Propagated states x[k+1]
15 %
16 % See also extendedKalmanFilter, unscentedKalmanFilter
17
18 %   Copyright 2016 The MathWorks, Inc.
19
20 %#codegen
21
22 % The tag %#codegen must be included if you wish to
       generate code with
23 % MATLAB Coder.
24
```

```
% Euler integration of continuous-time dynamics x'=f(x)
    with sample time dt
dt = 0.05; % [s] Sample time
x = x + vdpStateFcnContinuous(x)*dt;
end

function dxdt = vdpStateFcnContinuous(x)
%vdpStateFcnContinuous Evaluate the van der Pol ODEs for
    mu = 1
dxdt = [x(2); (1-x(1)^2)*x(2)-x(1)];
end
```

A.3 Measurement function

```
initialStateGuess = [4;1;3;4]; % xhat[k|k-1]
%initialStateGuess = [1;4;2;5]; % bad intials

% Construct the filter
ukf = unscentedKalmanFilter(...
    @MeskinStateFcn,... % State transition function
    @MeskinMeasurementNonAddFcn,... % Measurement
        function
    initialStateGuess,...
    'HasAdditiveMeasurementNoise',false)%,...
    %'alpha', 0.9,...
    %'kappa', 3,...
    %'beta', 2);

R = .001; % Variance of the measurement noise v[k]
ukf.MeasurementNoise = R;

ukf.ProcessNoise = diag([0.02 0.01 .03 .04]); %stores the
    process noise covariance

T = 0.1; % [s] Filter sample time
timeVector = 0:T:5;
[~,xTrue]=ode45(@Meskin1,timeVector,initialStateGuess);

rng(1); % Fix the random number generator for
    reproducible results
yTrue = xTrue(:,1);
yMeas = yTrue + (sqrt(R)*randn(size(yTrue)));
% sqrt(R): Standard deviation of noise
% randn(size(yTrue)): randomly sample 51 elements from
    ytrue
%yMeas = yTrue; Assumes no noise
```

```
29
30 for k=1:numel(yMeas)
31     % Let k denote the current time.
32
33     % Residuals (or innovations): Measured output -
           Predicted output
34     %e(k) = yMeas(k) - MeskinMeasurementFcn(ukf.State); %
           ukf.State is x[k|k-1] at this point
35
36     % Incorporate the measurements at time k into the
           state estimates by
37     % using the "correct" command. This updates the State
           and StateCovariance
38     % properties of the filter to contain x[k|k] and P[k|
           k]. These values
39     % are also produced as the output of the "correct"
           command.
40
41     [xCorrectedUKF(k,:), PCorrected(k,:,:)] = correct(ukf
           , yMeas(k));
42
43     % Predict the states at next time step, k+1. This
           updates the State and
44     % StateCovariance properties of the filter to contain
           x[k+1|k] and
45     % P[k+1|k]. These will be utilized by the filter at
           the next time step.
46     predict(ukf);
47
48 end
49
50
51 figure();
52 color1=[0,0.4470, 0.7410]; %blue
53 color2=[0.4660 0.6740 0.1880]; %green
54 color3=[0.8500 0.3250 0.0980]; %orange
55 color4=[0.9290 0.6940 0.1250]; %yellow
56
57 subplot(2,2,1);
58 plot(timeVector, xTrue(:,1), 'Color', color1, 'LineStyle'
       , '-',...
59     'LineWidth', 1)
60 hold on
61 plot(timeVector,xCorrectedUKF(:,1), 'Color', color1, '
       LineStyle', '--',...
62     'LineWidth', 1)
```

```
63  hold on
64  plot(timeVector,yMeas, 'Color', color1, 'Marker', '.',...
65      'LineWidth', 1)
66
67  set(gca, 'FontSize', 15);
68  set(gcf, 'Color', 'None');
69  a = legend('True','UKF estimate','Measured')
70  a.FontSize = 10;
71  ylim([0 5]);
72  ylabel('x_1', 'FontSize', 15);
73  xlabel('Time [s]', 'FontSize', 15);
74
75  subplot(2,2,2);
76  plot(timeVector, xTrue(:,2), 'Color', color2, 'LineStyle'
        , '-',...
77      'LineWidth', 1)
78  hold on
79  plot(timeVector,xCorrectedUKF(:,2), 'Color', color2, '
        LineStyle', '--',...
80      'LineWidth', 1)
81  set(gca, 'FontSize', 15);
82  b = legend('True','UKF estimate')
83  b.FontSize = 10;
84  ylim([0 5]);
85  xlabel('Time [s]', 'FontSize', 15);
86  ylabel('x_2', 'FontSize', 15);
87  xlabel('Time [s]', 'FontSize', 15);
88
89  subplot(2,2,3);
90  plot(timeVector, xTrue(:,3), 'Color', color3, 'LineStyle'
        , '-',...
91      'LineWidth', 1)
92  hold on
93  plot(timeVector,xCorrectedUKF(:,3), 'Color', color3, '
        LineStyle', '--',...
94      'LineWidth', 1)
95  set(gca, 'FontSize', 15);
96  c = legend('True','UKF estimate')
97  c.FontSize = 10;
98  %ylim([-2.6 2.6]);
99  ylabel('x_3', 'FontSize', 15);
100 xlabel('Time [s]', 'FontSize', 15);
101
102 subplot(2,2,4);
103 plot(timeVector, xTrue(:,4), 'Color', color4, 'LineStyle'
        , '-',...
```

```
104      'LineWidth', 1)
105  hold on
106  plot(timeVector,xCorrectedUKF(:,4), 'Color', color4, '
         LineStyle', '--',...
107      'LineWidth', 1)
108  set(gca, 'FontSize', 15);
109  d = legend('True','UKF estimate','Measured')
110  d.FontSize = 10;
111  %ylim([-2.6 2.6]);
112  ylabel('x_4', 'FontSize', 15);
113  xlabel('Time [s]', 'FontSize', 15);
114
115  %saveas(gcf,'\Users\lindseytam\Desktop\thesis\
         Meskin_states_badIntial.png')
116  %saveas(gcf,'\Users\lindseytam\Desktop\thesis\
         Meskin_states.png')
117
118  subplot(1,1,1);
119
120  plot(timeVector, xTrue(:,1), 'Color', color1, 'LineStyle'
         , '-',...
121      'LineWidth', 1)
122  hold on
123  plot(timeVector,xCorrectedUKF(:,1), 'Color', color1, '
         LineStyle', '--',...
124      'LineWidth', 1)
125  hold on
126  plot(timeVector,yMeas, 'Color', color1, 'Marker', '.',...
127      'LineWidth', 1)
128  hold on
129  plot(timeVector, xTrue(:,2), 'Color', color2, 'LineStyle'
         , '-',...
130      'LineWidth', 1)
131  hold on
132  plot(timeVector,xCorrectedUKF(:,2), 'Color', color2, '
         LineStyle', '--',...
133      'LineWidth', 1)
134  hold on
135  plot(timeVector, xTrue(:,3), 'Color', color3, 'LineStyle'
         , '-',...
136      'LineWidth', 1)
137  hold on
138  plot(timeVector,xCorrectedUKF(:,3), 'Color', color3, '
         LineStyle', '--',...
139      'LineWidth', 1)
140  hold on
```

```
141 plot(timeVector, xTrue(:,4), 'Color', color4, 'LineStyle'
        , '-',...
142     'LineWidth', 1)
143 hold on
144 plot(timeVector,xCorrectedUKF(:,4), 'Color', color4, '
    LineStyle', '--',...
145     'LineWidth', 1)
146
147 set(gca, 'FontSize', 15);
148 set(gcf, 'color', 'none');
149 set(gca, 'color', 'none');
150 e = legend('x1 True','x1 UKF','x2 True', 'x2 UKF', 'x3
    True','x3 UKF','x4 True', 'x4 UKF')
151 e.FontSize = 10;
152 ylim([0 5]);
153 ylabel('x_4', 'FontSize', 15);
154 xlabel('Time [s]', 'FontSize', 15);
155
156 %saveas(gcf,'\Users\lindseytam\Desktop\thesis\
    Meskin_overall_badIntial.png')
157 saveas(gcf,'\Users\lindseytam\Desktop\thesis\
    Meskin_overall.png')
```

Appendix B

Meskin Code

B.1 Main function

```
initialStateGuess = [4;1;3;4]; % xhat[k|k-1]
%initialStateGuess = [1;4;2;5]; % bad intials

% Construct the filter
ukf = unscentedKalmanFilter(...
    @MeskinStateFcn,... % State transition function
    @MeskinMeasurementNonAddFcn,... % Measurement
        function
    initialStateGuess,...
    'HasAdditiveMeasurementNoise',false)%,...
    %'alpha', 0.9,...
    %'kappa', 3,...
    %'beta', 2);

R = .001; % Variance of the measurement noise v[k]
ukf.MeasurementNoise = R;

ukf.ProcessNoise = diag([0.02 0.01 .03 .04]); %stores the
    process noise covariance

T = 0.1; % [s] Filter sample time
timeVector = 0:T:5;
[~,xTrue]=ode45(@Meskin1,timeVector,initialStateGuess);

rng(1); % Fix the random number generator for
    reproducible results
yTrue = xTrue(:,1);
yMeas = yTrue + (sqrt(R)*randn(size(yTrue)));
```

```
% sqrt(R): Standard deviation of noise
% randn(size(yTrue)): randomly sample 51 elements from
    ytrue
%yMeas = yTrue; Assumes no noise

for k=1:numel(yMeas)
    % Let k denote the current time.

    % Residuals (or innovations): Measured output -
        Predicted output
    %e(k) = yMeas(k) - MeskinMeasurementFcn(ukf.State); %
        ukf.State is x[k|k-1] at this point

    % Incorporate the measurements at time k into the
        state estimates by
    % using the "correct" command. This updates the State
        and StateCovariance
    % properties of the filter to contain x[k|k] and P[k|
        k]. These values
    % are also produced as the output of the "correct"
        command.

    [xCorrectedUKF(k,:), PCorrected(k,:,:)] = correct(ukf
        , yMeas(k));

    % Predict the states at next time step, k+1. This
        updates the State and
    % StateCovariance properties of the filter to contain
        x[k+1|k] and
    % P[k+1|k]. These will be utilized by the filter at
        the next time step.
    predict(ukf);

end


figure();
color1=[0,0.4470, 0.7410]; %blue
color2=[0.4660 0.6740 0.1880]; %green
color3=[0.8500 0.3250 0.0980]; %orange
color4=[0.9290 0.6940 0.1250]; %yellow

subplot(2,2,1);
plot(timeVector, xTrue(:,1), 'Color', color1, 'LineStyle'
    , '-',...
    'LineWidth', 1)
```

```
60  hold on
61  plot(timeVector,xCorrectedUKF(:,1), 'Color', color1, '
        LineStyle', '--',...
62      'LineWidth', 1)
63  hold on
64  plot(timeVector,yMeas, 'Color', color1, 'Marker', '.',...
65      'LineWidth', 1)
66
67  set(gca, 'FontSize', 15);
68  set(gcf, 'Color', 'None');
69  a = legend('True','UKF estimate','Measured')
70  a.FontSize = 10;
71  ylim([0 5]);
72  ylabel('x_1', 'FontSize', 15);
73  xlabel('Time [s]', 'FontSize', 15);
74
75  subplot(2,2,2);
76  plot(timeVector, xTrue(:,2), 'Color', color2, 'LineStyle'
        , '-',...
77      'LineWidth', 1)
78  hold on
79  plot(timeVector,xCorrectedUKF(:,2), 'Color', color2, '
        LineStyle', '--',...
80      'LineWidth', 1)
81  set(gca, 'FontSize', 15);
82  b = legend('True','UKF estimate')
83  b.FontSize = 10;
84  ylim([0 5]);
85  xlabel('Time [s]', 'FontSize', 15);
86  ylabel('x_2', 'FontSize', 15);
87  xlabel('Time [s]', 'FontSize', 15);
88
89  subplot(2,2,3);
90  plot(timeVector, xTrue(:,3), 'Color', color3, 'LineStyle'
        , '-',...
91      'LineWidth', 1)
92  hold on
93  plot(timeVector,xCorrectedUKF(:,3), 'Color', color3, '
        LineStyle', '--',...
94      'LineWidth', 1)
95  set(gca, 'FontSize', 15);
96  c = legend('True','UKF estimate')
97  c.FontSize = 10;
98  %ylim([-2.6 2.6]);
99  ylabel('x_3', 'FontSize', 15);
100 xlabel('Time [s]', 'FontSize', 15);
```

```
101
102 subplot(2,2,4);
103 plot(timeVector, xTrue(:,4), 'Color', color4, 'LineStyle'
    , '-',...
104     'LineWidth', 1)
105 hold on
106 plot(timeVector,xCorrectedUKF(:,4), 'Color', color4, '
    LineStyle', '--',...
107     'LineWidth', 1)
108 set(gca, 'FontSize', 15);
109 d = legend('True','UKF estimate','Measured')
110 d.FontSize = 10;
111 %ylim([-2.6 2.6]);
112 ylabel('x_4', 'FontSize', 15);
113 xlabel('Time [s]', 'FontSize', 15);
114
115 %saveas(gcf,'\Users\lindseytam\Desktop\thesis\
    Meskin_states_badIntial.png')
116 %saveas(gcf,'\Users\lindseytam\Desktop\thesis\
    Meskin_states.png')
117
118 subplot(1,1,1);
119
120 plot(timeVector, xTrue(:,1), 'Color', color1, 'LineStyle'
    , '-',...
121     'LineWidth', 1)
122 hold on
123 plot(timeVector,xCorrectedUKF(:,1), 'Color', color1, '
    LineStyle', '--',...
124     'LineWidth', 1)
125 hold on
126 plot(timeVector,yMeas, 'Color', color1, 'Marker', '.',...
127     'LineWidth', 1)
128 hold on
129 plot(timeVector, xTrue(:,2), 'Color', color2, 'LineStyle'
    , '-',...
130     'LineWidth', 1)
131 hold on
132 plot(timeVector,xCorrectedUKF(:,2), 'Color', color2, '
    LineStyle', '--',...
133     'LineWidth', 1)
134 hold on
135 plot(timeVector, xTrue(:,3), 'Color', color3, 'LineStyle'
    , '-',...
136     'LineWidth', 1)
137 hold on
```

```
plot(timeVector,xCorrectedUKF(:,3), 'Color', color3, '
    LineStyle', '--',...
     'LineWidth', 1)
hold on
plot(timeVector, xTrue(:,4), 'Color', color4, 'LineStyle'
    , '-',...
     'LineWidth', 1)
hold on
plot(timeVector,xCorrectedUKF(:,4), 'Color', color4, '
    LineStyle', '--',...
     'LineWidth', 1)

set(gca, 'FontSize', 15);
set(gcf, 'color', 'none');
set(gca, 'color', 'none');
e = legend('x1 True','x1 UKF','x2 True', 'x2 UKF', 'x3
    True','x3 UKF','x4 True', 'x4 UKF')
e.FontSize = 10;
ylim([0 5]);
ylabel('x_4', 'FontSize', 15);
xlabel('Time [s]', 'FontSize', 15);

%saveas(gcf,'\Users\lindseytam\Desktop\thesis\
    Meskin_overall_badIntial.png')
saveas(gcf,'\Users\lindseytam\Desktop\thesis\
    Meskin_overall.png')
```

B.2 State function

```
function x = MeskinStateFcn(x)
% Trying Meskin in two dimensions
dt = 0.05;
t = 0; % dummy time variable for MeskinODE
x = x + MeskinODE(t, x)*dt;
end

%{
function dxdt = MeskinStateFcnContinuous(x)
parameter_values;

dxdt = [a_1 * x(3)^g_13 - b_1 * x(1)^h_11;
        a_2 * x(1)^g_21 - b_2 * x(2)^h_22;
        a_3 * x(2)^g_32 - b_3 * x(3)^h_33 * x(4)^h_34;
        a_4 * x(1)^g_41 - b_4 * x(4)^h_44];
end
%}
```

B.3 Measurement function

```
function yk = MeskinMeasurementNonAddFcn(xk,vk)
% Author: Ltam
% Date: November 17, 2019
% Summary:
% Inputs:    xk = states at time k, x[k]
%            vk = measurement noise vector at time, k v[k]
% Outputs:   yk = measurements at time k
%
% The measurement is the first state with multiplicative
    noise
yk = xk(1)+vk;
%yk =xk*(1+vk);
end
```

B.4 Parameter Values

```
%This script saves the baseline parameter values for our
    model

a_1 = 20;
a_2 = 8;
a_3 = 3;
a_4 = 2;

b_1 = 10;
b_2 = 3;
b_3 = 5;
b_4 = 6;

g_11 = 0;
g_21 = 0.5;
g_31 = 0;
g_41 = 0.5;

g_12 = 0;
g_22 = 0;
g_32 = 0.75;
g_42 = 0;

g_13 = -0.8;
g_23 = 0;
g_33 = 0;
g_43 = 0;
```

```
27
28  g_14 = 0;
29  g_24 = 0;
30  g_34 = 0;
31  g_44 = 0;
32
33  h_11 = 0.5;
34  h_21 = 0;
35  h_31 = 0;
36  h_41 = 0;
37
38  h_12 = 0;
39  h_22 = 0.75;
40  h_32 = 0;
41  h_42 = 0;
42
43  h_13 = 0;
44  h_23 = 0;
45  h_33 = 0.5;
46  h_43 = 0;
47
48  h_14 = 0;
49  h_24 = 0;
50  h_34 = 0.2;
51  h_44 = 0.8;
```