ASEN 5044 Final Project: Orbit Determination

Ben Gavin, Joaquin Ramirez, Nathan Zhao

December 2022

1 0. Contributions

Member	**Contributions**
Nathan Zhao	Jacobian derivations, Model linearization, Linear Simulation,Document, EKF, TMT, observation data estimate, Document preparation
Ben Gavin	Jacobian derivations, Model linearization, Nonlinear simulation, LKF, EKF, TMT, observation data estimate, Document preparation
Levi Ramirez	EKF and LKF design. (plotting, simulation,etc). Overleaf organization and explanation of results

December 13 2022

College of Engineering & Applied Science
UNIVERSITY OF COLORADO **BOULDER**

Contents

2 Introduction

The task at hand was to estimate the position of spacecrafts such as TOPEX-POSEIDON, Jason I, II, or III with as relatively high degree of accuracy. In order to do so, the dynamical system was represented in an Earth-centered x-y coordinate frame. The equations of motion were given as

$$\begin{aligned} \ddot{X} &= -\frac{\mu X}{r^3} + u_1 + \tilde{w}_1 \\ \ddot{Y} &= -\frac{\mu Y}{r^3} + u_2 + \tilde{w}_2 \end{aligned} \tag{1}$$

where μ is the gravitational parameter, $r = \sqrt{X^2 + Y^2}$, u are inputs into the system, and $\tilde{w}$ is white noise in the system. The system can be represented by

$$\begin{aligned} \boldsymbol{x}(t) &= [X, \dot{X}, Y, \dot{Y}]^T \\ \boldsymbol{u}(t) &= [u_1, u_2]^T \\ \tilde{\boldsymbol{w}}(t) &= [\tilde{w}_1, \tilde{w}_2]^T \end{aligned} \tag{2}$$

The nonlinear outputs of the system can them be represented by

$$\boldsymbol{y}^i(t) = \begin{bmatrix} \rho^i(t) \\ \dot{\rho}^i(t) \\ \phi^i(t) \end{bmatrix} + \tilde{\boldsymbol{v}}^{\boldsymbol{i}}(t) \tag{3}$$

where $\tilde{\boldsymbol{v}}^{\boldsymbol{i}}(t)$ represents measurement error, i refers to one of twelve ground systems with postions $X^i(t), Y^i(t)$ and $\rho^i(t), \dot{\rho}^i(t), \phi^i(t)$ represent relative range, range rate, and elevation angel respectively. These can be represented mathematically by

$$\begin{aligned} \rho^i(t) &= \sqrt{\left(X(t) - X_s^i(t)\right)^2 + \left(Y(t) - Y_s^i(t)\right)^2} \\ \dot{\rho}^i(t) &= \frac{\left[X(t) - X_s^i(t)\right] \cdot \left[\dot{X}(t) - \dot{X}_s^i(t)\right] + \left[Y(t) - Y_s^i(t)\right] \cdot \left[\dot{Y}(t) - \dot{Y}_s^i(t)\right]}{\rho^i(t)} \\ \phi^i(t) &= \tan^{-1}\left(\frac{Y(t) - Y_s^i(t)}{X(t) - X_s^i(t)}\right). \end{aligned} \tag{4}$$

The position of the ground stations can be estimated as

$$\begin{aligned} X^i(t) &= R_E \cos\left(\omega_E t + \theta^i(0)\right), \\ Y^i(t) &= R_E \sin\left(\omega_E t + \theta^i(0)\right) \end{aligned} \tag{5}$$

With this now set, the system can be examined with several different methods to create a proper state estimation.

3 Part I DETERMINISTIC SYSTEM ANALYSIS

3.1 Problem 1

With the system defined, the first step for analysis is to linearize it. This is done by calculating the CT jacobians of the system. The system was reorganized into the form of

$$\mathbf{f}(x) = \dot{\boldsymbol{x}} = \begin{bmatrix} x_2 \\ -\frac{\mu x_1}{\left(x_1^2+x_3^2\right)^{3/2}} + u_1 + \tilde{w}_1 \\ -\frac{\mu x_3}{\left(x_1^2+x_3^2\right)^{3/2}} + u_2 + \tilde{w}_2 \end{bmatrix}$$

$$\mathbf{h}(x) = \begin{bmatrix} \sqrt{\left(X(\iota) - X_s^i(l)\right)^2 + \left(Y(\iota) - Y_s^i(t)\right)^2} \\ \frac{\left[X(\iota)-X_s^i(\iota)\right]\cdot\left[\dot{X}(\iota)-\dot{X}_s^i(\iota)\right]+\left[Y(\iota)-Y_s^i(\iota)\right]\cdot\left[\dot{Y}(\iota)-\dot{Y}_s^i(\iota)\right]}{\rho^i(\iota)} \\ \tan^{-1}\left(\frac{Y(\iota)-Y_s^i(l)}{X(\iota)-X_s^i(\iota)}\right) \end{bmatrix} \tag{6}$$

From here, the jacobians are calculated as

$$\begin{aligned} \tilde{A} &= \left.\frac{\partial f}{\partial x}\right|_{nom} \\ \tilde{B} &= \left.\frac{\partial f}{\partial u}\right|_{nom} \\ \tilde{\Gamma} &= \left.\frac{\partial f}{\partial w}\right|_{nom} \\ \tilde{H} &= \left.\frac{\partial h}{\partial x}\right|_{nom} \end{aligned} \tag{7}$$

Evaluating these partial derivatives yields the following results with the work found in the appendix.

$$\tilde{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{\mu(2x_1^2-x_3^2}{\left(x_1^2+x_3^2\right)^{5/2}} & 0 & \frac{3\mu x_1 x_3}{\left(x_1^2+x_3^2\right)^{5/2}} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{3\mu x_1 x_3}{\left(x_1^2+x_3^2\right)^{5/2}} & 0 & \frac{\mu(2x_3^2-x_1^2}{\left(x_1^2+x_3^2\right)^{5/2}} & 0 \end{bmatrix} \tag{8}$$

$$\tilde{B} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{9}$$

$$\tilde{\Gamma} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{10}$$

$$\tilde{H} = \begin{bmatrix} \frac{x_1 - X^i}{\rho} & 0 & \frac{x_3 - Y^i}{\rho(t)} & 0 \\ \frac{x_2 - \dot{X}^i}{\rho} + (X^i - x_1)\dot{\rho} & \frac{x_1 - X^i}{\rho} & \frac{x_4 - \dot{Y}^i}{\rho} + (Y^i - x_3)\dot{\rho} & \frac{x_3 - Y^i}{\rho} \\ \frac{Y^i - x3}{(x_1 - X^i)^2 + (x_3 - Y^i)^2} & 0 & \frac{x1 - X^i}{(x_1 - X^i)^2 + (x_3 - Y^i)^2} & 0 \end{bmatrix} \tag{11}$$

3.2 Problem 2

With the jacobians now obtained, the system can be linearized around an equilibrium point. The nominal operation point is defined by the following:

$$\begin{aligned} X(0) &= 6678 \\ \dot{X}(0) &= 0 \\ Y(0) &= 0 \\ \dot{Y}(0) &= r_0 \cdot \sqrt{\frac{\mu}{r_0^3}} \text{ km/s} \end{aligned} \tag{12}$$

The conversion from the CT nonlinear model to a DT linearized model is illustrated below:

$$\begin{aligned} \tilde{F} &= I + \Delta t \cdot \tilde{A} \\ \tilde{G} &= \Delta t \cdot \tilde{B} \\ \tilde{\Omega} &= \Delta t \cdot \tilde{\Gamma} \end{aligned} \tag{13}$$

To linearize about an equilibrium point, the conditions in equation 12 were used to develop a basic trigonometric model of a circular orbit to describe x_1 and x_3, with its derivative to describe x_2 and x_4. Then, these were plugged into the $\tilde{A}$ to calculate the DT linear matrix representation of the spacecraft model, $\tilde{F}$. $\tilde{F}$ was then multiplied by the perturbations to get the set of perturbations for the next iteration. The perturbations of each state were then added to the nominal states described by the trigonometric model to update the state. For the outputs of the system, the nominal states were plugged into the $\tilde{H}$, and then similarly multiplied by the perturbations to recieve the output perturbations. This was added to the nominal state calculated by $\rho(t)$, $\dot{\rho(t)}$, and $\phi(t)$ equations to obtain the output states. The results are as follows. Throughout this process, however, it was noticed that the states were not constant over time, meaning that the system was time-varying, meaning there is no observatory or stability matrix.

3.3 Problem 3

The linearized results were obtained using method outlined in the previous section. An initial perturbation of $[0, 0.075, 0, -0.021]^T$ was used with the results as follows.

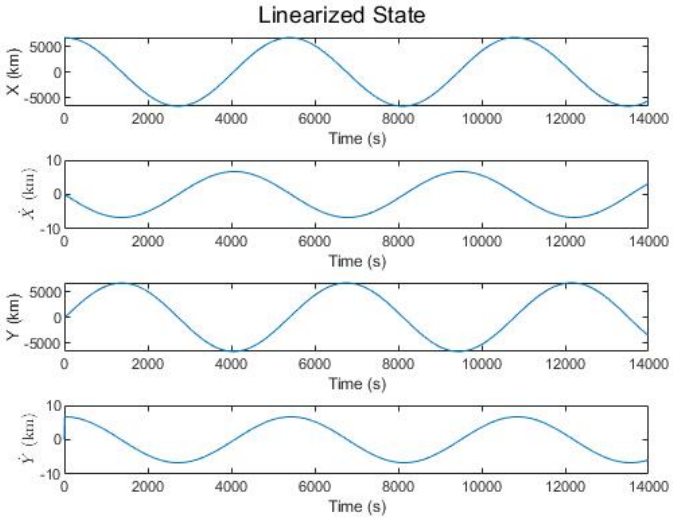

Figure 1: Linearized State

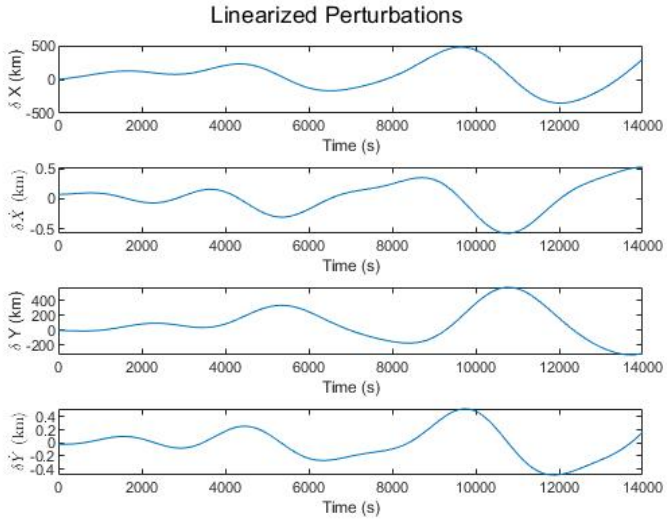

Figure 2: Linearized Perturbations

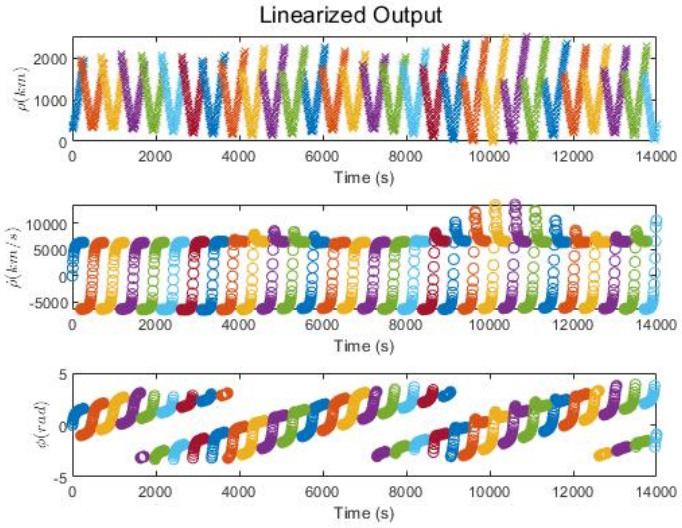

Figure 3: Linearized Outputs

For a nonlinear system, instead of using the linearized dynamics from earlier, ODE45 was used to integrate though the equations of motions to create a nominal state. With this, there is no longer a track of perturbations, as instead of adding a change in state for the next time step, the ODE45 integrates to find the next time step. The results for this are shown below.

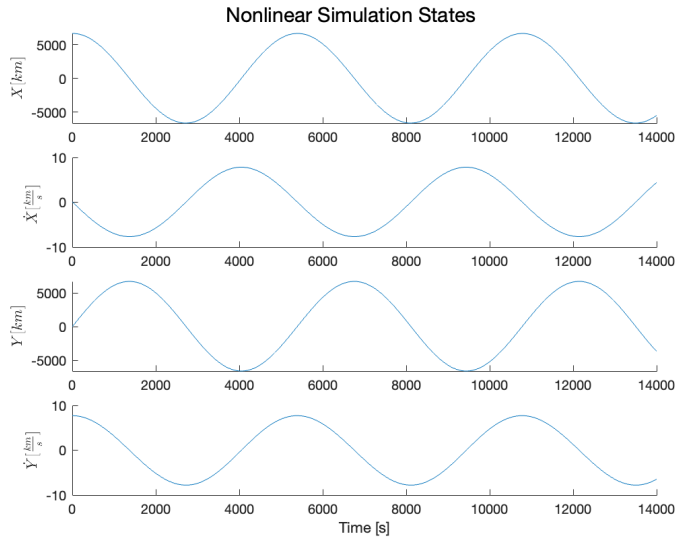

Figure 4: Nonlinearized State

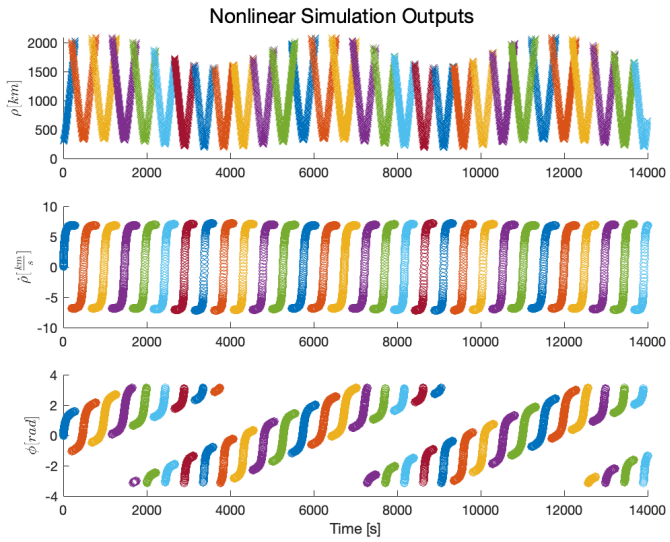

Figure 5: Nonlinearized Outputs

Comparing the states of the two methods, it can be seen as

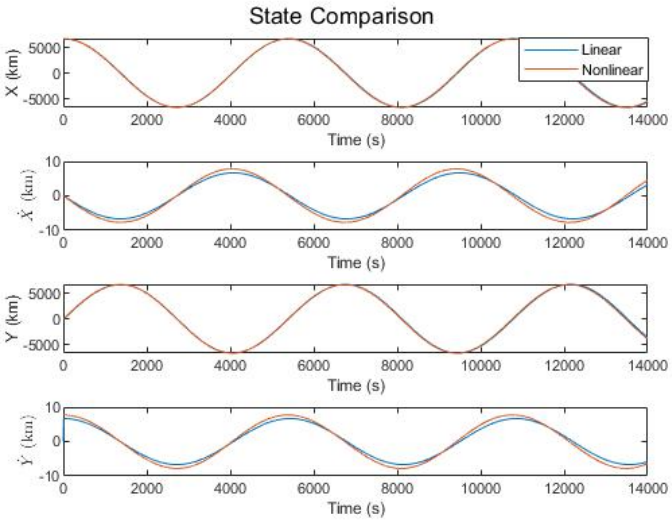

Figure 6: Method State Comparison

The positions seem to follow closely, but begin to diverge slightly as time increases. The velocity, however, doesn't start off matching and also demonstrates a further diverging as time increases. These results are expected, as with linearization there becomes a loss in accuracy. Looking closer at

the outputs based on the two different systems, the nonlinearized model has the output performing at a consistent periodic fashion. In comparison the linearized model is less consistent, having results such as the ρ increasing in amplitude over time or $\dot{rho}$ experiencing sudden jumps at later times for certain stations, again reinforcing the idea that the linearized model becomes less accurate as time increases.

4 Part II STOCHASTIC NONLINEAR FILTERING

4.1 Problem 4

When implementing the linearized Kalman filter, we first start with the values derived in problem 2. From this, we then set a P0 matrix as well as truth values for Q and R. Now the time can be iterated through and follow a similar process in problem 3. The F matrix is obtained from the state to obtain the next perturbation. That F matrix is also used to obtain

$$P_{k+1}^{-} = F_k P_k^{+} F_k^T + \Omega_k Q_k \Omega_k^T \tag{14}$$

Where P_k^+ and Q_k are initialized with the value calculated from before. This is then used to update the state until the time has been fully iterated through. Below are the given results.

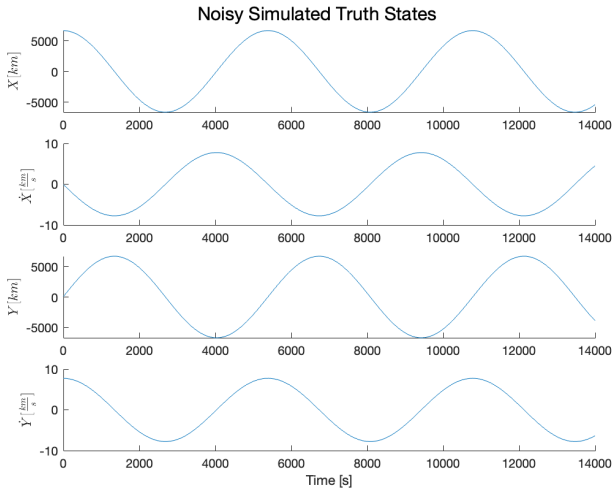

Figure 7: Simulated Noisy States

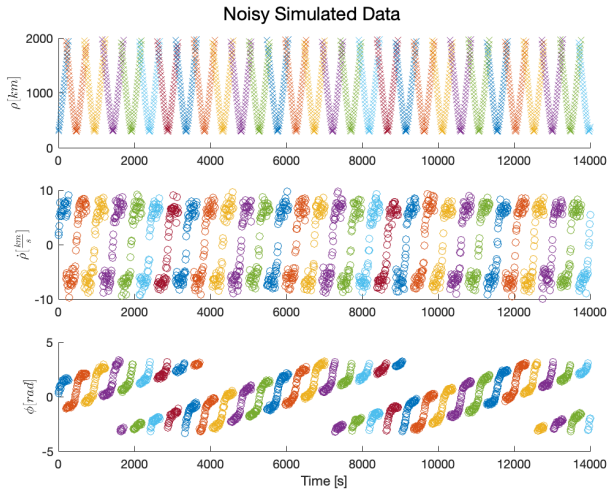

Figure 8: Simulated Noisy Data

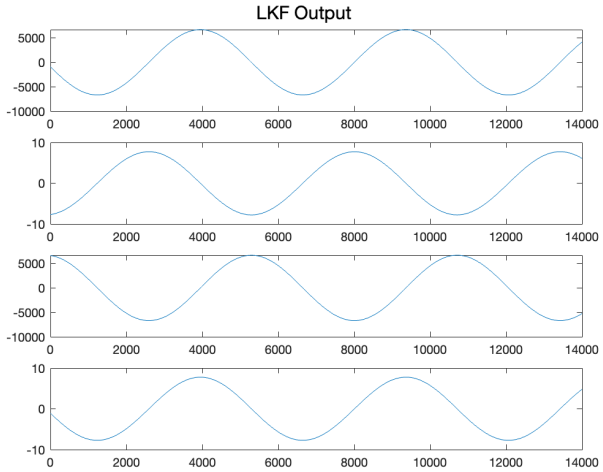

Figure 9: LKF State Outputs

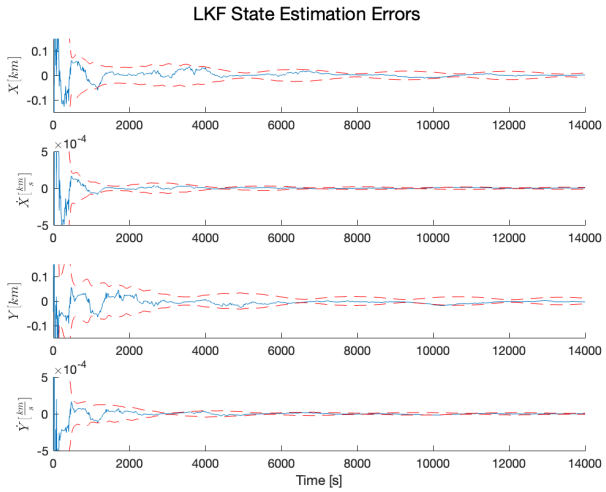

Figure 10: LKF State Errors

This filter was initialized with the following:

$$\hat{\delta x}^{+}(0) = [0, 0, 0, 0]^T \tag{15}$$

$$P^{+}(0) = \begin{bmatrix} 0.001 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0.001 \end{bmatrix} \tag{16}$$

After results were obtained, we conducted truth model testing via NEES and NIS to check whether the state and measurement errors are consistent with the system's noise models. The NEES test characterizes the total state errors between the truth data and the output of the LKF. Truth model testing was conducted with a Monte-Carlo simulation run 10 times in which the ground truth trajectory was simulated many times and filtered through the linear Kalman filter. This metric is calculated at each time step using the following equation.

$$\epsilon_{x,k} = e_{x,k}^T (P_k^+)^{-1} e_{x,k} \tag{17}$$

where

$$e_{x,k} = x_k - \hat{x}_k^+ \tag{18}$$

To determine whether these computed values were consistent with the chi-squared distribution for a system with 4 degrees of freedom, they were plotted along with measurement bounds. These bounds were calculated using MATLAB's built in chi2inv function as follows.

$$r1 = chi2inv(\frac{\alpha}{2}, Nn)/N \tag{19}$$

$$r2 = chi2inv(1 - \frac{\alpha}{2}, Nn)/N \tag{20}$$

where α was chosen to be 0.05 so that 5% of the measurements would fall outside of the bounds, n was set to 4: the number of degrees of freedom, and N was equal to the number of Monte-Carlo runs. In order to fit the NEES values withing the bounds, the provided Q matrix had to be scaled by a factor of 10^{-12}. The simulations with the unscaled Q matrix yielded values below the calculated bounds which implied that the variances in the Q matrix were too pesimistic and needed to be reduced. The NEES results with the tuned Q matrix can be seen below.

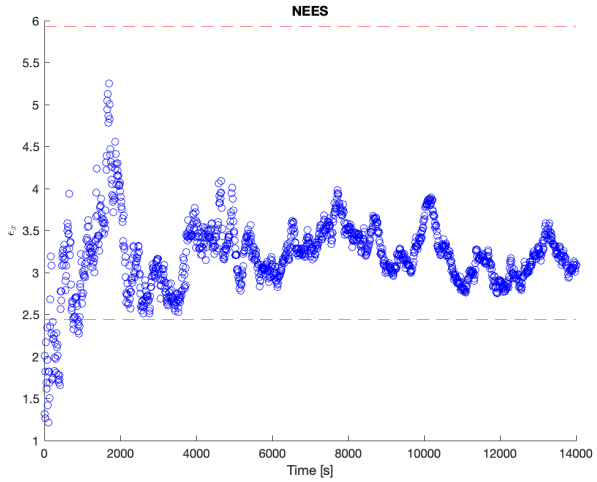

Figure 11: LKF NEES

The NIS test characterizes the errors between the truth data and the expected data given the estimated state. This metric is calculated using the following equation.

$$\epsilon_{y,k} = e_{y,k}^T (S_k^+)^{-1} e_{y,k} \tag{21}$$

where

$$e_{y,k} = y_k - \hat{y}_k^- \tag{22}$$

$$S_k = (H_{k+1} P_{k+1}^- H_{k+1}^T + R)^{-1} \tag{23}$$

To determine whether these computed values were consistent with the chi-squared distribution for a system with 3 degrees of freedom, they were plotted along with measurement bounds. These bounds were calculated similarly to the NEES test.

$$r1 = chi2inv(\frac{\alpha}{2}, Np)/N \tag{24}$$

$$r2 = chi2inv(1 - \frac{\alpha}{2}, Np)/N \tag{25}$$

where α was chosen to be 0.05 so that 5% of the measurements would fall outside of the bounds, p was set to 3: the number of degrees of freedom, and N was equal to the number of Monte-Carlo runs. No tuning was required to fit the NIS results within these calculated bounds. The NIS results can be seen below.

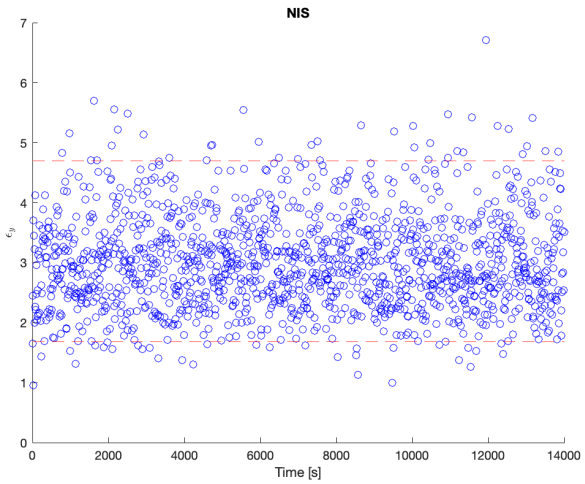

Figure 12: LKF NIS

4.2 Problem 5

Moving on to an Extend Kalman Filter, instead of linearzing around the nominal state, the EKF utilizes the nonlinear dynamics and linearizes around the current estimate. This linearization around the current state allows it to make a small adjustment for perturbations before moving on to the next state and restarting the process again. This prevents an issue of inaccuracies compounding as time increases. To implement this, an ODE45 at each time step was conducted to find the state which will be represented as $\hat{x}^-_{k+1}$. This $\hat{x}^-_{k+1}$ was then used to calculate a new F matrix which is used in Equation 14 and a new H matrix. That P and H matrix are then used to calculate a new gain matrix

$$\tilde{K}_{k+1} = P^-_{k+1}\tilde{H}^T_{k+1}\left[\tilde{H}_{k+1}P^-_{k+1}\tilde{H}^T_{k+1} + R_{k+1}\right]^{-1} \tag{26}$$

where R_{k+1} was treated as the provided truth value for R. This new gain matrix can then be used to obtain the next state and P matrix using the following equations.

$$\begin{aligned} \hat{x}^+_{k+1} &= \hat{x}^-_{k+1} + \tilde{K}_{k+1}\tilde{e}_{y_{k+1}} \\ P^+_{k+1} &= \left(I - \tilde{K}_{k+1}\tilde{H}_{k+1}\right)P^-_{k+1} \end{aligned} \tag{27}$$

where $\tilde{e}_{y_{k+1}}$ is the difference between the nominal data and estimated data from calculated using the new H matrix. The process can now start again with the new state and P matrix. To see the results of the EKF, the same noisy state and data from Figures 7 and 8 was used as the nominal

solution. The inital P matrix used was

$$P^{+}(0) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.001 \end{bmatrix} \tag{28}$$

and Q was initially set to the provided truth Q. The results from the EKF are as follows

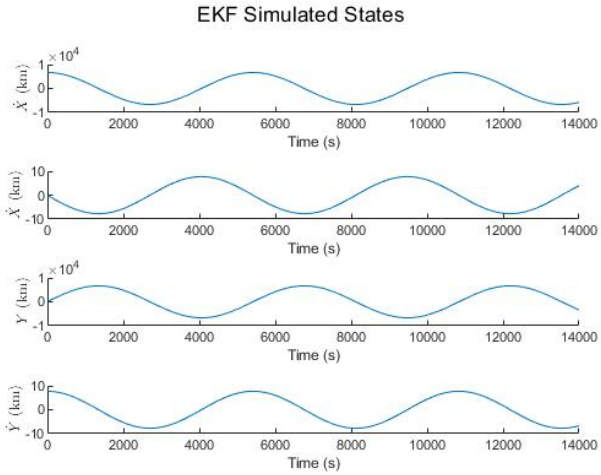

Figure 13: EKF State Outputs

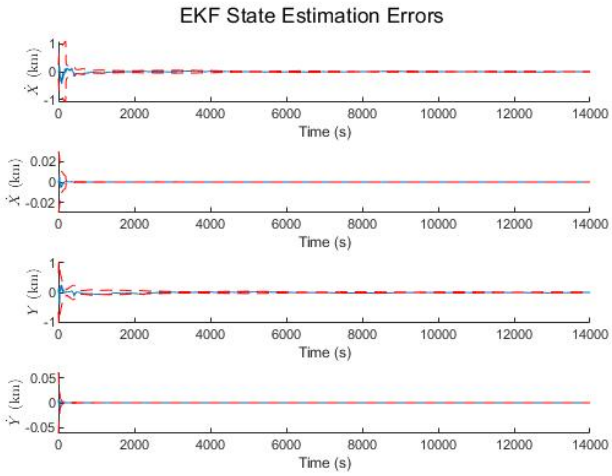

Figure 14: EKF State Errors

Comparing the EKF errors to the LKF errors, it can be seen that the EKF performs much better than the LKF at state estimation. There is a similar trend of there being a high amount of initial error relative to later on, but this decreases much faster than the LKF. The overall values of error also tend to be much lower than the LKF, being about a magnitude lower, furthering the idea of that an EKF is much better than an LKF.

Similarly to the LKF, an NEES and NIS were conducted to evaluate the validity of the measurement errors. The exact same method was followed as outlined in Problem 5, where the Q matrix was adjusted such that 95% of the data would fall within the bounds. The same parameters of 10 Monte-Carlo runs, $\alpha = 0.05$, and n being 4 and 3 in the resepective test. The final Q value ended up being 10^{-6} multiplied by the provided Q matrix. The results from the NEES and NIS are as follows.

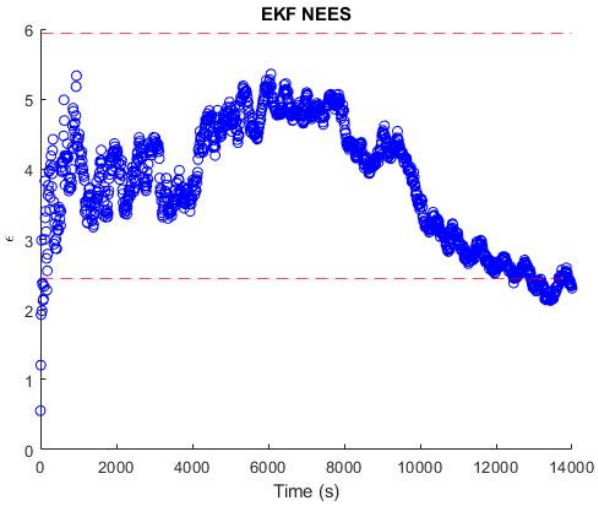

Figure 15: EKF NEES

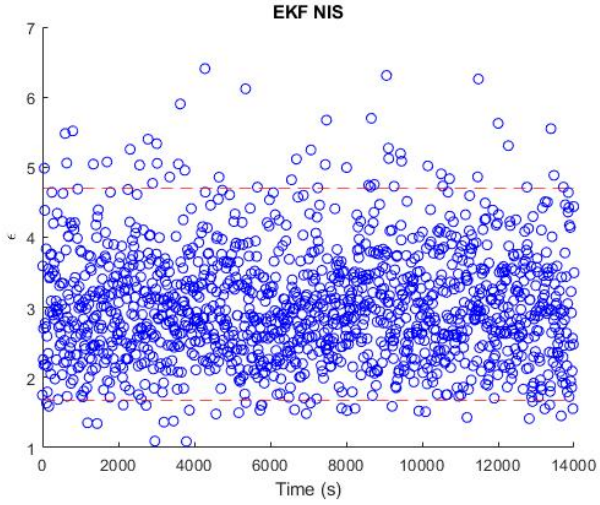

Figure 16: EKF NIS

4.3 Problem 6

Up to this point, the LKF and EKF were implemented using a nominal trajectory obtained from the provided dynamics. To be fully evaluate the performance of both filters, they need to be implemented with a true trajectory instead. A provided data set for y was provided and used inside the filters. The results are shown below.

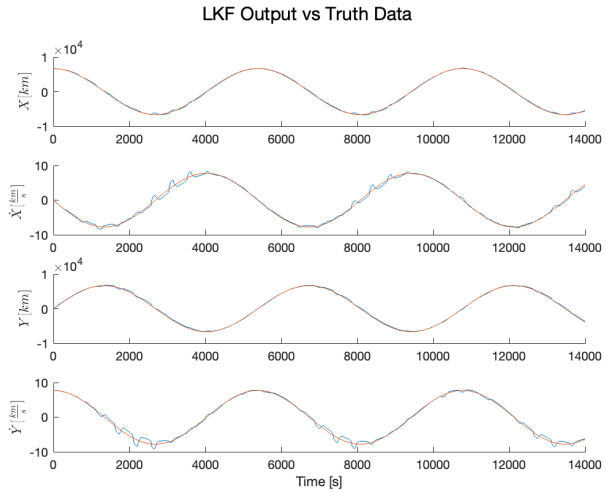

Figure 17: LKF Output vs Truth Data

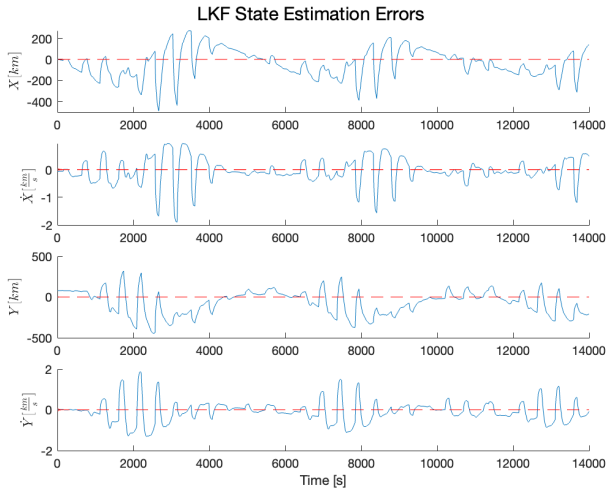

Figure 18: LKF Error Using Data

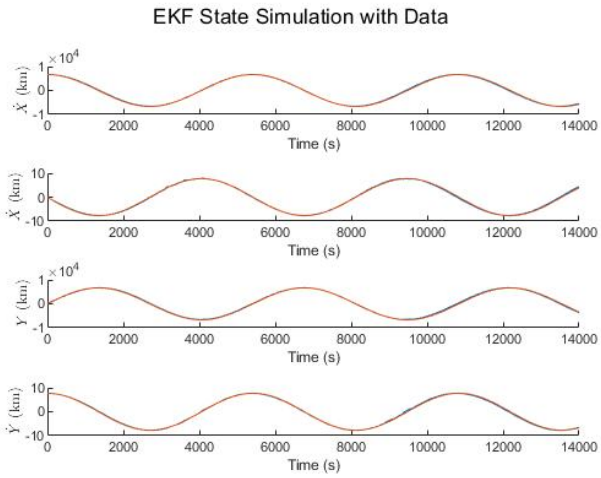

Figure 19: EKF State Comparison with Data

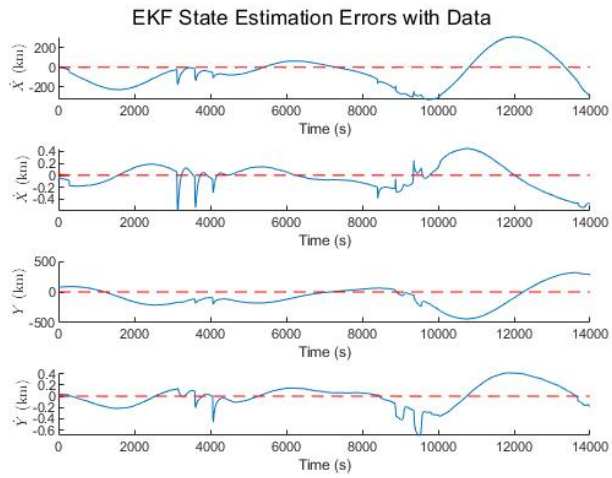

Figure 20: EKF Error Using Data

Upon initial inspection both filters have issues estimating the states. Looking closer at the errors, however, the LKF demonstrates a periodic error, while the EKF starts off with relatively low errors and increases as time increases. The LKF also demonstrates an overall higher magnitude of errors when compared to the EKF. This leads to the idea that the EKF is better at state estimation, which falls in line with conclusions derived in Problems 4 and 5. As a result of the errors, the LKF estimates a much more jagged state compared to the smooth nature of the EKF. Within the provided y data, there were empty cells for certain timesteps. In order to counteract this, the filters kept track of the most recent time step with data, and used that in the estimation. The jumps in the LKF states and the errors in its measurement suggests that it had more trouble performing estimation with missing states compared to the EKF, once again ensuring the idea that the EKF does a "better" job at estimating states.

Even though the EKF does a better job at estimating, it is important to note that the filters are still not perfect at reading the data. This can come from a multitude of factors. It is possible that there are unaccounted for properties of the system that would affect the calculations of x and y. The initial definition of the system does assume an ideal orbit. It is also possible that there an error in the collection of the y data that would lead to a faulty estimation, leading to more error when calculating the states.

5 Advanced Questions

5.1 Haiku

Tried to shoot the moon
Missed by a few light-seconds
Bad sensor readings

5.2 Unscented Kalman Filter

An unscented Kalman filter is a type of algorithm that can be used to estimate the state of a system from noisy measurements. At a high level, an unscented kalman filter works by using a set of carefully chosen "sigma points" to approximate the distribution of the state of the system at any given time. These sigma points are chosen so that they spread out over the space of possible states in a way that captures the shape of the distribution as accurately as possible.

Once the sigma points have been chosen, the unscented kalman filter uses them to compute the mean and covariance of the state distribution at each time step. This information is then used to predict the state of the system at the next time step, and updates the estimated state based on new measurements.

One key advantage of the unscented kalman filter is that it is able to accurately capture the shape of the state distribution, even when that distribution is highly non-Gaussian (i.e., not well modeled by a normal distribution). This makes it more effective than other types of Kalman filter in many real-world scenarios.

The results of the unscented kalman filter developed from the class lecture slides can be seen in the plot below. This plot compares the unscented kalman filter state against the truth state calculated in part 1 of the project. In addition to this, the error plot was included below to observe the differences in magnitudes seen between each state.

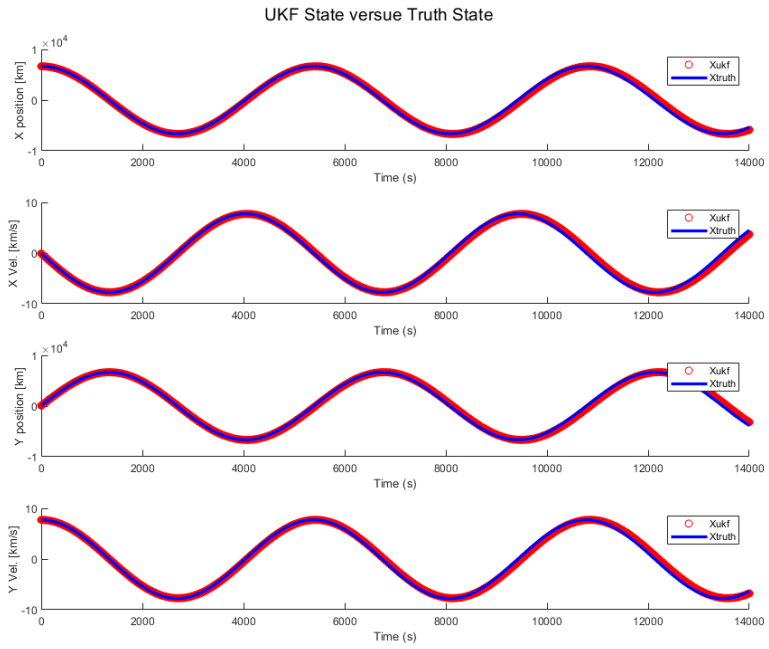

Figure 21: UKF State Estimation Using Noisy Nonlinear Dynamical Model Inputs

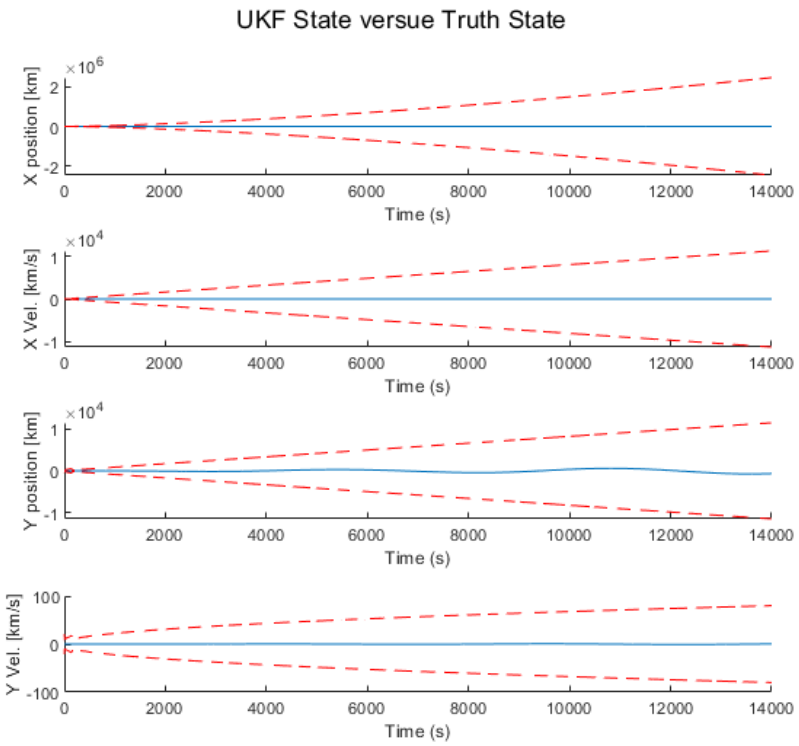

Figure 22: UKF Error Using Noisy Nonlinear Dynamical Model Inputs

Unfortunately, the unscented Kalman Filter design gains could not be tuned with the actual measurement data, but a similar process to tune the UKF would be used as is described in problem 4. In addition to the gain values, there are also other parameters that can be tuned in an unscented kalman filter, such as the sigma point weights and the covariance matrices. These parameters can be tuned in a similar way, using either a trial-and-error approach.

6 Appendix

6.1 CT Jacobian Derivation

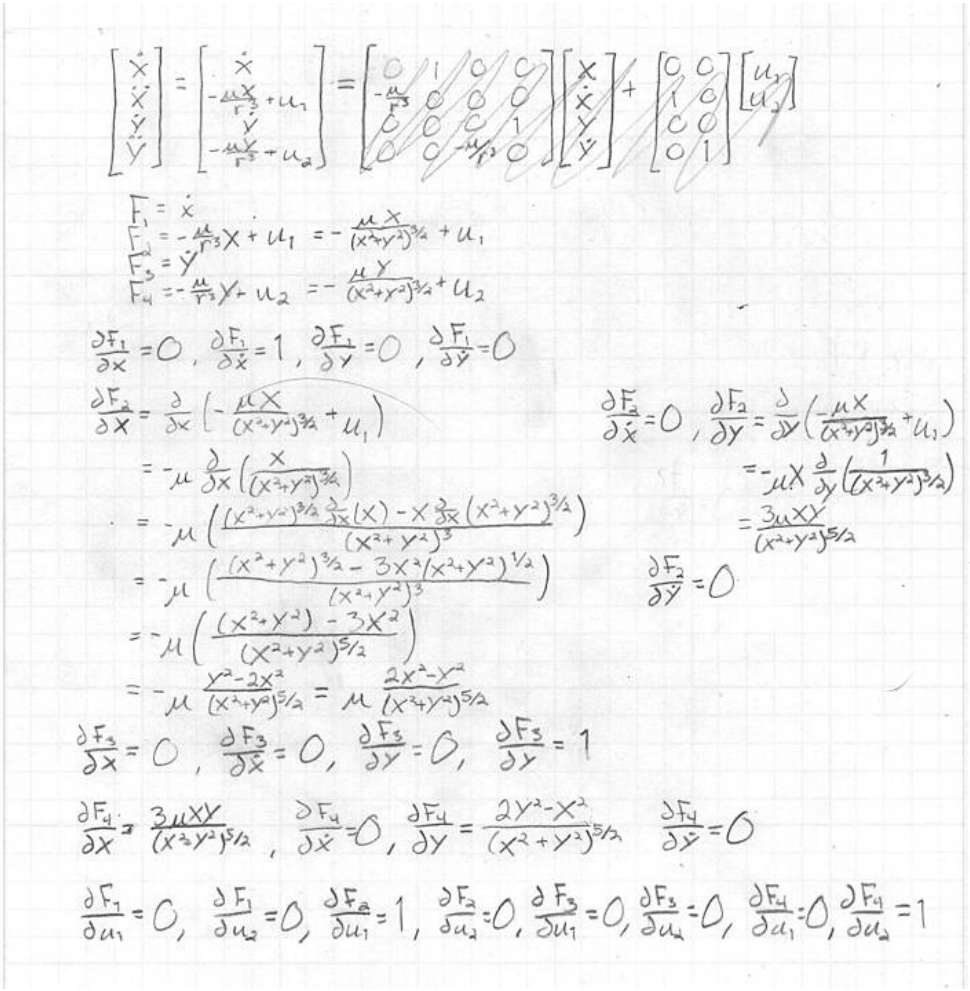

Figure 23: Derivation of the jacobian for the f(x) state matrix.

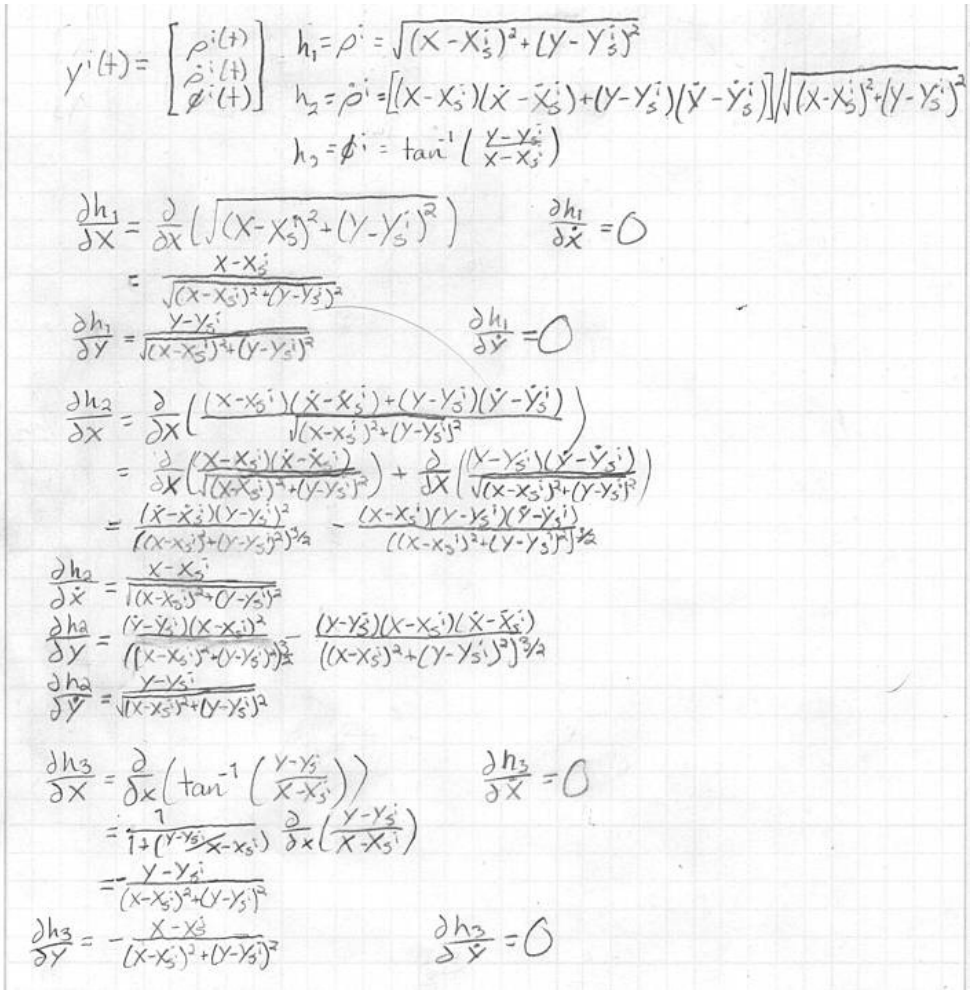

Figure 24: Derivation of the jacobian for the h(x) output state matrix

6.2 Code

```
%Housekeeping
clear
clc
close all
%% Part 1
rng(100)
load('orbitdeterm_finalproj_KFdata.mat')
rE = 6378;
mu = 4*10^5;
omega_e = 2*pi/86400;
```

```
11
12 perturb_x0 = [0,0.075,0,-0.021]';
13 x0 = [6678, 0, 0, 6678 * sqrt(mu/(6678^3))]';
14
15 B = [0  0;
16      1  0;
17      0  0;
18      0  1];
19 Gamma = [0  0;
20           1  0;
21           0  0;
22           0  1];
23 delT = 10;
24
25 x = x0+perturb_x0;
26
27 P = 2*pi*sqrt(6678^3/mu);
28 figure
29 for j = 1:12
30     theta0 = (j-1)*pi/6;
31     y = [];
32     for i = 1:length(tvec)
33         x_state = x(:,i);
34         x1 = 6678*cos(2*pi/P * tvec(i));
35         x2 = -6678*sin(2*pi/P * tvec(i));
36         x3 = 6678*sin(2*pi/P * tvec(i));
37         x4 = 6678*cos(2*pi/P * tvec(i));
38         [F,G] = linearizeMat([x1 x2 x3 x4]', delT);
39
40         perturb_x0_new = F*perturb_x0(:,i);
41 %       x_state(1) = x1;
42 %       x_state(3) = x3;
43         x_state_new = [x1 x2 x3 x4]' + perturb_x0_new;
44         x = [x x_state_new];
45
46 %         x1 = x_state(1);
47 %         x2 = x_state(2);
48 %         x3 = x_state(3);
49 %         x4 = x_state(4);
50
51         Xi = rE * cos(omega_e * tvec(i) + theta0);
52         Yi = rE * sin(omega_e * tvec(i) + theta0);
53         Xi_dot = -rE * omega_e * sin(omega_e * tvec(i) + theta0);
54         Yi_dot = rE * omega_e * cos(omega_e * tvec(i) + theta0);
55
56         rho = sqrt((x1 - Xi)^2 + (x3 - Yi)^2);
```

```
57          rho_dot = ((x1 - Xi)*(x2 - Xi_dot) + (x3 - Yi)*(x4 - Yi_dot)) /
                rho;
58          phi = atan2((x3 - Yi),(x1 - Xi));
59
60          H = [(x1-Xi)/rho 0 (x3-Yi)/rho 0;
61               (x2-Xi_dot)/rho+(Xi-x1)*rho_dot/rho^2 (x1-Xi)/rho (x4-
                    Yi_dot)/rho+(Yi-x3)*rho_dot/rho^2 (x3-Yi)/rho;
62               (Yi-x3)/((x1-Xi)^2 + (x3-Yi)^2) 0 (x1-Xi)/((x1-Xi)^2 + (x3-
                    Yi)^2) 0];
63          perturb_y = H*perturb_x0(:,i);
64          ystate = [rho rho_dot phi]' + perturb_y;
65          y = [y ystate];
66
67          perturb_x0 = [perturb_x0 perturb_x0_new];
68
69          thetai = atan2(Yi,Xi);
70          if abs(angdiff(thetai,phi)) > pi/2
71             y(1,i) = nan;
72             y(2,i) = nan;
73             y(3,i) = nan;
74         end
75      end
76      subplot(3,1,1)
77      plot(tvec,y(1,:),'o')
78      hold on
79      xlabel('Time (s)')
80      ylabel('$\rho (km)$','Interpreter','latex')
81      subplot(3,1,2)
82      plot(tvec,y(2,:),'o')
83      hold on
84      xlabel('Time (s)')
85      ylabel('$\dot{\rho} (km)$','Interpreter','latex')
86      subplot(3,1,3)
87      plot(tvec,y(3,:),'o')
88      hold on
89      xlabel('Time (s)')
90      ylabel('$\phi (rad)$','Interpreter','latex')
91      sgtitle('Linearized Output')
92  end
93
94
95  figure
96  subplot(4,1,1)
97  plot(tvec,x(1,1:length(tvec)))
98  xlabel('Time (s)')
99  ylabel('X (km)')
```

```
subplot(4,1,2)
plot(tvec,x(2,1:length(tvec)))
xlabel('Time (s)')
ylabel('$\dot{X}$ (km)','Interpreter','latex')
subplot(4,1,3)
plot(tvec,x(3,1:length(tvec)))
xlabel('Time (s)')
ylabel('Y (km)')
subplot(4,1,4)
plot(tvec,x(4,1:length(tvec)))
xlabel('Time (s)')
ylabel('$\dot{Y}$ (km)','Interpreter','latex')
sgtitle('Linearized States')

figure
subplot(4,1,1)
plot(tvec,perturb_x0(1,1:length(tvec)))
xlabel('Time (s)')
ylabel('\delta X (km)')
subplot(4,1,2)
plot(tvec,perturb_x0(2,1:length(tvec)))
xlabel('Time (s)')
ylabel('$\delta \dot{X}$ (km)','Interpreter','latex')
subplot(4,1,3)
plot(tvec,perturb_x0(3,1:length(tvec)))
xlabel('Time (s)')
ylabel('\delta Y (km)')
subplot(4,1,4)
plot(tvec,perturb_x0(4,1:length(tvec)))
xlabel('Time (s)')
ylabel('$\delta \dot{Y}$ (km)','Interpreter','latex')
sgtitle('Linearized Perturbations')

%% 3

dT = 10;

tspan = [0 1400*dT];

mu = 398600;

r0 = 6678;

x0 = [r0; 0; 0; r0 * sqrt(mu/r0^3)];

dx0 = [0;0.075;0;-0.021];
```

```
146
147 opts = odeset('reltol',1e-12,'abstol',1e-12);
148
149 [t,x] = ode45(@(t,x) NLODE(t,x,mu),tspan,x0+dx0,opts);
150
151 figure()
152 sgtitle('Nonlinear Simulation States')
153 subplot(4,1,1)
154 hold on
155 plot(t,x(:,1))
156 ylabel('$X [km]$','Interpreter','latex')
157 hold off
158 subplot(4,1,2)
159 hold on
160 plot(t,x(:,2))
161 ylabel('$\dot{X} [\frac{km}{s}]$ ','Interpreter','latex')
162 hold off
163 subplot(4,1,3)
164 hold on
165 plot(t,x(:,3))
166 ylabel('$Y [km]$','Interpreter','latex')
167 hold off
168 subplot(4,1,4)
169 hold on
170 plot(t,x(:,4))
171 xlabel('Time [s]')
172 ylabel('$\dot{Y} [\frac{km}{s}]$','Interpreter','latex')
173 hold off
174
175 tmax = length(t);
176
177 RE = 6378;
178
179 wE = 2*pi/86400;
180
181 for i = 1:12
182
183     thetas0(i) = (i-1)*pi/6;
184
185     for k = 1:tmax
186         tk = t(k);
187         Xs = RE*cos(wE*tk+thetas0(i));
188         Ys = RE*sin(wE*tk+thetas0(i));
189         Xsd = -RE*wE*sin(wE*tk+thetas0(i));
190         Ysd = RE*wE*cos(wE*tk+thetas0(i));
191         thetas(i,k) = atan2(Ys,Xs);
```

```
192             rho(i,k) = sqrt((x(k,1)-Xs)^2 + (x(k,3)-Ys)^2);
193             rhod(i,k) = ((x(k,1)-Xs)*(x(k,2)-Xsd)+(x(k,3)-Ys)*(x(k,4)-Ysd))
                    /rho(i,k);
194             phi(i,k) = atan2(x(k,3)-Ys,x(k,1)-Xs);
195
196             if abs(angdiff(thetas(i,k),phi(i,k))) > pi/2
197                 rho(i,k) = nan;
198                 rhod(i,k) = nan;
199                 phi(i,k) = nan;
200             end
201
202
203         end
204     end
205
206     figure()
207     sgtitle('Nonlinear Simulation Outputs')
208     subplot(3,1,1)
209     hold on
210     for i = 1:12
211         scatter(t,rho(i,:),'x')
212     end
213     ylabel('$\rho [km]$','Interpreter','latex')
214     hold off
215     subplot(3,1,2)
216     hold on
217     for i = 1:12
218         scatter(t,rhod(i,:))
219     end
220     ylabel('$\dot{\rho} [\frac{km}{s}]$','Interpreter','latex')
221     hold off
222     subplot(3,1,3)
223     hold on
224     for i = 1:12
225         scatter(t,phi(i,:))
226     end
227     ylabel('$\phi [rad]$','Interpreter','latex')
228     xlabel('Time [s]')
229     hold off
230
231
232     %% Part II STOCHASTIC NONLINEAR FILTER
233     %data vars: Qtrue, Rtrue, tvec, ydata
234
235     %Important Notes:
236     %No input u
```

```
237 dt = delT;
238
239 % Qk = [0.02 0; 0 0.02];
240 Qk = Qtrue;
241
242
243 Omk = dt*[0 0 ; 1 0; 0 0 ; 0 1];
244 Sw = chol(Qk,'lower');
245 Sv = chol(Rtrue,'lower');
246
247 % Monte Carlo Sim
248
249 N = 10;
250 n = 2;
251
252 alpha = 0.05;
253
254 for m = 1:N
255
256     x_k = [6678, 0, 0, 6678 * sqrt(mu/(6678^3))]';
257
258 %Noisy Data sim of x
259 for i = 1:length(tvec)
260
261     %Simulate truth
262     w_k = mvnrnd(zeros(2,1),Qk); % of Qtrue
263     t_k = tvec(i);
264
265     [~,x_nom] = ode45(@(t,x) odeLKF(t_k,x,[0;0]),[0 dt],x_k);
266
267     x_nom = x_nom';
268
269     x_k = x_nom(:,end);
270
271     xnomp(:,i) = x_nom(:,end);
272
273     w = Sw*randn(2,1);
274     x_t(:,i) = xnomp(:,i) + Omk*w; % Truth solution
275     %Simulate measurement truth
276     for j = 1:12
277         v = Sv*randn(3,1);
278         y_t(:,i,j) = dynamicsCalc(x_t(:,i),j,t_k)+v;
279         if any(~isnan(y_t(:,i,j)))
280             station(:,i,1) = j;
281         end
282     end
```

```
end

[x_lkf,P_lkf,NEES(m,:),NIS(m,:)] = LKF(x_t,y_t,station,tvec);

xdiff = x_t - x_lkf;

end

NEESmean = mean(NEES);
NISmean = mean(NIS);

%Plotting noiseless and noisey signal
figure
sgtitle('Noisy Simulated Truth States')
for i = 1:4
    subplot(4,1,i)
    hold on
    plot(tvec, x_t(i,:))
    hold off
end

figure()
sgtitle('Noisy Simulated Data')
subplot(3,1,1)
hold on
for i = 1:12
    scatter(tvec,y_t(1,:,i),'x')
end
ylabel('$\rho [km]$','Interpreter','latex')
hold off
subplot(3,1,2)
hold on
for i = 1:12
    scatter(tvec,y_t(2,:,i))
end
ylabel('$\dot{\rho} [\frac{km}{s}]$','Interpreter','latex')
hold off
subplot(3,1,3)
hold on
for i = 1:12
    scatter(tvec,y_t(3,:,i))
end
ylabel('$\phi [rad]$','Interpreter','latex')
xlabel('Time [s]')
hold off

```

```
329 figure
330 sgtitle('LKF State Estimation Errors')
331 for i = 1:4
332     subplot(4,1,i)
333     hold on
334     plot(tvec, xdiff(i,:))
335     sigma = 2*sqrt(P_lkf(i,i,:));
336     sigma = reshape(sigma,1,1401);
337     plot(tvec, sigma,'r--')
338     plot(tvec, -sigma,'r--')
339     hold off
340 end
341
342 n = 4;
343
344 r1 = chi2inv(alpha/2,N*n)/N;
345 r2 = chi2inv(1-alpha/2,N*n)/N;
346
347 figure
348 hold on
349 title('NEES')
350 yline(r1,'r--')
351 yline(r2,'r--')
352 scatter(tvec,NEESmean,'b')
353 hold off
354
355 n = 3;
356
357 r1 = chi2inv(alpha/2,N*n)/N;
358 r2 = chi2inv(1-alpha/2,N*n)/N;
359
360 figure
361 hold on
362 title('NIS')
363 yline(r1,'r--')
364 yline(r2,'r--')
365 scatter(tvec,NISmean,'b')
366 hold off
367
368 %% 6
369
370 [x_lkf,P_lkf,NEES,NIS] = LKFdata(x_t,station,tvec);
371
372 xdiff = x_t(:,2:1401) - x_lkf;
373
374
```

```
375 figure
376 sgtitle('LKF output')
377 for i = 1:4
378     subplot(4,1,i)
379     hold on
380     plot(tvec(2:1401), x_lkf(i,:))
381     hold off
382 end
383
384
385 figure
386 sgtitle('LKF State Estimation Errors')
387 for i = 1:4
388     subplot(4,1,i)
389     hold on
390     plot(tvec(2:1401), xdiff(i,:))
391     sigma = 2*sqrt(P_lkf(i,i,:));
392     sigma = reshape(sigma,1,1400);
393     plot(tvec(2:1401), sigma,'r--')
394     plot(tvec(2:1401), -sigma,'r--')
395     hold off
396 end
```

```
1  function [x_new,y_new] = LKF(F,H,x_nom0,dt)
2  %LKF Linearized Kalman Filter
3
4  %Calculate period for nominal state
5  mu = 4*10^5;
6  P = 2*pi*sqrt(6678^3/mu);
7
8  for i = 1:length(tvec)
9  %Recalc state nominal state solution for time step
10     x1 = 6678*cos(2*pi/P * tvec(i));
11     x2 = -6678*sin(2*pi/P * tvec(i));
12     x3 = 6678*sin(2*pi/P * tvec(i));
13     x4 = 6678*cos(2*pi/P * tvec(i));
14
15 %Find updated F matrix and H matrix
16
17
18
19 perturb_x0p1 = F*
20 x_new = x_nom0 + perturb_x0p1
21
22
23 end
```

```
24
25
26 end
```

```
1  % House keeping
2  clc
3  clear all
4  close all
5
6  %% Extended Kalman Filter
7  load('orbitdeterm_finalproj_KFdata.mat')
8  rE = 6378;
9  mu = 4*10^5;
10 omega_e = 2*pi/86400;
11
12 perturb_x0 = [0,0.075,0,-0.021]';
13 x0 = [6678, 0, 0, 6678 * sqrt(mu/(6678^3))]';
14
15 B = [0  0;
16      1  0;
17      0  0;
18      0  1];
19 Gamma = [0  0;
20           1  0;
21           0  0;
22           0  1];
23 delT = 10;
24
25 x = [];
26
27 P = 2*pi*sqrt(6678^3/mu);
28 y_tf = [];
29 y_tot = nan(3,length(tvec));
30 station = zeros(1,length(tvec));
31 x_tot = [];
32 for j = 1:12
33      theta0 = (j-1)*pi/6;
34      y = [];
35      x_tot = [x_tot; x];
36      x = x0+perturb_x0(:,1);
37      for i = 1:length(tvec)
38           x_state = x(:,i);
39           x1 = 6678*cos(2*pi/P * tvec(i));
40           x2 = -6678*sin(2*pi/P * tvec(i));
41           x3 = 6678*sin(2*pi/P * tvec(i));
42           x4 = 6678*cos(2*pi/P * tvec(i));
```

```
43          [F,G] = linearizeMat([x1 x2 x3 x4]', delT);
44
45          perturb_x0_new = F*perturb_x0(:,i);
46      %       x_state(1) = x1;
47      %       x_state(3) = x3;
48          x_state_new = [x1 x2 x3 x4]' + perturb_x0_new;
49          x = [x x_state_new];
50
51  %           x1 = x_state(1);
52  %           x2 = x_state(2);
53  %           x3 = x_state(3);
54  %           x4 = x_state(4);
55
56          Xi = rE * cos(omega_e * tvec(i) + theta0);
57          Yi = rE * sin(omega_e * tvec(i) + theta0);
58          Xi_dot = -rE * omega_e * sin(omega_e * tvec(i) + theta0);
59          Yi_dot = rE * omega_e * cos(omega_e * tvec(i) + theta0);
60
61          rho = sqrt((x1 - Xi)^2 + (x3 - Yi)^2);
62          rho_dot = ((x1 - Xi)*(x2 - Xi_dot) + (x3 - Yi)*(x4 - Yi_dot)) /
                rho;
63          phi = atan2((x3 - Yi),(x1 - Xi));
64
65          H = [(x1-Xi)/rho 0 (x3-Yi)/rho 0;
66              (x2-Xi_dot)/rho+(Xi-x1)*rho_dot/rho^2 (x1-Xi)/rho (x4-
                    Yi_dot)/rho+(Yi-x3)*rho_dot/rho^2 (x3-Yi)/rho;
67              (Yi-x3)/((x1-Xi)^2 + (x3-Yi)^2) 0 (x1-Xi)/((x1-Xi)^2 + (x3-
                    Yi)^2) 0];
68          perturb_y = H*perturb_x0(:,i);
69          ystate = [rho rho_dot phi]' + perturb_y;
70          y = [y ystate];
71
72          perturb_x0 = [perturb_x0 perturb_x0_new];
73
74          thetai = atan2(Yi,Xi);
75          if abs(angdiff(thetai,phi)) > pi/2
76             y(1,i) = nan;
77             y(2,i) = nan;
78             y(3,i) = nan;
79          else
80              station(i) = j;
81          end
82      end
83      repy = isnan(y_tot);
84      y_tot(repy) = y(repy);
85      y_tf(:,:,j) = y;
```

```
86  end
87
88  for i = 1:length(tvec)
89      [~,xode] = ode45(@(t,xODE) odeLKF(tvec(i),xODE,[0  0]),[0  10],x(:,i)
        );
90  end
91
92  P0 = diag([0.01, 0.001, 0.01, 0.001]);
93
94  Qk = 10^-7*Qtrue;
95  x_k = [6678, 0, 0, 6678 * sqrt(mu/(6678^3))]';
96  Omk = 10*[0 0 ; 1 0; 0 0 ; 0 1];
97  Sw = chol(Qk,'lower');
98  Sv = chol(Rtrue,'lower');
99  % Monte Carlo Sim
100
101 N = 10;
102 n = 2;
103
104 alpha = 0.05;
105
106 for m = 1:N
107
108     x_k = [6678, 0, 0, 6678 * sqrt(mu/(6678^3))]';
109
110 for i = 1:length(tvec)
111
112     %Simulate truth
113     w_k = mvnrnd(zeros(2,1),Qk); % of Qtrue
114     t_k = tvec(i);
115
116     [~,x_nom] = ode45(@(t,x) odeLKF(t_k,x,[0;0]),[0  10],x_k);
117
118     x_k = x_nom(end,:)';
119
120     xnomp(:,i) = x_nom(end,:)';
121
122     w = Sw*randn(2,1);
123     x_t(:,i) = xnomp(:,i) + Omk*w; % Truth solution
124     %Simulate measurement truth
125     for j = 1:12
126         v = Sv*randn(3,1);
127         y_t(:,i,j) = dynamicsCalc(x_t(:,i),j,t_k)+v;
128         if any(~isnan(y_t(:,i,j)))
129             station(:,i,1) = j;
130         end
```

```
131         end
132     end
133         [x_state,Pp,NEES(m,:),NIS(m,:)] = EKF(x0,P0,tvec,station,Qk,Rtrue,
                y_t,xnomp);
134         xdiff = x_t - x_state(:,2:end);
135
136     end
137     %% NEES/NIS Stat Calc
138
139
140
141     figure
142     for i = 1:4
143         subplot(4,1,i)
144         hold on
145         plot(tvec, x_state(i,2:end))
146         hold off
147     end
148     sgtitle('EKF Simulated States')
149
150     NEESmean = mean(NEES);
151     NISmean = mean(NIS);
152
153     %Plotting noiseless and noisey signal
154     figure
155     sgtitle('Noisy Simulated Truth States')
156     for i = 1:4
157         subplot(4,1,i)
158         hold on
159         plot(tvec, x_t(i,:))
160         hold off
161     end
162
163     figure()
164     sgtitle('Noisy Simulated Data')
165     subplot(3,1,1)
166     hold on
167     for i = 1:12
168         scatter(tvec,y_t(1,:,i),'x')
169     end
170     ylabel('$\rho [km]$','Interpreter','latex')
171     hold off
172     subplot(3,1,2)
173     hold on
174     for i = 1:12
175         scatter(tvec,y_t(2,:,i))
```

```
176 end
177 ylabel('$\dot{\rho} [\frac{km}{s}]$','Interpreter','latex')
178 hold off
179 subplot(3,1,3)
180 hold on
181 for i = 1:12
182     scatter(tvec,y_t(3,:,i))
183 end
184 ylabel('$\phi [rad]$','Interpreter','latex')
185 xlabel('Time [s]')
186 hold off
187
188 figure
189 sgtitle('EKF State Estimation Errors')
190 for i = 1:4
191     subplot(4,1,i)
192     hold on
193     plot(tvec, xdiff(i,:))
194     sigma = 2*sqrt(Pp(i,i,:));
195     sigma = reshape(sigma,1,1401);
196     plot(tvec, sigma,'r--')
197     plot(tvec, -sigma,'r--')
198     hold off
199 end
200
201 r1 = chi2inv(alpha/2,N*4)/N;
202 r2 = chi2inv(1-alpha/2,N*4)/N;
203 figure
204 hold on
205 title('EKF NEES')
206 yline(r1,'r--')
207 yline(r2,'r--')
208 scatter(tvec,NEESmean,'b')
209 hold off
210
211 r1 = chi2inv(alpha/2,N*3)/N;
212 r2 = chi2inv(1-alpha/2,N*3)/N;
213 figure
214 hold on
215 title('EKF NIS')
216 scatter(tvec,NISmean,'b')
217 yline(r1,'r--')
218 yline(r2,'r--')
219 hold off
220
221 %% 6
```

```
222
223 [x_statedat,Pp,~,~] = EKFdata(x0,P0,tvec,station,Qtrue,Rtrue,y_t,x_t);
224 xdiff = x_t - x_statedat;
225
226 figure
227 sgtitle('EKF State Estimation Errors with data')
228 for i = 1:4
229     subplot(4,1,i)
230     hold on
231     plot(tvec, x_statedat(i,:))
232     plot(tvec,x_t(i,:),'-')
233 %     sigma = 2*sqrt(Pp(i,i,:));
234 %     sigma = reshape(sigma,1,1401);
235 %     plot(tvec, sigma,'r--')
236 %     plot(tvec, -sigma,'r--')
237     hold off
238 end
239 sgtitle('EKF State Simulation with Data')
240
241 figure
242 for i = 1:4
243     subplot(4,1,i)
244     hold on
245     plot(tvec, xdiff(i,:))
246     sigma = 2*sqrt(Pp(i,i,:));
247     sigma = reshape(sigma,1,1400);
248     plot(tvec(1:1400), sigma,'r--')
249     plot(tvec(1:1400), -sigma,'r--')
250     hold off
251 end
252 sgtitle('EKF State Estimation Errors with Data')
```

EKF.m

```
1 function [H] = linearizeH(i,x_vals,tvec,stat)
2 %Initialize
3 theta0 = (stat-1)*pi/6;% determine theta
4 rE = 6378;
5 mu = 4*10^5;
6 omega_e = 2*pi/86400;
7
8 %Allocate xvalues for H calculation
9 x1 = x_vals(1);
10 x2 = x_vals(2);
11 x3 = x_vals(3);
12 x4 = x_vals(4);
13
```

```
%Equations to determine a stations distance from the sat
Xi = rE * cos(omega_e * tvec(i) + theta0);
Yi = rE * sin(omega_e * tvec(i) + theta0);
Xi_dot = -rE * omega_e * sin(omega_e * tvec(i) + theta0);
Yi_dot = rE * omega_e * cos(omega_e * tvec(i) + theta0);

%More things to calculate
rho = sqrt((x1 - Xi)^2 + (x3 - Yi)^2);
rho_dot = ((x1 - Xi)*(x2 - Xi_dot) + (x3 - Yi)*(x4 - Yi_dot)) / rho;
phi = atan2((x3 - Yi),(x1 - Xi));

%Finally the output
H = [(x1-Xi)/rho 0 (x3-Yi)/rho 0;
            (x2-Xi_dot)/rho+(Xi-x1)*rho_dot/rho^2 (x1-Xi)/rho (x4-
                Yi_dot)/rho+(Yi-x3)*rho_dot/rho^2 (x3-Yi)/rho;
            (Yi-x3)/((x1-Xi)^2 + (x3-Yi)^2) 0 (x1-Xi)/((x1-Xi)^2 + (x3-
                Yi)^2) 0];

end
```

```
function [F,G] = linearizeMat(x_vals,dt)

x1 = x_vals(1);
x2 = x_vals(2);
x3 = x_vals(3);
x4 = x_vals(4);

rE = 6378;
mu = 4*10^5;
omega_e = 2*pi/86400;

A = [0 1 0 0;
            mu*(2*x1^2 -x3^2)/(x1^2 + x3^2)^(5/2) 0 3*mu*x1*x3/(x1^2 +
                x3^2)^(5/2) 0
            0 0 0 1;
            3*mu*x1*x3/(x1^2 + x3^2)^(5/2) 0 mu*(2*x3^2 -x1^2)/(x1^2 +
                x3^2)^(5/2) 0];
F = eye(4) + A*dt;

B = [0 0;
     1 0;
     0 0;
     0 1];

G = dt*B;
```

```
25
26 end
```

```
1  function [y_s,theta_i,phi_i] = dynamicsCalc(x_vals,stat,tvec)
2
3  %Initialize
4  theta0 = (stat-1)*pi/6; %determine theta
5  rE = 6378;
6  mu = 4*10^5;
7  omega_e = 2*pi/86400;
8
9  %Allocate xvalues for H calculation
10 x1 = x_vals(1);
11 x2 = x_vals(2);
12 x3 = x_vals(3);
13 x4 = x_vals(4);
14
15 %Equations to determine a stations distance from the sat
16 Xi = rE * cos(omega_e * tvec + theta0);
17 Yi = rE * sin(omega_e * tvec + theta0);
18 Xi_dot = -rE * omega_e * sin(omega_e * tvec + theta0);
19 Yi_dot = rE * omega_e * cos(omega_e * tvec + theta0);
20
21
22 %More things to calculate
23 rho = sqrt((x1 - Xi)^2 + (x3 - Yi)^2);
24 rho_dot = ((x1 - Xi)*(x2 - Xi_dot) + (x3 - Yi)*(x4 - Yi_dot)) / rho;
25 phi = atan2((x3 - Yi),(x1 - Xi));
26
27 y_s = [rho rho_dot phi]';
28
29 theta_i = atan2(Xi,Yi);
30 phi_i = atan2((x_vals(3)- Yi),(x_vals(1)- Xi));
31
32 thetai = atan2(Yi,Xi);
33     if abs(angdiff(thetai,phi)) > pi/2
34         y_s(1) = nan;
35         y_s(2) = nan;
36         y_s(3) = nan;
37    end
38 end
```

```
1  function dxdt = NLODE(t,x,mu)
2
3  Xpos = x(1);
4  Xvel = x(2);
5  Ypos = x(3);
```

```
Yvel = x(4);

r = norm([Xpos Ypos]);

Xacc = -mu*Xpos/r^3;
Yacc = -mu*Ypos/r^3;

dxdt = [Xvel;Xacc;Yvel;Yacc];

end
```

```
function [x_ukf,P_ukf,NEES,NIS] = UKF(x_t,y_t,station,tvec)
load('orbitdeterm_finalproj_KFdata.mat')
mu = 398600;

% y_d = y_t()
% if any(any(isnan(y_d))) || any(any(isempty(y_d)))
% %             yk = ystat(:,:,station(i));
% %             yk = yk(:,i);
%             stationCur = station(i);
%         else
%             stationCur = y_d(4);
%             yk = y_d(1:3,1);
%         end

dt = 10;
Omk = dt*[0 0 ; 1 0; 0 0 ; 0 1];
perturb_x0 = [0,0.075,0,-0.021];

P0 = 1e-3*eye(4);
P_p = P0; %What is P0? set it smalll..
dx_p = perturb_x0';
Qk = Qtrue;
% Qk = [0.02 0; 0 0.02];
Rp1 = Rtrue;

xp = [6678, 0, 0, 6678 * sqrt(mu/(6678^3))]';
Pp = 100*eye(4,4);
y_k = zeros(3,1);
Qk = Qtrue;
Rk = Rtrue;

%preallocate for UKF loop
n = length(x_t(:,1));
kappa = 0;
beta = 2;
```

```
alpha = 1e10;% estimate
lambda = alpha^2 * (n+kappa)-n;
tlast = 0;

for i = 1:length(tvec)
    %reset each loop
%     Pp = 100*eye(4,4);
    Sk = chol(Pp,'lower');
    Qk = eye(4); %Will need to adjust later
%     Pm_p1 = zeros(4,4);
    Pyy_p1 = zeros(3,3);
    Pxy_p1 = zeros(4,3);


%1. dynamics prediction step from time step k->k+1
    %%%%%%%%%%%%%%%%%%%
    %part a
    %%%%%%%%%%%%%%%%%%%
    chik0 = xp;
    for j = 1:n
        chik(:,j) = xp + (sqrt(n+lambda)*Sk(j,:)');
    end
    for j = (n+1):2*n
        chik(:,j) = xp - (sqrt(n+lambda)*Sk(j-n,:)');
    end
    chi_comb = [chik0, chik];
    inp_ic = chi_comb;
    %%%%%%%%%%%%%%%%%%%
    %part b
    %%%%%%%%%%%%%%%%%%%
    for j = 1:2*n+1
        func = @(t,inp)[inp(2); -mu*inp(1)/(sqrt((inp(1)^2+inp(3)^2)
            ^3);
             inp(4); -mu*inp(3)/(sqrt((inp(1)^2+inp(3)^2))^3)];
        tspan = [0 10]; %%%%%%%%%%
        [~, xout] = ode45(func,tspan,inp_ic(:,j));
        chi_m(:,j) = xout(end,:)';
    end
    %%%%%%%%%%%%%%%%%%%
    %part c - recombine resultants
    %%%%%%%%%%%%%%%%%%%
    for j = 1:2*n %includes 0 thus +1
        w_mi(:,j) = 1/(2*(n+lambda));
        w_ci(:,j) = w_mi(j);
    end
    w_m = [lambda/(n+lambda),w_mi];
```

```
81      w_c = [lambda/(n+lambda)+1-alpha^2+beta,w_ci];
82      xm_p1 = sum(w_m.*chi_m,2); % sum along dim 2
83
84      P_iter = zeros(4,4);
85      Pm_p1 = zeros(4,4);
86      for j = 1:2*n+1
87          P_iter = w_c(:,j).*(chi_m(:,j) - xm_p1)*(chi_m(:,j) - xm_p1)' +
                Qk;%THIS IS THE ERROR
88          Pm_p1 = Pm_p1 + P_iter;
89      end
90  %2. Measurement Update Step at time k+1 given observation y(k+1)
91      %%%%%%%%%%%%%%%%%%%
92      %part a - generate sigma pts
93      %%%%%%%%%%%%%%%%%%%
94      Sk_p1 = chol(Pm_p1,'lower');
95  %     Sk_p1 = chol(eye(4),'lower');
96      %repeat above steps for chi k+1
97      chik0_p1 = xm_p1;
98      for j = 1:n
99          chik_p1(:,j) = xm_p1 + (sqrt(n+lambda)*Sk_p1(j,:)');
100     end
101     for j = (n+1):2*n
102         chik_p1(:,j) = xp - (sqrt(n+lambda)*Sk_p1(j-n,:)');
103     end
104     chik_p1 =[chik0_p1 chik_p1]; %combine
105     %%%%%%%%%%%%%%%%%%%
106     %part b - generate sigma pts
107     %%%%%%%%%%%%%%%%%%%
108     for j = 1:2*n+1
109         [~,gam_p1(:,j),~,~] = linearizeH(i,chik_p1(:,j),tvec,station(:,
                i));
110     end
111     %%%%%%%%%%%%%%%%%%%
112     %part c - get predicted measurement mean and measurement covar
113     %%%%%%%%%%%%%%%%%%%
114     ym_p1 = sum(w_m.*gam_p1,2);
115     P_iter_yy = zeros(3,3);
116     for j = 1:2*n+1
117         P_iter_yy = w_c(:,j).*(gam_p1(:,j) - ym_p1)*(gam_p1(:,j) -
                ym_p1)' + Rk;
118         Pyy_p1 = Pyy_p1 + P_iter_yy;
119     end
120     %%%%%%%%%%%%%%%%%%%
121     %part d - get state measurement cross-covariance matrix (nxp)
122     %%%%%%%%%%%%%%%%%%%
123     for j = 1:2*n+1
```

```
124         Pxy_p1_iter = w_c(:,j).*(chik_p1(:,j)- xm_p1)*(gam_p1(:,j)-ym_p1)';
                %%
125         Pxy_p1 = Pxy_p1 + Pxy_p1_iter;
126         end
127         %%%%%%%%%%%%%%%%%%%
128         %part e - Estimate kalman gain matrix (nxp)
129         %%%%%%%%%%%%%%%%%%%
130         Kk_p1 = Pxy_p1*inv(Pyy_p1);
131         %%%%%%%%%%%%%%%%%%%
132         %part f - Perform kalman state and covariance update with
                observation yk+1 (nxp)
133         %%%%%%%%%%%%%%%%%%%
134         xp_p1 = xm_p1 + Kk_p1*(y_t(:,i,station(i))- ym_p1);
135         Pp_p1 = Pm_p1 - Pxy_p1*inv(Pyy_p1)*Pxy_p1';
136
137         %Store variables
138         x_ukf(:,i) = xp_p1;
139         P_ukf(:,:,i) = Pp_p1;
140         %Set for next iteration
141         xp = xp_p1;
142         Pp = Pp_p1;
143         tlast = tvec(i);
144     end
145
146     NEES = 1;
147     NIS = 1;
148
149     end
```

```
1   %Housekeeping
2   clear
3   clc
4   close all
5   %% Part 1
6   rng(100)
7   load('orbitdeterm_finalproj_KFdata.mat')
8   rE = 6378;
9   mu = 4*10^5;
10  omega_e = 2*pi/86400;
11
12  perturb_x0 = [0,0.075,0,-0.021]';
13  x0 = [6678, 0, 0, 6678 * sqrt(mu/(6678^3))]';
14
15  B = [0  0;
16       1  0;
17       0  0;
```

```
18      0  1];
19  Gamma = [0  0;
20           1  0;
21           0  0;
22           0  1];
23  delT = 10;
24
25  x = x0+perturb_x0;
26
27  P = 2*pi*sqrt(6678^3/mu);
28  figure
29  for j = 1:12
30      theta0 = (j-1)*pi/6;
31      y = [];
32      for i = 1:length(tvec)
33          x_state = x(:,i);
34          x1 = 6678*cos(2*pi/P * tvec(i));
35          x2 = -6678*sin(2*pi/P * tvec(i));
36          x3 = 6678*sin(2*pi/P * tvec(i));
37          x4 = 6678*cos(2*pi/P * tvec(i));
38          [F,G] = linearizeMat([x1 x2 x3 x4]', delT);
39
40          perturb_x0_new = F*perturb_x0(:,i);
41      %       x_state(1) = x1;
42      %       x_state(3) = x3;
43          x_state_new = [x1 x2 x3 x4]' + perturb_x0_new;
44          x = [x x_state_new];
45
46  %           x1 = x_state(1);
47  %           x2 = x_state(2);
48  %           x3 = x_state(3);
49  %           x4 = x_state(4);
50
51          Xi = rE * cos(omega_e * tvec(i) + theta0);
52          Yi = rE * sin(omega_e * tvec(i) + theta0);
53          Xi_dot = -rE * omega_e * sin(omega_e * tvec(i) + theta0);
54          Yi_dot = rE * omega_e * cos(omega_e * tvec(i) + theta0);
55
56          rho = sqrt((x1 - Xi)^2 + (x3 - Yi)^2);
57          rho_dot = ((x1 - Xi)*(x2 - Xi_dot) + (x3 - Yi)*(x4 - Yi_dot)) /
                rho;
58          phi = atan2((x3 - Yi),(x1 - Xi));
59
60          H = [(x1-Xi)/rho 0 (x3-Yi)/rho 0;
61              (x2-Xi_dot)/rho+(Xi-x1)*rho_dot/rho^2 (x1-Xi)/rho (x4-
                    Yi_dot)/rho+(Yi-x3)*rho_dot/rho^2 (x3-Yi)/rho;
```

```
62              (Yi-x3)/((x1-Xi)^2 + (x3-Yi)^2) 0 (x1-Xi)/((x1-Xi)^2 + (x3-
                    Yi)^2) 0];
63          perturb_y = H*perturb_x0(:,i);
64          ystate = [rho rho_dot phi]' + perturb_y;
65          y = [y ystate];
66
67          perturb_x0 = [perturb_x0 perturb_x0_new];
68
69          thetai = atan2(Yi,Xi);
70          if abs(angdiff(thetai,phi)) > pi/2
71              y(1,i) = nan;
72              y(2,i) = nan;
73              y(3,i) = nan;
74          end
75      end
76      subplot(3,1,1)
77      plot(tvec,y(1,:),'o')
78      hold on
79      xlabel('Time (s)')
80      ylabel('$\rho (km)$','Interpreter','latex')
81      subplot(3,1,2)
82      plot(tvec,y(2,:),'o')
83      hold on
84      xlabel('Time (s)')
85      ylabel('$\dot{\rho} (km)$','Interpreter','latex')
86      subplot(3,1,3)
87      plot(tvec,y(3,:),'o')
88      hold on
89      xlabel('Time (s)')
90      ylabel('$\phi (rad)$','Interpreter','latex')
91      sgtitle('Linearized Output')
92  end
93
94
95  figure
96  subplot(4,1,1)
97  plot(tvec,x(1,1:length(tvec)))
98  xlabel('Time (s)')
99  ylabel('X (km)')
100 subplot(4,1,2)
101 plot(tvec,x(2,1:length(tvec)))
102 xlabel('Time (s)')
103 ylabel('$\dot{X}$ (km)','Interpreter','latex')
104 subplot(4,1,3)
105 plot(tvec,x(3,1:length(tvec)))
106 xlabel('Time (s)')
```

```
107 ylabel('Y (km)')
108 subplot(4,1,4)
109 plot(tvec,x(4,1:length(tvec)))
110 xlabel('Time (s)')
111 ylabel('$\dot{Y}$ (km)','Interpreter','latex')
112 sgtitle('Linearized States')
113
114 figure
115 subplot(4,1,1)
116 plot(tvec,perturb_x0(1,1:length(tvec)))
117 xlabel('Time (s)')
118 ylabel('\delta X (km)')
119 subplot(4,1,2)
120 plot(tvec,perturb_x0(2,1:length(tvec)))
121 xlabel('Time (s)')
122 ylabel('$\delta \dot{X}$ (km)','Interpreter','latex')
123 subplot(4,1,3)
124 plot(tvec,perturb_x0(3,1:length(tvec)))
125 xlabel('Time (s)')
126 ylabel('\delta Y (km)')
127 subplot(4,1,4)
128 plot(tvec,perturb_x0(4,1:length(tvec)))
129 xlabel('Time (s)')
130 ylabel('$\delta \dot{Y}$ (km)','Interpreter','latex')
131 sgtitle('Linearized Perturbations')
132
133 %% 3
134
135 dT = 10;
136
137 tspan = [0 1400*dT];
138
139 mu = 398600;
140
141 r0 = 6678;
142
143 x0 = [r0; 0; 0; r0 * sqrt(mu/r0^3)];
144
145 dx0 = [0;0.075;0;-0.021];
146
147 opts = odeset('reltol',1e-12,'abstol',1e-12);
148
149 [t,x] = ode45(@(t,x) NLODE(t,x,mu),tspan,x0+dx0,opts);
150
151 figure()
152 sgtitle('Nonlinear Simulation States')
```

```
153 subplot(4,1,1)
154 hold on
155 plot(t,x(:,1))
156 ylabel('$X [km]$','Interpreter','latex')
157 hold off
158 subplot(4,1,2)
159 hold on
160 plot(t,x(:,2))
161 ylabel('$\dot{X} [\frac{km}{s}]$ ','Interpreter','latex')
162 hold off
163 subplot(4,1,3)
164 hold on
165 plot(t,x(:,3))
166 ylabel('$Y [km]$','Interpreter','latex')
167 hold off
168 subplot(4,1,4)
169 hold on
170 plot(t,x(:,4))
171 xlabel('Time [s]')
172 ylabel('$\dot{Y} [\frac{km}{s}]$','Interpreter','latex')
173 hold off
174
175 tmax = length(t);
176
177 RE = 6378;
178
179 wE = 2*pi/86400;
180
181 for i = 1:12
182
183     thetas0(i) = (i-1)*pi/6;
184
185     for k = 1:tmax
186         tk = t(k);
187         Xs = RE*cos(wE*tk+thetas0(i));
188         Ys = RE*sin(wE*tk+thetas0(i));
189         Xsd = -RE*wE*sin(wE*tk+thetas0(i));
190         Ysd = RE*wE*cos(wE*tk+thetas0(i));
191         thetas(i,k) = atan2(Ys,Xs);
192         rho(i,k) = sqrt((x(k,1)-Xs)^2 + (x(k,3)-Ys)^2);
193         rhod(i,k) = ((x(k,1)-Xs)*(x(k,2)-Xsd)+(x(k,3)-Ys)*(x(k,4)-Ysd))
              /rho(i,k);
194         phi(i,k) = atan2(x(k,3)-Ys,x(k,1)-Xs);
195
196         if abs(angdiff(thetas(i,k),phi(i,k))) > pi/2
197             rho(i,k) = nan;
```

```
198                 rhod(i,k) = nan;
199                 phi(i,k) = nan;
200             end
201
202
203         end
204     end
205
206     figure()
207     sgtitle('Nonlinear Simulation Outputs')
208     subplot(3,1,1)
209     hold on
210     for i = 1:12
211         scatter(t,rho(i,:),'x')
212     end
213     ylabel('$\rho [km]$','Interpreter','latex')
214     hold off
215     subplot(3,1,2)
216     hold on
217     for i = 1:12
218         scatter(t,rhod(i,:))
219     end
220     ylabel('$\dot{\rho} [\frac{km}{s}]$','Interpreter','latex')
221     hold off
222     subplot(3,1,3)
223     hold on
224     for i = 1:12
225         scatter(t,phi(i,:))
226     end
227     ylabel('$\phi [rad]$','Interpreter','latex')
228     xlabel('Time [s]')
229     hold off
230
231
232     %% Part II STOCHASTIC NONLINEAR FILTER
233     %data vars: Qtrue, Rtrue, tvec, ydata
234
235     %Important Notes:
236     %No input u
237     dt = delT;
238
239     % Qk = [0.02 0; 0 0.02];
240     Qk = Qtrue;
241
242
243     Omk = dt*[0 0 ; 1 0; 0 0 ; 0 1];
```

```
Sw = chol(Qk,'lower');
Sv = chol(Rtrue,'lower');

% Monte Carlo Sim

N = 10;
n = 2;

alpha = 0.05;

for m = 1:N

    x_k = [6678, 0, 0, 6678 * sqrt(mu/(6678^3))]';

%Noisy Data sim of x
for i = 1:length(tvec)

    %Simulate truth
    w_k = mvnrnd(zeros(2,1),Qk); % of Qtrue
    t_k = tvec(i);

    [~,x_nom] = ode45(@(t,x) odeLKF(t_k,x,[0;0]),[0 dt],x_k);

    x_nom = x_nom';

    x_k = x_nom(:,end);

    xnomp(:,i) = x_nom(:,end);

    w = Sw*randn(2,1);
    x_t(:,i) = xnomp(:,i) + Omk*w; % Truth solution
    %Simulate measurement truth
    for j = 1:12
        v = Sv*randn(3,1);
        y_t(:,i,j) = dynamicsCalc(x_t(:,i),j,t_k)+v;
        if any(~isnan(y_t(:,i,j)))
            station(:,i,1) = j;
        end
    end
end

[x_ukf,P_ukf,NEES,NIS] = UKF(x_t,y_t,station,tvec);

xdiff = x_t - x_ukf;

end
```

```
290
291 NEESmean = mean(NEES);
292 NISmean = mean(NIS);
293 %%
294 %Plotting noiseless and noisey signal
295 figure
296 sgtitle('Noisy Simulated Truth States')
297 for i = 1:4
298     subplot(4,1,i)
299     hold on
300     plot(tvec, x_t(i,:))
301     hold off
302 end
303
304 figure()
305 sgtitle('Noisy Simulated Data')
306 subplot(3,1,1)
307 hold on
308 for i = 1:12
309     scatter(tvec,y_t(1,:,i),'x')
310 end
311 ylabel('$\rho [km]$','Interpreter','latex')
312 hold off
313 subplot(3,1,2)
314 hold on
315 for i = 1:12
316     scatter(tvec,y_t(2,:,i))
317 end
318 ylabel('$\dot{\rho} [\frac{km}{s}]$','Interpreter','latex')
319 hold off
320 subplot(3,1,3)
321 hold on
322 for i = 1:12
323     scatter(tvec,y_t(3,:,i))
324 end
325 ylabel('$\phi [rad]$','Interpreter','latex')
326 xlabel('Time [s]')
327 hold off
328
329 figure
330 sgtitle('UKF State versue Truth State')
331 for i = 1:4
332     subplot(4,1,i)
333     hold on
334 %     plot(tvec, xdiff(i,:))
335     sigma = (x_ukf(i,:));
```

```
%      sigma = reshape(sigma,1,1401);
hold on
    plot(tvec, sigma,'ro')
    plot(tvec, x_t(i,:),'b','linewidth',2.5)
        ylabel(ylab(i))
    xlabel('Time (s)')
 hold off

%      plot(tvec, -sigma,'r--')
legend('Xukf', 'Xtruth')
    hold off
end

figure
sgtitle('UKF State versue Truth State')
ylab = ["X position [km]" , "X Vel. [km/s]", "Y position [km]","Y Vel.
    [km/s]"];
for i = 1:4
    subplot(4,1,i)
    hold on
    plot(tvec, xdiff(i,:))
    sigma = 2*sqrt(P_ukf(i,i,:));
    sigma = reshape(sigma,1,1401);
    plot(tvec, sigma,'r--')
    plot(tvec, -sigma,'r--')
    ylabel(ylab(i))
    xlabel('Time (s)')
    hold off
end

n = 4;

r1 = chi2inv(alpha/2,N*n)/N;
r2 = chi2inv(1-alpha/2,N*n)/N;

figure
hold on
title('NEES')
yline(r1,'r--')
yline(r2,'r--')
scatter(tvec,NEESmean,'b')
hold off

n = 3;

r1 = chi2inv(alpha/2,N*n)/N;
```

```
r2 = chi2inv(1-alpha/2,N*n)/N;

figure
hold on
title('NIS')
yline(r1,'r--')
yline(r2,'r--')
scatter(tvec,NISmean,'b')
hold off

%% 6

% [x_lkf,P_lkf,NEES,NIS] = LKFdata(x_t,station,tvec);
%
% xdiff = x_t(:,2:1401) - x_lkf;
%
%
% figure
% sgtitle('LKF output')
% for i = 1:4
%     subplot(4,1,i)
%     hold on
%     plot(tvec(2:1401), x_lkf(i,:))
%     hold off
% end
%
%
% figure
% sgtitle('LKF State Estimation Errors')
% for i = 1:4
%     subplot(4,1,i)
%     hold on
%     plot(tvec(2:1401), xdiff(i,:))
%     sigma = 2*sqrt(P_lkf(i,i,:));
%     sigma = reshape(sigma,1,1400);
%     plot(tvec(2:1401), sigma,'r--')
%     plot(tvec(2:1401), -sigma,'r--')
%     hold off
% end
```