University of Colorado Boulder

Aerospace Engineering Sciences

ASEN 5044 Statistical Estimation for Dynamical Systems

Final Project Report - Statistical Orbit Determination

Authors: Corey LePine and William Watkins

14th December, 2021

College of Engineering & Applied Science
UNIVERSITY OF COLORADO BOULDER

0. Team Member Contributions

Team Member	Final Project Report Contributions
Corey LePine	Question 1, Question 3 (CT and Nonlinear), Question 5, Question 6
William Watkins	Question 2, Question 3 (DT and Linearized), Question 4, Question 6, Advanced Limerick

I. Deterministic System Analysis

1. CT Jacobians for CT Linearized Model Parameters

To find the CT Jacobians of the system, we must define the common vectors and variables used for this analysis. First, the state vector for the system is given below. Where X and $\dot{X}$ are the position and velocity of the spacecraft in the $\hat{X}$ direction. Y and $\dot{Y}$ are the position and velocity of the spacecraft in the $\hat{Y}$ direction.

$$\mathbf{x} = \begin{bmatrix} X \\ \dot{X} \\ Y \\ \dot{Y} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{1}$$

The derivative of the state vector is given below.

$$\mathbf{xdot} = \begin{bmatrix} \dot{X} \\ \ddot{X} \\ \dot{Y} \\ \ddot{Y} \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{-\mu x_1}{\sqrt{x_1^2+x_3^2}^3} + u_1 + w_1 \\ x_4 \\ \frac{-\mu x_3}{\sqrt{x_1^2+x_3^2}^3} + u_2 + w_2 \end{bmatrix} \tag{2}$$

The vector below is the state vector for the ground stations that track the spacecraft while in orbit. The i denotes which ground station is currently viewing the spacecraft.

$$\mathbf{x}^i = \begin{bmatrix} X^i \\ \dot{X}^i \\ Y^i \\ \dot{Y}^i \end{bmatrix} = \begin{bmatrix} x_1^i \\ x_2^i \\ x_3^i \\ x_4^i \end{bmatrix} \tag{3}$$

There are only two inputs that act on the spacecraft and their vector is given below.

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{4}$$

The disturbance vector that acts on the system is given below.

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \tag{5}$$

Since the nominal trajectory for this system is a circular orbit, the distance between the center of the Earth to the spacecraft is shown in equation 6.

$$r = \sqrt{x_1^2 + x_3^2} \tag{6}$$

After defining the common vectors used, we can begin finding the CT jacobian matrices. We define a f matrix and components that represent the derivative of the state of the spacecraft.

$$\mathbf{xdot} = f(\mathbf{x},\mathbf{u},\mathbf{w},t) = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{-\mu x_1}{r^3} + u_1 + w_1 \\ x_4 \\ \frac{-\mu x_3}{r^3} + u_2 + w_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{-\mu x_1}{\sqrt{x_1^2+x_3^2}^3} + u_1 + w_1 \\ x_4 \\ \frac{-\mu x_3}{\sqrt{x_1^2+x_3^2}^3} + u_2 + w_2 \end{bmatrix} \tag{7}$$

The A matrix of the CT system is found as follows. Note, that since the state of the spacecraft varies with time, the A matrix is LTV. Based on the states of the system, A should be a 4x4 matrix. Note that A should be evaluated at the nominal condition for linearization.

$$A_{nom} = \begin{bmatrix} \frac{df_1}{dx_1} & \frac{df_1}{dx_2} & \frac{df_1}{dx_3} & \frac{df_1}{dx_4} \\ \frac{df_2}{dx_1} & \frac{df_2}{dx_2} & \frac{df_2}{dx_3} & \frac{df_2}{dx_4} \\ \frac{df_3}{dx_1} & \frac{df_3}{dx_2} & \frac{df_3}{dx_3} & \frac{df_3}{dx_4} \\ \frac{df_4}{dx_1} & \frac{df_4}{dx_2} & \frac{df_4}{dx_3} & \frac{df_4}{dx_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{\mu(2x_1^2-x_3^2)}{r^5} & 0 & \frac{3\mu x_1 x_3}{r^5} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{3\mu x_1 x_3}{r^5} & 0 & \frac{\mu(2x_3^2-x_1^2)}{r^5} & 0 \end{bmatrix} \tag{8}$$

The B matrix of the CT system is given below. Unlike the A matrix, B is LTI. Based on the state and size of the inputs, B should be a 4x2 matrix.

$$B = \begin{bmatrix} \frac{df_1}{du_1} & \frac{df_1}{du_2} \\ \frac{df_2}{du_1} & \frac{df_2}{du_2} \\ \frac{df_3}{du_1} & \frac{df_3}{du_2} \\ \frac{df_4}{du_1} & \frac{df_4}{du_2} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{9}$$

The Γ matrix of the CT system is given below. It is similar to the B matrix in that is LTI and is a 4x2 matrix since there are only 2 disturbances acting on the system.

$$\Gamma = \begin{bmatrix} \frac{df_1}{dw_1} & \frac{df_1}{dw_2} \\ \frac{df_2}{dw_1} & \frac{df_2}{dw_2} \\ \frac{df_3}{dw_1} & \frac{df_3}{dw_2} \\ \frac{df_4}{dw_1} & \frac{df_4}{dw_2} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{10}$$

In a similar manner to the state vector and its CT jacobians, the measurement vector (shown below) is used to find the C matrix.

$$y = h(\mathbf{x},\mathbf{x}^i) = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} \sqrt{(x_1 - x_1^i)^2 + (x_3 - x_3^i)^2} \\ \frac{(x_1-x_1^i)(x_2-x_2^i)+(x_3-x_3^i)(x_4-x_4^i)}{\sqrt{(x_1-x_1^i)^2+(x_3-x_3^i)^2}} \\ \arctan\frac{x_3-x_3^i}{x_1-x_1^i} \end{bmatrix} \tag{11}$$

The method for solving the C matrix is given below. Based on the measurements and the state of the system, C is a 3x4 matrix and is LTV. Like the A matrix, the given C matrix must be evaluated at the nominal condition for linearization.

$$C = \begin{bmatrix} \frac{dh_1}{dx_1} & \frac{dh_1}{dx_2} & \frac{dh_1}{dx_3} & \frac{dh_1}{dx_4} \\ \frac{dh_2}{dx_1} & \frac{dh_2}{dx_2} & \frac{dh_2}{dx_3} & \frac{dh_2}{dx_4} \\ \frac{dh_3}{dx_1} & \frac{dh_3}{dx_2} & \frac{dh_3}{dx_3} & \frac{dh_3}{dx_4} \end{bmatrix} = \begin{bmatrix} \frac{x_1-x_1^i}{\sqrt{(x_1-x_1^i)^2+(x_3-x_3^i)^2}} & 0 & \frac{x_3-x_3^i}{\sqrt{(x_1-x_1^i)^2+(x_3-x_3^i)^2}} & 0 \\ C(2,1) & \frac{x_1-x_1^i}{\sqrt{(x_1-x_1^i)^2+(x_3-x_3^i)^2}} & C(2,3) & \frac{x_3-x_3^i}{\sqrt{(x_1-x_1^i)^2+(x_3-x_3^i)^2}} \\ \frac{x_3^i-x_3}{(x_1-x_1^i)^2+(x_3-x_3^i)^2} & 0 & \frac{x_1-x_1^i}{(x_1-x_1^i)^2+(x_3-x_3^i)^2} & 0 \end{bmatrix} \tag{12}$$

$$C(2,1) = \frac{(x_2 - x_2^i)[(x_1 - x_1^i)^2 + (x_3 - x_3^i)^2] - (x_1 - x_1^i)[(x_1 - x_1^i)(x_2 - x_2^i) + (x_3 - x_3^i)(x_4 - x_4^i)]}{((x_1 - x_1^i)^2 + (x_3 - x_3^i)^2)^{\frac{3}{2}}} \tag{13}$$

$$C(2,3) = \frac{(x_4 - x_4^i)[(x_1 - x_1^i)^2 + (x_3 - x_3^i)^2] - (x_3 - x_3^i)[(x_1 - x_1^i)(x_2 - x_2^i) + (x_3 - x_3^i)(x_4 - x_4^i)]}{((x_1 - x_1^i)^2 + (x_3 - x_3^i)^2)^{\frac{3}{2}}} \tag{14}$$

2. Linearization and DT Linearized Model Matrices

First, the nominal trajectory for the spacecraft must be calculated. For this spacecraft, the nominal trajectory is:

$$X(t) = x_1 = r_0 cos(\sqrt{\frac{\mu t}{{r_0}^3}}) \tag{15}$$

$$\dot{X}(t) = x_2 = -r_0 sin(\sqrt{\frac{\mu t}{{r_0}^3}})\sqrt{\frac{\mu t}{{r_0}^3}} \tag{16}$$

$$Y(t) = x_3 = r_0 sin(\sqrt{\frac{\mu t}{{r_0}^3}}) \tag{17}$$

$$\dot{Y}(t) = x_4 = r_0 cos(\sqrt{\frac{\mu t}{{r_0}^3}})\sqrt{\frac{\mu t}{{r_0}^3}} \tag{18}$$

$$r = \sqrt{(r_0 cos(\sqrt{\frac{\mu t}{{r_0}^3}}))^2 + (r_0 sin(\sqrt{\frac{\mu t}{{r_0}^3}}))^2} \tag{19}$$

Linearizing the CT Jacobians about this trajectory, we get:

$$A_{nom} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{\mu(2(r_0 cos(\sqrt{\frac{\mu t}{{r_0}^3}}))^2 - (r_0 sin(\sqrt{\frac{\mu t}{{r_0}^3}}))^2)}{(\sqrt{(r_0 cos(\sqrt{\frac{\mu t}{{r_0}^3}}))^2 + (r_0 sin(\sqrt{\frac{\mu t}{{r_0}^3}}))^2})^5} & 0 & \frac{3\mu(r_0 cos(\sqrt{\frac{\mu t}{{r_0}^3}}))(r_0 sin(\sqrt{\frac{\mu t}{{r_0}^3}}))}{(\sqrt{(r_0 cos(\sqrt{\frac{\mu t}{{r_0}^3}}))^2 + (r_0 sin(\sqrt{\frac{\mu t}{{r_0}^3}}))^2})^5} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{3\mu(r_0 cos(\sqrt{\frac{\mu t}{{r_0}^3}}))(r_0 sin(\sqrt{\frac{\mu t}{{r_0}^3}}))}{(\sqrt{(r_0 cos(\sqrt{\frac{\mu t}{{r_0}^3}}))^2 + (r_0 sin(\sqrt{\frac{\mu t}{{r_0}^3}}))^2})^5} & 0 & \frac{\mu(2(r_0 sin(\sqrt{\frac{\mu t}{{r_0}^3}}))^2 - (r_0 cos(\sqrt{\frac{\mu t}{{r_0}^3}}))^2)}{(\sqrt{(r_0 cos(\sqrt{\frac{\mu t}{{r_0}^3}}))^2 + (r_0 sin(\sqrt{\frac{\mu t}{{r_0}^3}}))^2})^5} & 0 \end{bmatrix}$$

(20)

$$B_{nom} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{21}$$

$$\Gamma_{nom} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{22}$$

The locations of the ith station are known as

$$\begin{aligned} X^i(t) = x_1^i = R_E cos(\omega_E t + \theta^i(0)) \\ Y^i(t) = x_3^i = R_E sin(\omega_E t + \theta^i(0)) \end{aligned} \tag{23}$$

And the velocities of each station are

$$\begin{aligned}\dot{X}^i(t) &= x_2^i = -R_E sin(\omega_E t + \theta^i(0))\omega_E \\ \dot{Y}^i(t) &= x_4^i = R_E cos(\omega_E t + \theta^i(0))\omega_E\end{aligned} \tag{24}$$

$$C_{nom}^i = \begin{bmatrix} C_{1,1}^i & 0 & C_{1,3}^i & 0 \\ C_{2,1}^i & C_{2,2}^i & C_{2,3}^i & C_{2,4}^i \\ C_{3,1}^i & 0 & C_{3,3}^i & 0 \end{bmatrix} \tag{25}$$

with

$$\begin{aligned}
C_{1,1}^i &= \frac{r_0 cos(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E cos(\omega_E t + \theta^i(0))}{\sqrt{(r_0 cos(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E cos(\omega_E t + \theta^i(0)))^2 + (r_0 sin(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E sin(\omega_E t + \theta^i(0)))^2}} \\
C_{1,3}^i &= \frac{r_0 sin(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E sin(\omega_E t + \theta^i(0))}{\sqrt{(r_0 cos(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E cos(\omega_E t + \theta^i(0)))^2 + (r_0 sin(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E sin(\omega_E t + \theta^i(0)))^2}} \\
C_{2,1}^i &= \frac{(x_2 - x_2^i)[(x_1 - x_1^i)^2 + (x_3 - x_3^i)^2] - (x_1 - x_1^i)[(x_1 - x_1^i)(x_2 - x_2^i) + (x_3 - x_3^i)(x_4 - x_4^i)]}{((x_1 - x_1^i)^2 + (x_3 - x_3^i)^2)^{\frac{3}{2}}} \\
C_{2,2}^i &= \frac{r_0 cos(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E cos(\omega_E t + \theta^i(0))}{\sqrt{(r_0 cos(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E cos(\omega_E t + \theta^i(0)))^2 + (r_0 sin(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E sin(\omega_E t + \theta^i(0)))^2}} \\
C_{2,3}^i &= C(2,3) = \frac{(x_4 - x_4^i)[(x_1 - x_1^i)^2 + (x_3 - x_3^i)^2] - (x_3 - x_3^i)[(x_1 - x_1^i)(x_2 - x_2^i) + (x_3 - x_3^i)(x_4 - x_4^i)]}{((x_1 - x_1^i)^2 + (x_3 - x_3^i)^2)^{\frac{3}{2}}} \\
C_{2,4}^i &= \frac{r_0 sin(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E sin(\omega_E t + \theta^i(0))}{\sqrt{(r_0 cos(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E cos(\omega_E t + \theta^i(0)))^2 + (r_0 sin(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E sin(\omega_E t + \theta^i(0)))^2}} \\
C_{3,1}^i &= \frac{R_E sin(\omega_E t + \theta^i(0)) - r_0 sin(\sqrt{\frac{\mu t}{r_0{}^3}})}{(r_0 cos(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E cos(\omega_E t + \theta^i(0)))^2 + (r_0 sin(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E sin(\omega_E t + \theta^i(0)))^2} \\
C_{3,3}^i &= \frac{r_0 cos(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E cos(\omega_E t + \theta^i(0))}{(r_0 cos(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E cos(\omega_E t + \theta^i(0)))^2 + (r_0 sin(\sqrt{\frac{\mu t}{r_0{}^3}}) - R_E sin(\omega_E t + \theta^i(0)))^2}
\end{aligned} \tag{26}$$

The DT linearized matrices are

$$\begin{aligned} F_k &= I + A(t)\Delta t \\ G_k &= \Delta t B(t) \\ \Omega_k &= \Delta t \Gamma(t) \\ H_k^i &= C_{nom}^i \end{aligned} \tag{27}$$

Because the system is LTV, the observability and stability of the system are not applicable, and so will not be discussed.

3. Nonlinear and DT Linearized Simulations

Below we have a plot of the perturbations calculated from the continuous time nonlinearized model and the discrete time linearized model.

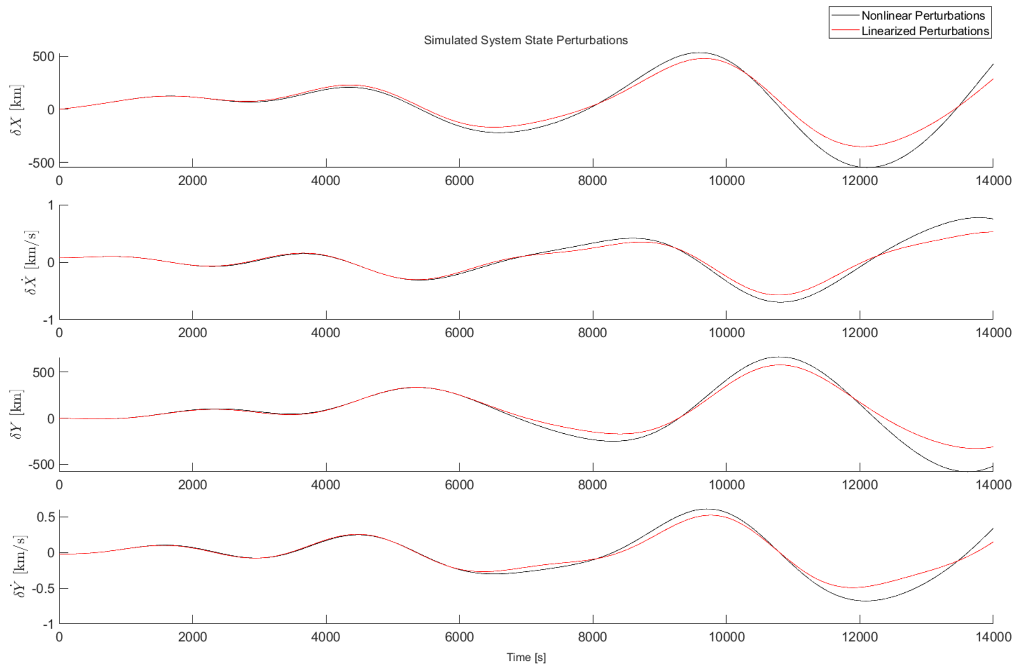

Fig. 1 System Perturbations

As is expected, the perturbations calculated by the linearized model increasingly deviate from those calculated by the nonlinear model. That is, the nonlinear dynamics predict a great perturbation than the linearized dynamics. This is due to the system deviating away from the nominal trajectory the discrete time model was linearized around. We can observe the effect this has on the states in the plot below.

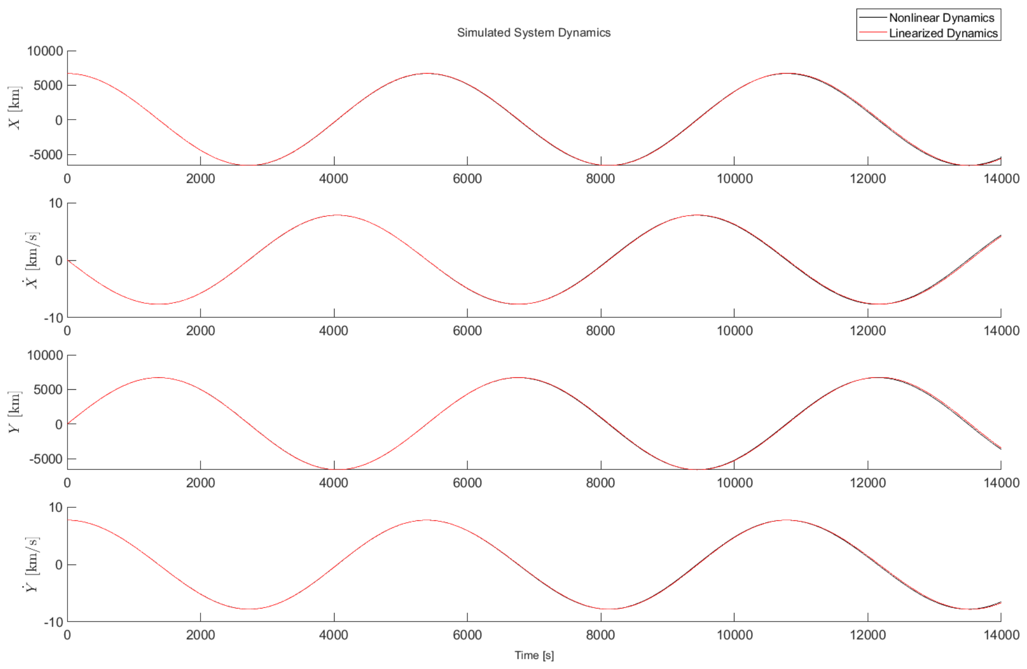

Fig. 2 States from Nonlinear and Linear Dynamics

While the scale of the plot can make the difference hard to see, the two models start to visually separate around $t = 7000s$, and they continue to deviate as the simulation goes on. However, the results are extremely similar and indicate the linearized model is contructed correctly. In addition, the states from both models, and the perturbations of the linearized model, are exactly equal to the solution sketches post on the Canvas website.

Moving onto the measurements, the nonlinear measurements are plotted below, as are the linearized measurements.

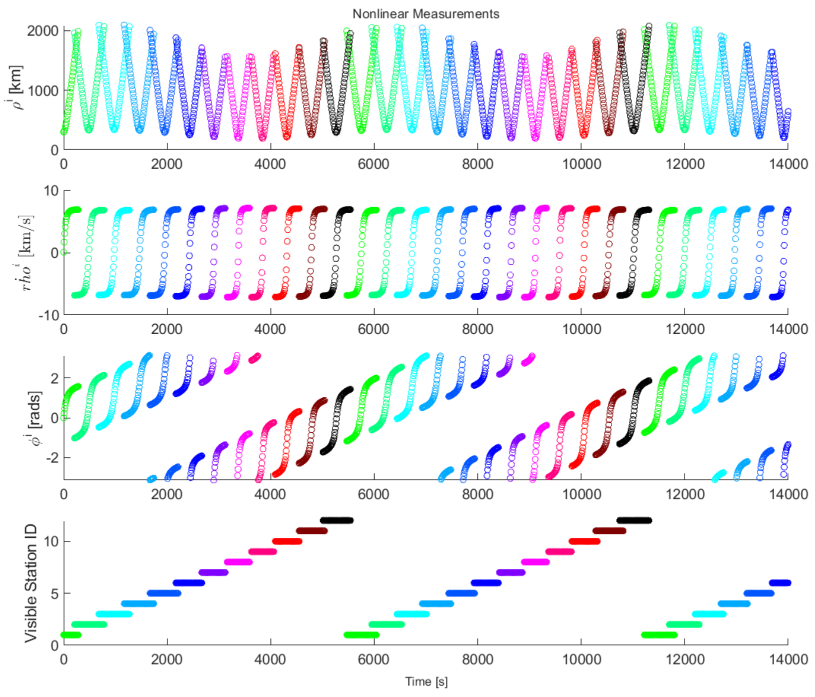

Fig. 3 Measurements from Nonlinear Dynamics

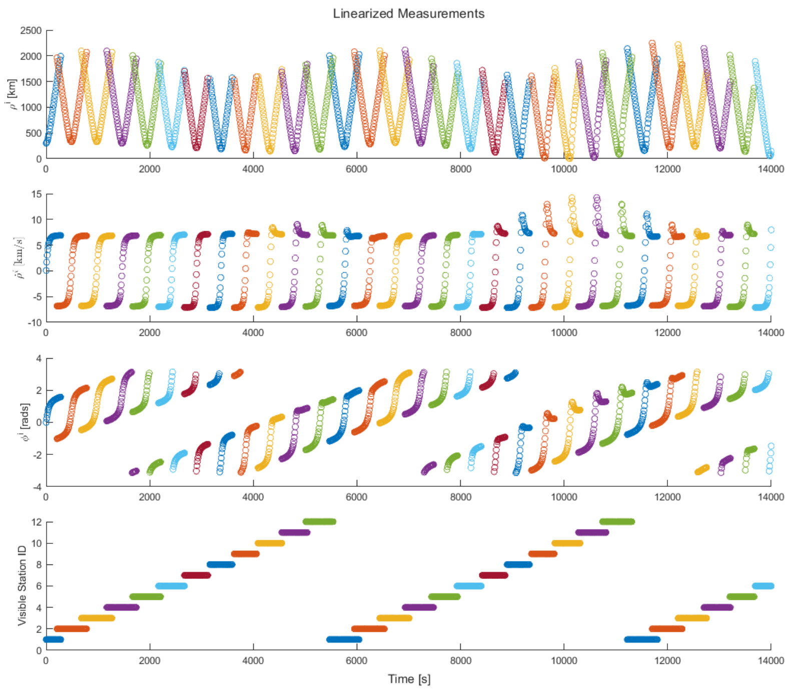

Fig. 4 Measurements from Linear Dynamics

Not only are both measurement plots similar to each other, they are also identical to the solution sketches provided on Canvas.

II. Stochastic Nonlinear Filtering

For this section of the report, we will be discussing how we implemented and tuned a Linearized Kalman Filter (LKF) and an Extended Kalman Filter (EKF). Both filters are used for linear time varying (LTV) systems that the normal Kalman Filter can not quite handle. For both filters, we used the provided nominal state trajectory initial condition, nominal control inputs (none for this problem), DT AWGN process and measurement noise covariance matrices provided on Canvas [REF].

The nominal state trajectory initial condition is:

$$\mathbf{x} = \begin{bmatrix} X \\ \dot{X} \\ Y \\ \dot{Y} \end{bmatrix} = \begin{bmatrix} 6678km \\ 0km/s \\ 0km \\ 7.7258km/s \end{bmatrix} \tag{28}$$

The DT AWGN process noise covariance matrix for this system is:

$$Q_{true} = \begin{bmatrix} 10^{-9} & 0 \\ 0 & 10^{-9} \end{bmatrix} \tag{29}$$

The DT AWGN measurement noise covariance matrix for this system is:

$$R_{true} = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \tag{30}$$

For tuning the KFs, we used Monte Carlo simulations of truth model testing (TMT) and KF dynamic consistency chi-squared tests (NEES and NIS). Note, for both KFs, a Monte Carlo run of 50 samples were done since this provided enough variation in the TMT about the nominal trajectory. More Monte Carlo runs could have been performed, but for the sake of computational time 50 runs were sufficient. The confidence value α used in both KFs is 0.05. This value of α was chosen since it is the typical confidence value for most statistical analyses.

4. Linearized Kalman Filter Tuning

To begin tuning the LKF, we first had to perform some TMT that was representative of the nonlinear dynamics of the system. For LKF TMT, we chose a "truth" state perturbation and covarince in which each perturbation about the nominal trajectory would be generated from. This "truth" state perturbation and covariance were chosen after using various other combinations. This perturbation about the nominal trajectory was generated using Matlab's mvnrnd() function, and was generated for each Monte Carlo run.

$$\delta\mathbf{x}_{truth} = \begin{bmatrix} 0km \\ 0.075km/s \\ 0km \\ -0.021km/s \end{bmatrix} \tag{31}$$

$$\mathbf{P}_{truth} = \begin{bmatrix} 0.001 & 0 & 0 & 0 \\ 0 & 0.0001 & 0 & 0 \\ 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0.0001 \end{bmatrix} \tag{32}$$

After generating the perturbations about the nominal trajectory, the perturbations were added to the nominal initial condition. This new initial state for the TMT was propagated forward in time using the nonlinear dynamics of the system.

Since this TMT needs to be representative of actual data, Q_{true} was used to model the noise that would appear in the system. The noise was generated using Matlab's mvnrnd() function. Matlab's ode45() function was called for each time step passing in the state from the previous time step but adding the AWGN to the state (specifically **Xdot** and **Ydot**).

After generating the TMT states or ground truth data, came generating the noisy measurements that could be seen by the ground stations using the ground truth data. The noisy measurements were generated in a similar manner to the ground truth data, used Matlab's mvnrnd() function to generate the process noise which was then added to the nonlinear computation of the ground truth states.

After generating the noisy ground truth states and noisy measurements, we began to tune the LKF. The LKF function, based on lecture 24 and 25, takes in an initial guess for the perturbations and state covariance, observation data, time step, process noise tuning parameter Q_{LKF}, measurement noise matrix R_{LKF}, Γ matrix (same as in Part 1), tracking station's state history, and the nominal state history. The initial guess for the state is the same as $\delta\mathbf{x}_{truth}$ since this is what the LKF TMT were based on. The initial state covariance for the system was 10^6 times larger than the $\mathbf{P}_{truth}$. This was done to allow variance in the initial state so that the true initial state that generated the observation data, could be within the scope of the LKF initialization.

A typical simulation of the the noisy ground truth state is shown below.

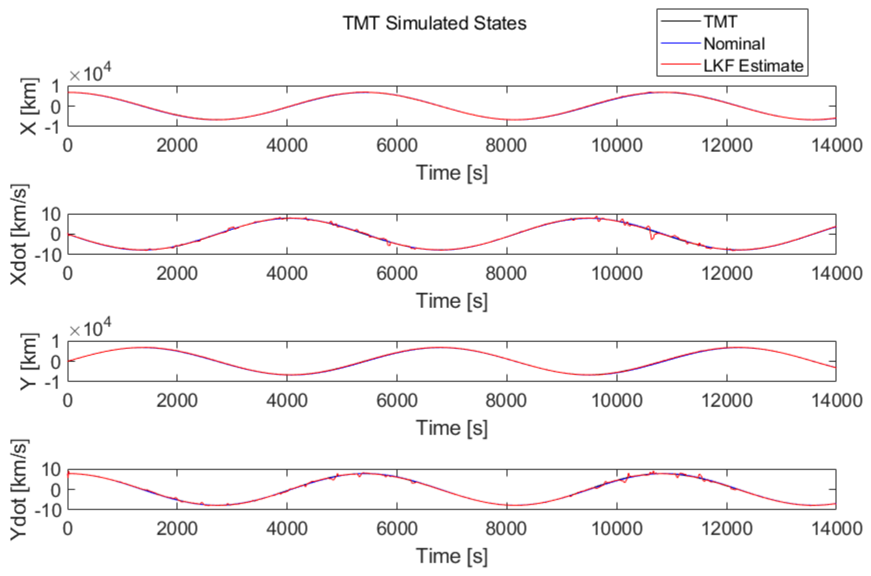

Fig. 5 Typical Simulation of Noisy Ground Truth States

The black line shows the TMT (ground truth) states, the blue line shows the Nominal linearized state, and the red line shows the LKF estimation of the states. It can be seen that the TMT ground truth and nominal trajectorys overlap each other since TMT ground truth is based around the nominal state. The LKF estimation follows the TMT ground truth decently in during some time steps but not so well during other times. The LKF estimation noticeably deviates from the ground truth and the nominal states between 4000s and 6000s as well as around 10000s. This indicates that the LKF is not entirely tuned properly and/or there's an issue with implementing the code. This will be explained shortly.

Using the generated TMT ground truth states, the TMT noisy measurements were generated and are shown below.

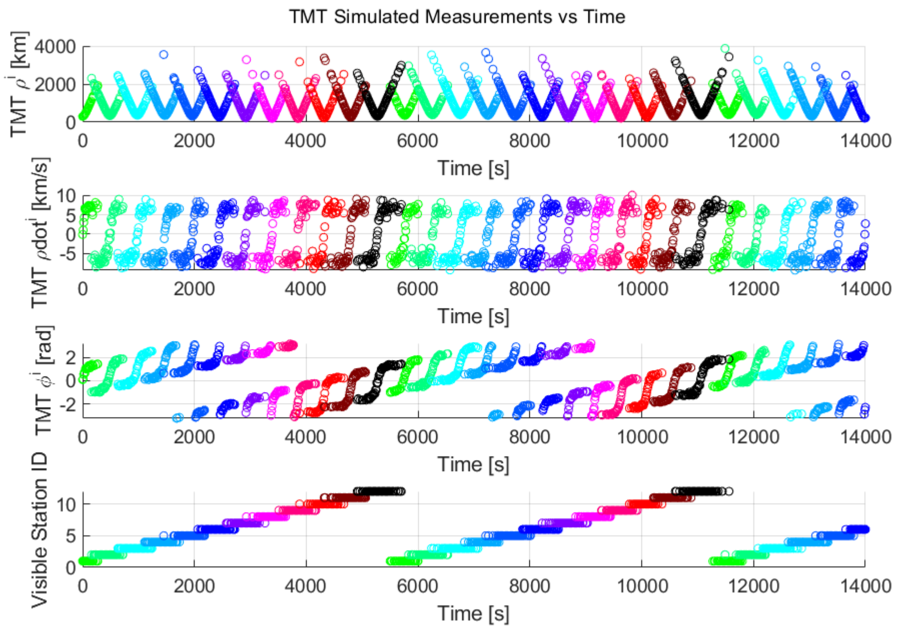

Fig. 6 Typical Simulation of Noisy Measurements

The TMT noisy measurements look like the nonlinear and linear measurements from part I. This indicates that the process of generating the TMT measurements was done correctly. The only issue with this figure, is that there appears to be inconsistent overlap of visible ground station observations. For example, the first time station 12 is seen (black circles), it overlaps with stations 11 and 1 but not consistently. This is mostly due to how the TMT measurements were stored and/or indexed.

It's seen in figure 6, that the LKF estimations jump around during certain times, thus looking at the LKF errors might help show why that is. The LKF state errors are the difference between the true state perturbations (TMT ground truth minus - nominal state trajectory) and the estimated state perturbations. A plot of the typical LKF state errors are shown in figure 8.

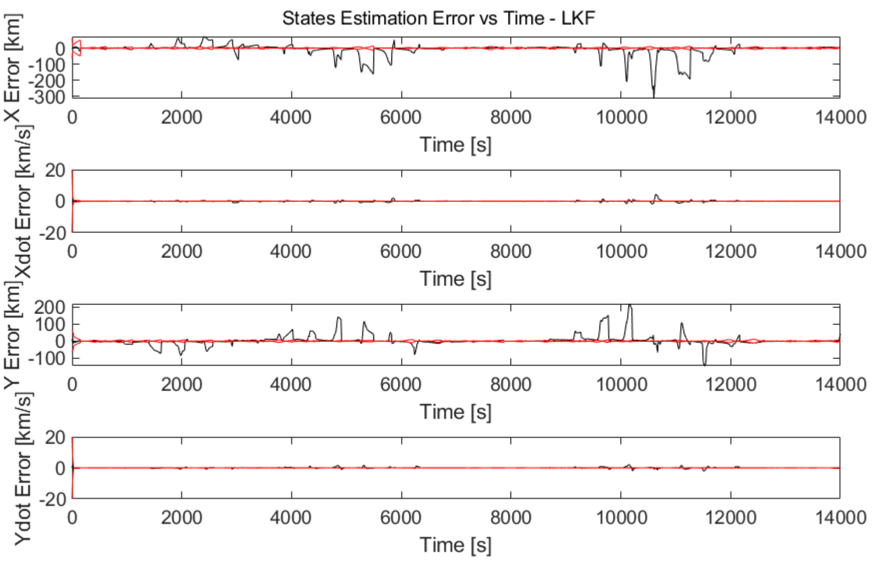

Fig. 7 Resulting LKF State Estimation Errors

In these LKF state estimation error plots, it is clearly seen that the LKF estimations deviate quite far from the true deviations. The errors, on the order of 100s of kilometers, is far too large for a LEO orbit such as this system since it could have the satellite impact the surface of the Earth (not ideal!). This indicates again that the LKF function and its parameters are not entirely correct.

Another thing to explore to see what could be wrong with the LKF is the innovations. The innovations are computed as the difference between the true measurement perturbations (TMT measurements - nominal observations) and the LKF estimated measurements. Figure 9 has the innovations vs time for a typical simulation.

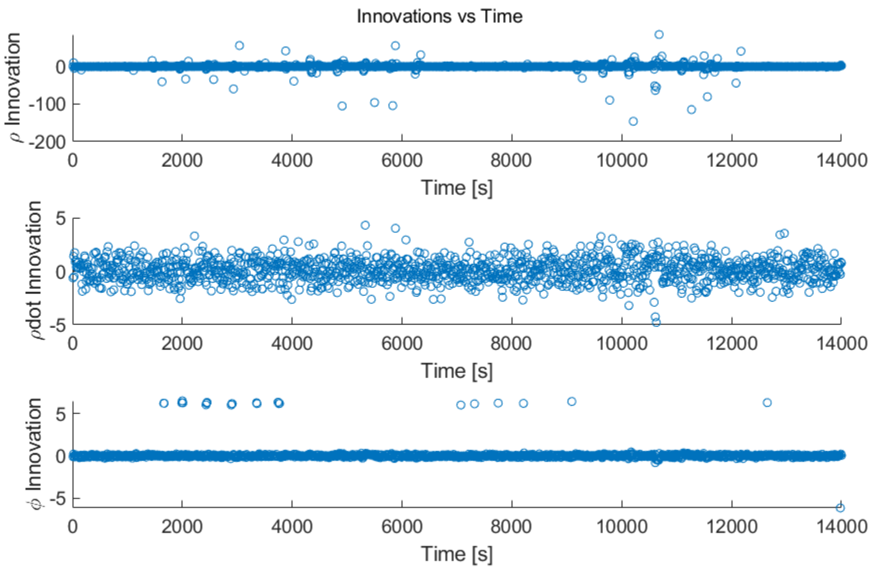

Fig. 8 Typical Simulation Innovations

From this figure, its seen that the innovations are quite off. With ρ having large innovations between 2000s and 6000s and again between 1000s and 12000s. While ρdot seems reasonable with bounds between -5 km/s and 5 km/s.

The ϕ innovations are quite large between 2000s and 4000s, 7000s and 1000s, and a couple other times as well. The seemingly random large innovations that are not at consistent times between the measurements gives more evidence that the LKF is not working correctly.

Even thought the perturbation errors and innovations were quite high, the best way to tell if the LKF was implemented correctly is with the NEES and NIS consistency chi-squared tests. These tests look at the perturbation errors ($e_{x,k}$) and innovations ($e_{y,k}$) are chi-squared distributed. The method for computing the NEES and NIS parameters were given in lecture 22 and 23. Their formulas are given below.

$$NEES = \epsilon_{x,k} = e_{x,k}^T (P_k^+)^{-1} e_{x,k} \tag{33}$$

$$NIS = \epsilon_{y,k} = e_{y,k}^T (S_k)^{-1} e_{y,k} \tag{34}$$

The NEES and NIS parameters are computed at each time step for each Monte Carlo run and put inside a matrix. The size of the matrix is N by k, where N is the number of Monte Carlo runs (50 in this case) and k is the number of steps (1400 for this system). The consistency test looks at the average of the NEES and NIS values at each step across all N runs. If $(1-\alpha) * 100\%$ of these averaged values fall within r_1 and r_2 bounds, then the given LKF is working correctly. These r_1 and r_2 bounds are the inverse cumulative distribution function of $\alpha/2$ and $1-\alpha/2$ (respectively) of the chi-squared distribution with $N * n$ (NEES) or $N * p$ (NIS) degrees of freedom normalized for N runs.

If the LKF was working correctly (not large errors), the main tuning parameter to have the LKF pass the consistency tests is Q_{LKF}. The truth perturbations and covariance also play key roles in the performance of the LKF, but it is assumed they are ideal. Since the Q_{true} for the system is quite small, the Q_{LKF} must be quite larger than the truth to allow for variations of the LKF estimation to capture the truth data. For this LKF, Q_{LKF} was 1000 times the size of Q_{true}.

Using the above method for computing the NEES consistency test using the TMT noisy ground truth, noisy measurements, and LKF itself, the NEES and NIS test results are below. The r1 and r2 bounds for the $50 * 4$ degrees of freedom normalized for $N = 50$ Monte Carlo runs are 3.2546 and 4.8212 respectively.

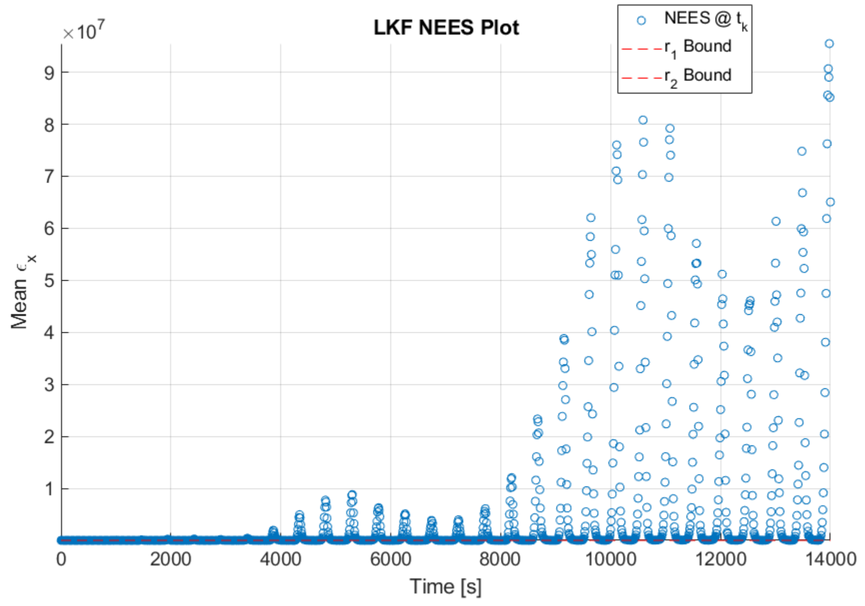

Fig. 9 LKF Consistency Plot - NEES

Out right, it is seen that the LKF does not pass the NEES consistency test. The r1 and r2 lines are barley visible on figure 10 since the mean NEES values blow up to the 10^7 magnitude. This is the biggest indicator that the LKF is not tuned nor working correctly. The biggest sources of errors could be how the perturbations errors and smug Kalman gain that minimizes the state covariance matrix. A combination of large errors with small covariances could blow up the NEES parameter.

Using the above method for computing the NIS consistency test using the TMT noisy ground truth, noisy measurements, and LKF itself, the NEES and NIS test results are below. The r1 and r2 bounds for the $50 * 3$ degrees of freedom normalized for $N = 50$ Monte Carlo runs are 2.3597 and 3.7160 respectively.

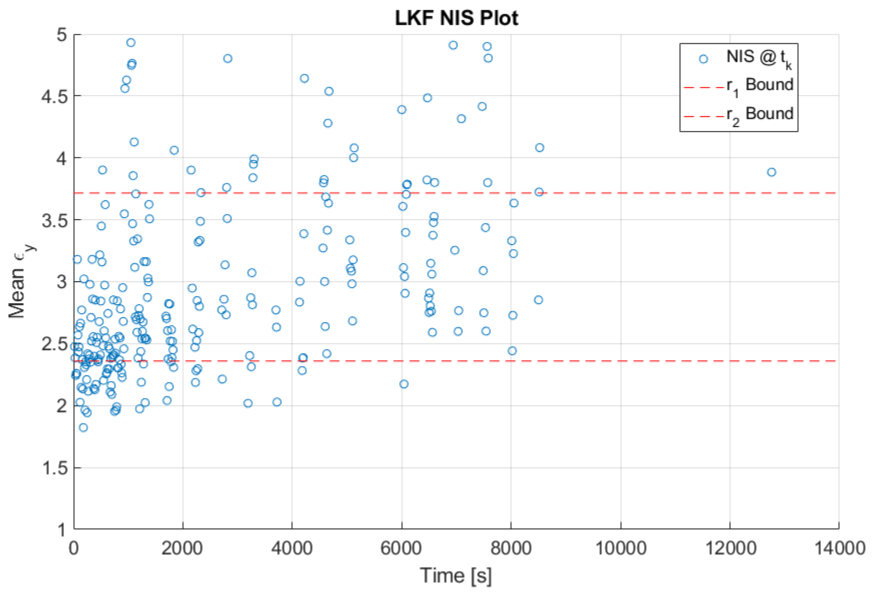

Fig. 10 LKF Consistency Plot - NIS

The NIS consistency test is slightly better than the NEES test, even if only for the 8000s. After that, the NIS values blow up like the NEES values. Again this is most likely due to the large innovation values, smug Kalman filter, or an incorrect LKF.

Multiple methods were attempted to correct the above seen errors with the LKF and consistency tests. These methods include fudging the Kalman gain with the LKF to not be so smug during the trouble time sets, limiting the perturbations deviation from the known truth perturbation, and going to office hours to discuss methodology and coding practices. Unfortunately, no method was able to correct the above LKF bugs. Other methods that could be done in the future to fix and properly tune the LKF is to start from scratch (code wise) and paying closer attention to code implementation. This LKF can still be physically used to estimate the state of a system, but the validity of the state estimations is low.

5. Extended Kalman Filter Tuning

Whereas the Linearized Kalman Filter uses a nominal state estimation to linearly propagate a perturbation, the Extended Kalman Filter instead uses a corrected state estimation produced from the full nonlinearized dynamics at each time step to propagate a disturbance. The dynamics are then linearized about this corrected state estimation.

Similar to the sequence of tuning the LKF, we must perform some TMT representative of the nonlinear dynamics of the system. We chose a "truth" state matrix and covariance from which each initial condition would be generated. These I.C.s were generated using MATLAB's mvnrnd() function for each Monte Carlo run, and were then propagated forward in time using the nonlinear dynamics of the system. To do so, MATLAB's ode45() function was used at each time step, adding AWGN to the previous state before passing it to ode45().

$$\mathbf{x}_{truth} = I.C.s = \begin{bmatrix} 6678km \\ 0km/s \\ 0km \\ 7.7258km/s \end{bmatrix} \tag{35}$$

$$\mathbf{P}_{truth} = \begin{bmatrix} 0.05 & 0 & 0 & 0 \\ 0 & 0.00025 & 0 & 0 \\ 0 & 0 & 0.05 & 0 \\ 0 & 0 & 0 & 0.00025 \end{bmatrix} \tag{36}$$

Q_{true} was once again used to model the noise that would appear in the system to ensure the TMT was representative of actual data. The noise was generated using the mvnrnd() command, and the noisy measurements were generated similarly to the LKF process.

Once this was all completed, the EKF could be tuned. The EKF function takes in almost the exact same parameters as the LKF function, with three exceptions: instead of an initial guess for the perturbations of the state, it takes in the initial conditions of the system, $\mathbf{x}_{truth}$; it takes in the time vector; and it does not take in the nominal state history. The initial state covariance guess for the system was roughly 10^2 times larger than $\mathbf{P}_{truth}$ for the same reason as in the LKF.

A typical simulation of the noisy ground truth state is shown below.

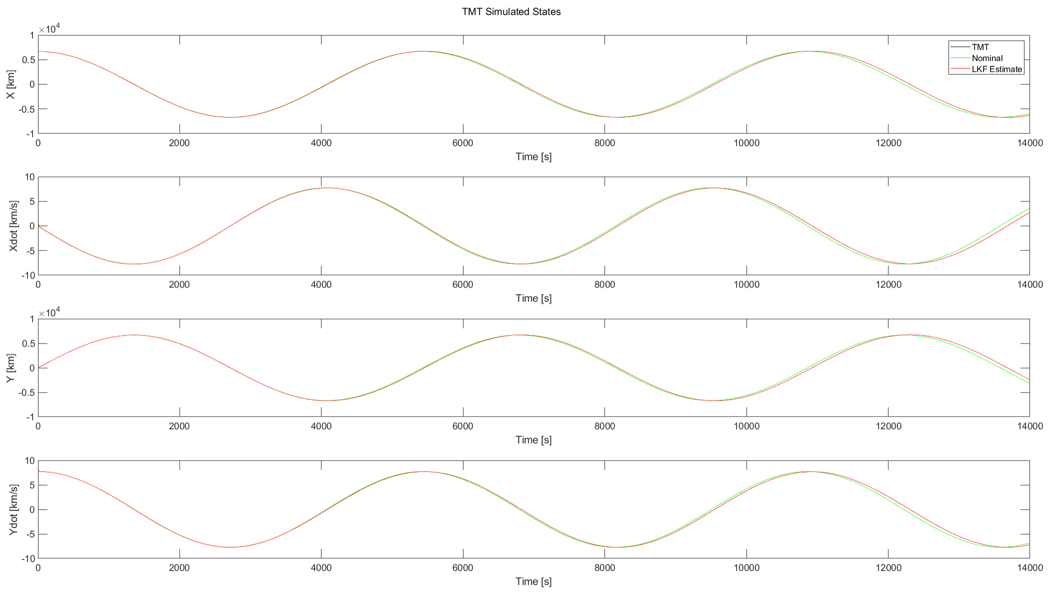

Fig. 11 Typical Simulation of Noisy Ground Truth States

The black line shows the TMT (ground truth) states, the green line the nominal linearized trajectory, and the red line the EKF estimate. The EKF follows the TMT ground truth remarkably well and is visually indistinguishable. This is a good first indicator that the EKF is tuned properly.

Next, the TMT noisy measurements were generated and are show below.

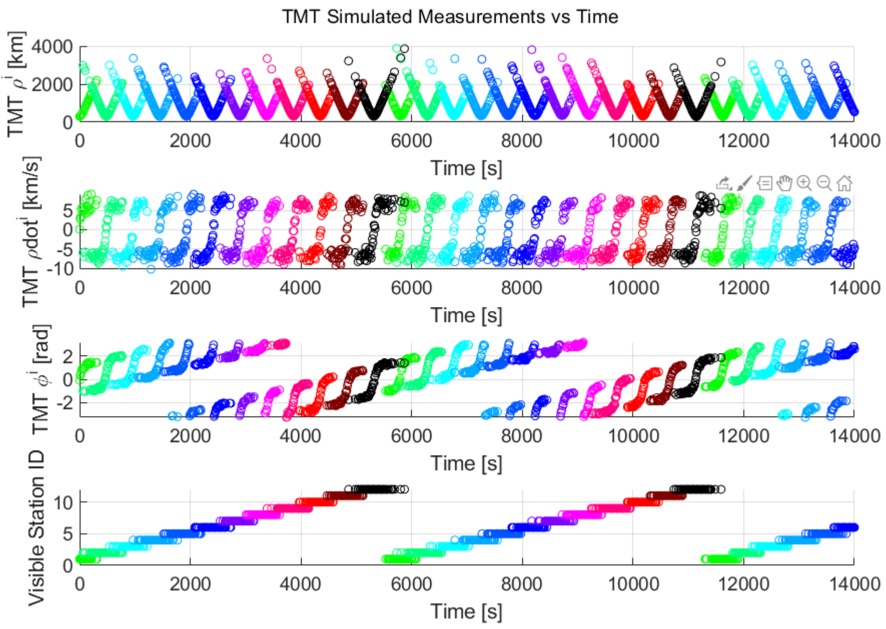

Fig. 12 Typical Simulation of Noisy Measurements

The results are visually similar to both the LKF noisy measurements plot and the measurements from part I. To further confirm that the EKF is tuned correctly, the estimation error for each state is plotted below.

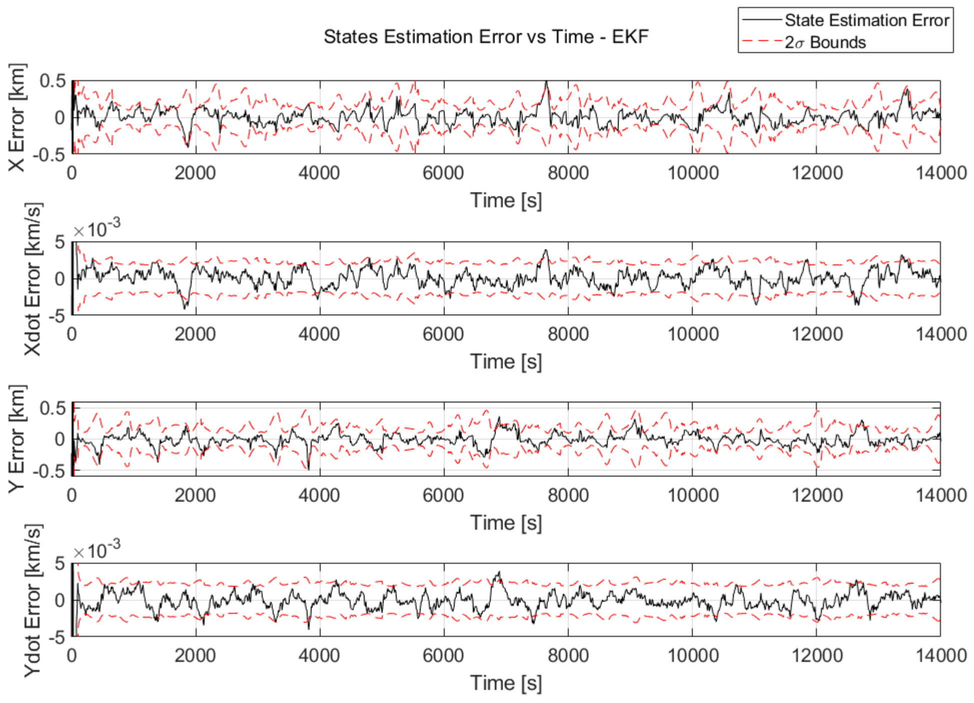

Fig. 13 Resulting EKF State Estimation Errors

It is apparent that the EKF, with ρ errors on the order of a few hundred meters giving a percent error of, at maximum, 0.16%, is much more capable of predicting the orbit than the LKF.

The previous plots have indicated that the EKF is tuned properly and giving good results. However, like with the LKF, the best way to tell if the EKF was implemented correctly is with NEES and NIS consistency chi-squared tests. These tests are accomplished using Eqns. 33 and 34, respectively, and are stored identically to the LKF described earlier.

The NEES and NIS consistency tests are calculated using the TMT noisy ground truth, noisy measurements, and the EKF. For the NEES test plotted below, the r1 and r2 bounds are the same for the LKF: $r1 = 3.2546$ and $r2 = 4.8212$.

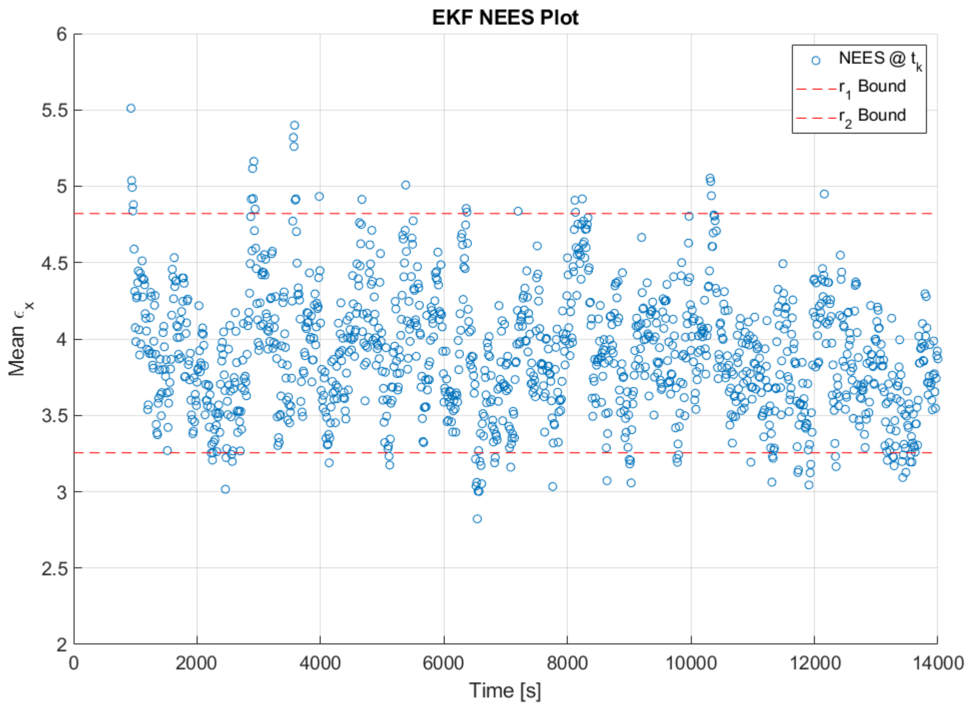

Fig. 14 EKF Consistency Plot - NEES

In this case, the majority of the values do fall between the r1 and r2 bounds; another good indicator that the EKF is working correctly.

The NIS test results are plotted below, with $r1 = 2.3597$ and $r2 = 3.7160$.

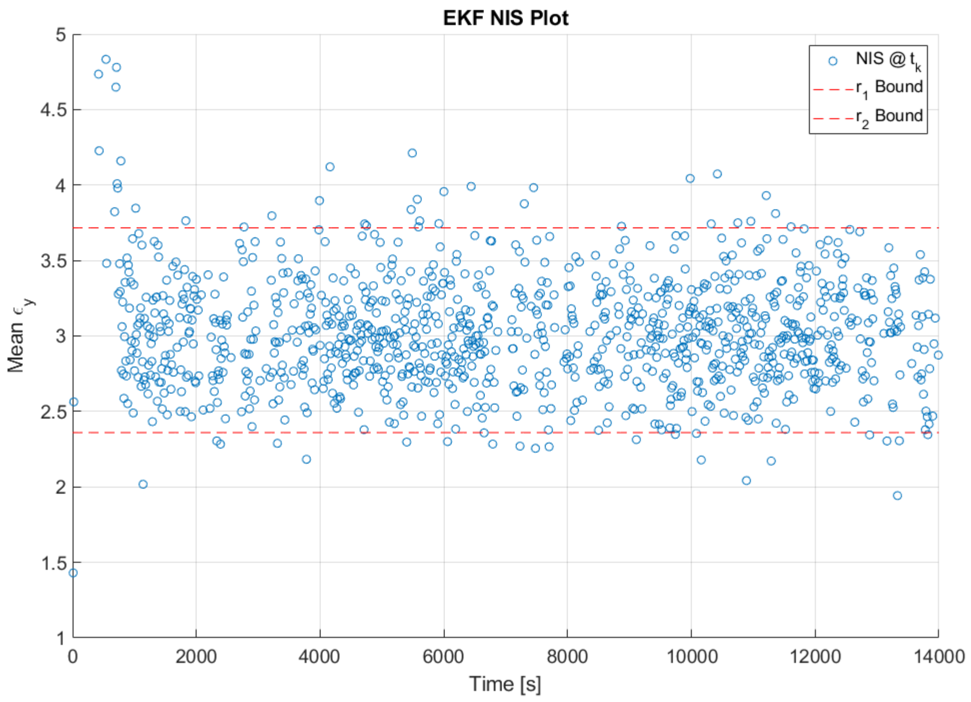

Fig. 15 EKF Consistency Plot - NIS

Once again, the majority of the values fall between r1 and r2.

These results all indicate that the EKF is correctly tuned, is not too smug, and is able to update it's state estimation without latching onto an incorrect trajectory or holding onto an old trajectory. To allow this, however, Q_{EKF} would be expected to be larger than Q_{true} in order for the EKF estimation to be able to capture the truth data. Interestingly, this is not the case: Q_{EKF} is roughly 95% Q_{true}.

6. Implementation of LKF and EKF for Provided Data

The provided observation data was given to the broken LKF and tuned EKF to estimate the state of the system that produced the observation data. The observation data is graphed in the figure below.

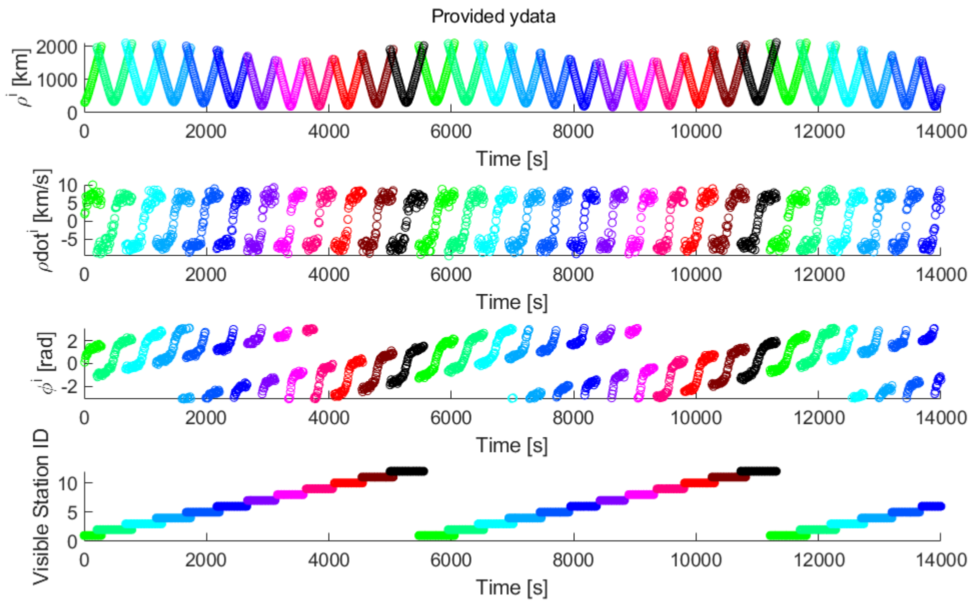

Fig. 16 Provided Observation Data for LKF and EKF Implementation

Passing the provided observation data to the broken LKF yields the below estimated state history of the system.

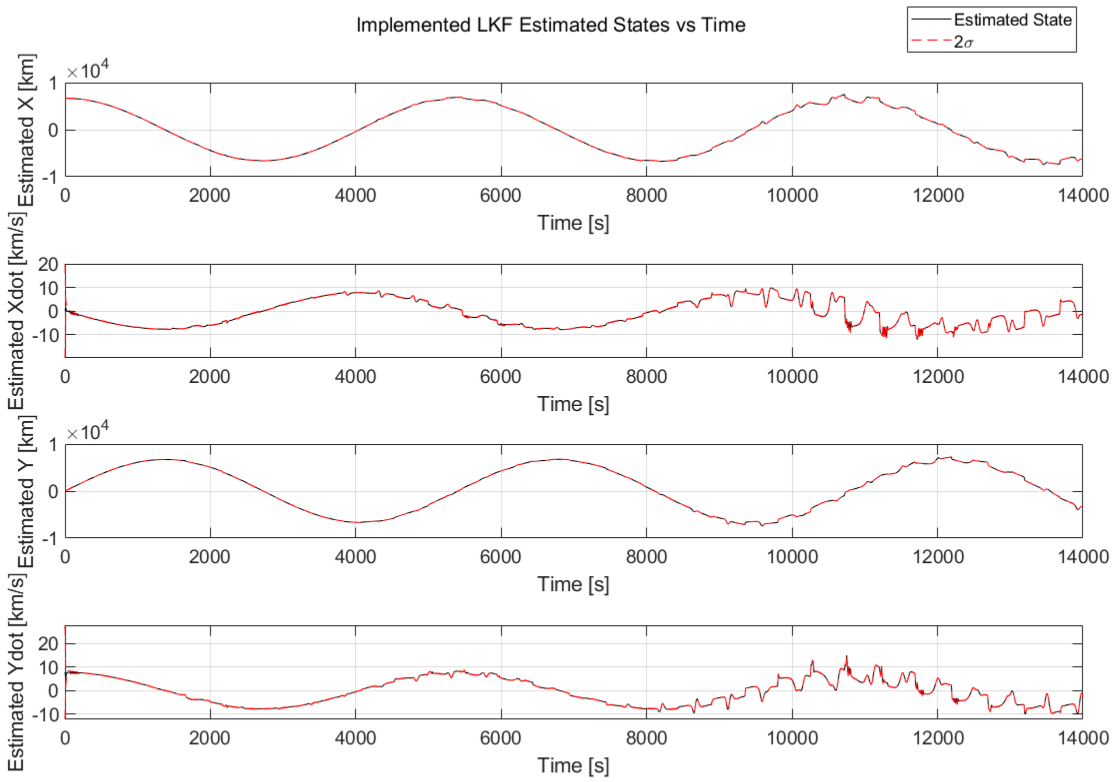

Fig. 17 LKF States Estimation using Observation Data

The LKF appears stable for the first 2000s, and then it breaks down with jumpy estimations. These spikes or jumps are quite noticeable after 8000s for the estimated Xdot and Ydot states. The table below shows sampled LKF estimated states and their standard deviations. This will be used to compare numerically with the EKF implementation.

Table 1 Sampled LKF State and Standard Deviation Estimations

Time [s]	0	3500	7000	10500	14000
X [km]	6678	-3923	-1786	5650	-6243
X STD [km]	31.6228	4.2083	0.0075i	5.8616	0.5893
Xdot [km/s]	0.075	6.1576	-7.23	0.6698	-0.7808
Xdot STD [km/s]	10	0.0403	0.0072	0.0523	0.0379
Y [km]	0	-5292	6419	-2112	-3412
Y STD [km]	31.6228	2.3076	0.6733	0.1786	1.5377
Ydot [km/s]	7.7048	-4.7526	-2.3466	4.0636	-2.6857
Ydot STD [km/s]	10	0.0385	0.0342	0.0196	0.0258

Passing the provided observation data to the tuned EKF yields the below estimated state history of the system.

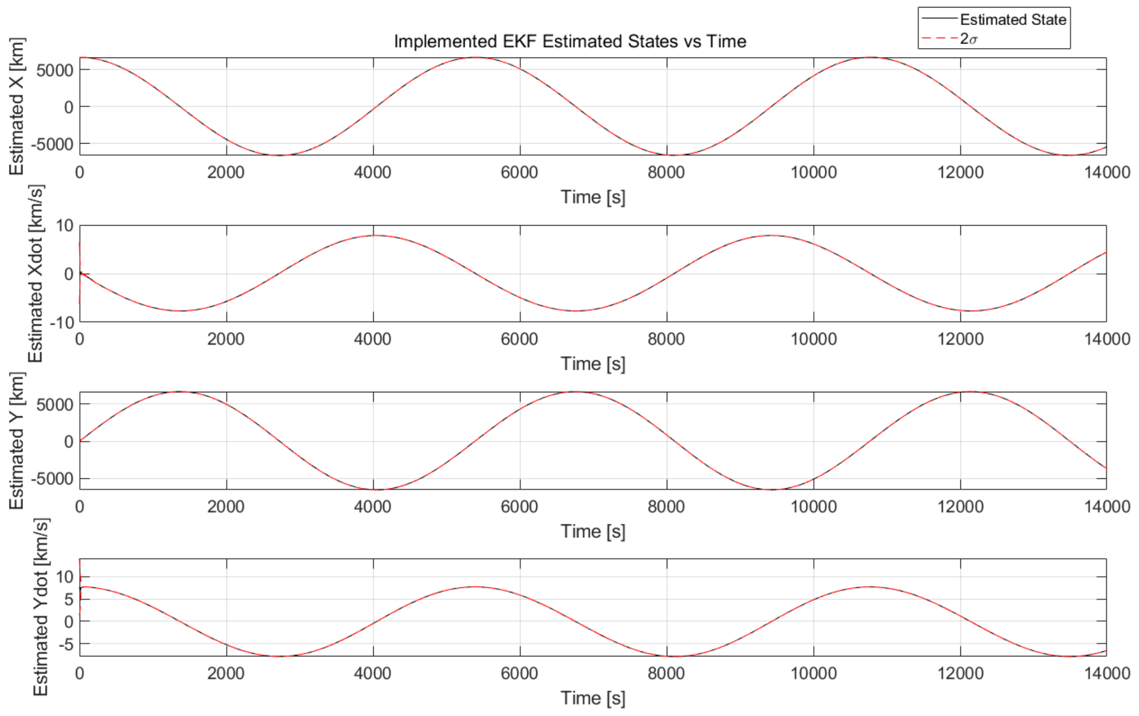

Fig. 18 EKF States Estimation using Observation Data

From figure 19, it's seen that the EKF estimated state history for the provided observation data is smooth with no noticeable spikes or jumps. Considering that the EKF passed both the NEES and NIS consistency tests, the validity of the EKF is high. The table below shows sampled EKF estimated states and their standard deviations. This will be used to compare numerically with the LKF implementation.

Table 2 Sampled EKF State and Standard Deviation Estimations

Time [s]	0	3500	7000	10500	14000
X [km]	6678	-3975	-1792	6361	-5408
X STD [km]	3.1623	0.1398	0.0436	0.1474	0.1288
Xdot [km/s]	0.075	6.1857	-7.3922	2.4259	4.4553
Xdot STD [km/s]	3.1623	0.0013	0.009	0.0011	0.0014
Y [km]	0	-5278	6414	-2030	-3725
Y STD [km]	3.1623	0.0991	0.0958	0.0858	0.0529
Ydot [km/s]	7.7048	-4.8128	-2.1411	7.3108	-6.4482
Ydot STD [km/s]	3.1623	0.0012	0.0011	0.0011	0.001

Comparing the figures, the EKF estimated states are smoother than the LKF, which makes sense, since the LKF did not work correctly. Even comparing the sampled estimated states and their standard deviations, the EKF had overall lower standard deviations.

Due to the fact that the LKF was not working correctly, the EKF estimation of the state should be held as the superior Kalman filter for this scenario. If the LKF was tuned and working properly, we would still trust the EKF to do a better job of state estimation. This is because the LKF linearizes about a nominal trajectory and estimates the perturbations form the truth/nominal where as the EKF predicts the state using nonlinear dynamics but corrects its self using linearization about the truth. It is inherent that the EKF should more closely follow the truth than the LKF. But both Kalman filters are valid filters depending on the purpose and application of the estimated states.

III. Advanced Questions

AQ 13 - Limerick

There once was a class about estimation;
To pass it required immense dedication.
"A filter is smug? What is this class?
And yet, by the skin of our teeth we pass,"
Says our broken LKF estimation.

Acknowledgments

A special thanks to Professor McMahon and TA Morgan for assistance during the project as well as being excellent instructors for this course.

References

[1] Simon, Dan. *Optimal State Estimation: Kalman, H∞, and Nonlinear Approaches* Wiley Interscience, 2006.

[2] University of Colorado Boulder, Department of Aerospace Engineering. *"final proj assignment ASEN 5044.pdf"*. University of Colorado Boulder ASEN 5044 Statistical Estimation for Dynamical Systems, Fall 2021.

[3] University of Colorado Boulder, Department of Aerospace Engineering. *"OrbitDetermination-fall2021.pdf"*. University of Colorado Boulder ASEN 5044 Statistical Estimation for Dynamical Systems, Fall 2021.

[4] University of Colorado Boulder, Department of Aerospace Engineering. *"OrbitDeterm_Part1SolnSketch.pdf"*. University of Colorado Boulder ASEN 5044 Statistical Estimation for Dynamical Systems, Fall 2021.

MATLAB Code

Table of Contents

Admin 1
Housekeeping 1
Constants 1
Part 1 2
1c) Nonlinear Dynamics 3
1c) Linearized DT 6
perturbations plot 6
Lin MEasurements 7
Plotting section for report 10
Provided Ydata 11
Part II - LKF Tunning 13
TMT Plots and Errors 15
Part II - LKF Implementation 19
Part II - EKF Tunning 20
EKF Implementation 26
A Solver 27
C Solver 28
EKF StatOD 28
LKF StatOD 31
StatOD NonLinear Measurements 34
Stat OD NonLinear Noise ODE 34
StatOD NonLinear ODE 35
Vary Color 35

Admin

```
%{

Names: Corey LePine and William Watkins
Professor: McMahon
Class: ASEN 5044 Stat Est for Dyn Sys
Date: December 14, 2021
Final Project: Statistical Orbit Determination

%}
```

Housekeeping

```
clc; clear all; close all;
```

Constants

```
load('orbitdeterm_finalproj_KFdata.mat')
u = 398600; % Earth's standard gravitational paremters [km^3/s^2]
r0 = 6678; % Nominal orbit radius [km]
Re = 6378; % Uniform radius of Earth [km]
```

```
we = 2*pi/86400; % Consant rotation rate of Earth [rad/s]
vel = sqrt(u/r0); % orbital velocity
circ = 2*pi*r0; % circumference of the orbit
T = circ / vel; % Orbital period

n = 4; % number of states
m = 2; % number of inputs, also happens to be number of distrubances
p = 3; % number of measurements
j = 6; % number of measurement stations

dt = 10; % step size [s]
tspan = 0:dt:14000;
initCon = [r0, 0, 0, r0*sqrt(u/r0^3)]; % initial state point - LTV
 sys, so
% have to linearize about a nominal trajectory, see below

DEBUG = 0; % Debug boolean

ColorSet = varycolor(12); % 12 Unique Colors for the Tracking Stations
```

Part 1

```
% 1a) Find the CT Jacobian Matracies
syms x1 x2 x3 x4 mu u1 u2 w1 w2 z1 z2 z3 z4 t % zi is the states of
 the ground stations

f = [x2;
    -mu*x1/sqrt(x1^2 + x3^2)^3 + u1 + w1;
    x4;
    -mu*x3/sqrt(x1^2 + x3^2)^3 + u2 + w2];
state = [x1, x2, x3, x4];
inputs = [u1, u2];
disturb = [w1, w2];

A = jacobian(f, state);
B = jacobian(f, inputs);
Gam = jacobian(f, disturb);

h = [sqrt((x1 - z1)^2 + (x3 - z3)^2);
    ((x1 - z1)*(x2 - z2) + (x3 - z3)*(x4 - z4))/(sqrt((x1 - z1)^2 +
 (x3 - z3)^2));
    atan((x3 - z3)/(x1 - z1))];

C = jacobian(h, state);
D = jacobian(h, inputs);

% 1b) Linearize about Nominal operating points
for ii = 1:length(tspan)
    nomCon(:,ii) = [r0 * cos(sqrt(u/r0^3)*tspan(ii)); -r0 *
 sin(sqrt(u/r0^3)*tspan(ii))*sqrt(u/r0^3);
            r0*sin(sqrt(u/r0^3)*tspan(ii)); r0*cos(sqrt(u/
r0^3)*tspan(ii))*sqrt(u/r0^3)];
            % Have to linearize about nominal trajectory!
```

```
end
mu = u;
```

1c) Nonlinear Dynamics

State NL

```
Rel_Tol = 1e-13;
Abs_Tol = Rel_Tol;
options = odeset('Stats', 'off', 'RelTol', Rel_Tol, 'AbsTol',
 Abs_Tol);

perts = [0, 0.075, 0, -0.021];
Initial_States = perts + initCon;

[Time_out, State_out] = ode45(@(Time, State) StatODNL_ODE(Time,
 State), tspan, Initial_States, options);

State_X = State_out(:, 1);
State_Xdot = State_out(:, 2);
State_Y = State_out(:, 3);
State_Ydot = State_out(:, 4);
% Perturbations
State_X_Pert = State_out(:,1) - nomCon(1,:)';
State_Xdot_Pert = State_out(:,2) - nomCon(2,:)';
State_Y_Pert = State_out(:,3) - nomCon(3,:)';
State_Ydot_Pert = State_out(:,4) - nomCon(4,:)';

figure()
subplot(4, 1, 1)
plot(Time_out, State_out(:, 1), 'k')
% xlabel('Time [s]')
ylabel('X [km]')
set(gca, 'FontSize', 14)

subplot(4, 1, 2)
plot(Time_out, State_out(:, 2), 'k')
% xlabel('Time [s]')
ylabel('$\dot{X}$ [km/s]', 'Interpreter','latex')
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
plot(Time_out, State_out(:, 3), 'k')
% xlabel('Time [s]')
ylabel('Y [km]')
ylim([-1e4, 1e4])
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
plot(Time_out, State_out(:, 4), 'k')
xlabel('Time [s]')
ylabel('$\dot{Y}$ [km/s]', 'Interpreter','latex')
set(gca, 'FontSize', 14)
```

```
sgtitle('States vs Time, Full Nonlinear Dynamics Simulation')

% Tracking Stations Positions
TS_IDS = 1:1:12; % tracking stations ids
theta_TS0 = (TS_IDS - 1)*pi/6; % tracking stations intial positions

for ii = 1:12 % tracking station X and Y position
    TS_X(:, ii) = Re*cos(we*Time_out + theta_TS0(ii));
    TS_Xdot(:, ii) = -Re*we*sin(we*Time_out + theta_TS0(ii));
    TS_Y(:, ii) = Re*sin(we*Time_out + theta_TS0(ii));
    TS_Ydot(:, ii) = Re*we*cos(we*Time_out + theta_TS0(ii));
    theta_TS(:, ii) = atan2(TS_Y(:, ii), TS_X(:, ii));
end

thetaCompare = theta_TS;

% Measurement NL
for ii = 1:12
    rho(:, ii) = sqrt((State_X - TS_X(:, ii)).^2 + (State_Y - TS_Y(:,
 ii)).^2);
    rho_dot(:, ii) = ((State_X - TS_X(:, ii)).*(State_Xdot -
 TS_Xdot(:, ii)) + (State_Y - TS_Y(:, ii)).*(State_Ydot - TS_Ydot(:,
 ii)))./rho(:, ii);
    phi(:, ii) = atan2((State_Y - TS_Y(:, ii)), (State_X - TS_X(:,
 ii)));
    visibleStation(:,ii) = ones(length(tspan),1) * ii;
end

phiCompare = phi;

% Wrap the upper and lower bounds between -pi and pi
% When the upper bound is above pi, need to wrap both bound down to -
pi
% Vice versa when lower bound is
thetaBound1Pos = theta_TS;
thetaBound1PosInd = find(thetaBound1Pos+pi/2 > pi);
thetaBound1Pos(thetaBound1PosInd) = thetaBound1Pos(thetaBound1PosInd)
 - 2*pi;
thetaBound1Neg = theta_TS;
thetaBound1Neg(thetaBound1PosInd) = thetaBound1Neg(thetaBound1PosInd)
 - 2*pi;

thetaBound2Neg = theta_TS;
thetaBound2NegInd = find(thetaBound2Neg-pi/2 < pi);
thetaBound2Neg(thetaBound2NegInd) = thetaBound2Neg(thetaBound2NegInd)
 + 2*pi;
thetaBound2Pos = theta_TS;
thetaBound2Pos(thetaBound2NegInd) = thetaBound2Pos(thetaBound2NegInd)
 + 2*pi;

% Visible Tracking Stations
figure('Position', [200, 200, 1200, 1000])
hold on
```

```
tl = tiledlayout(4,1);
title(tl, "Nonlinear Measurements");
xlabel(tl, "Time [s]");
nexttile
hold on
for ii = 1:12
    vis_index = find((phiCompare(:, ii) <= (pi/2 + thetaCompare(:,
 ii)) & phiCompare(:, ii) >= (-pi/2 + thetaCompare(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound1Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound1Neg(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound2Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound2Neg(:, ii))));
    if(DEBUG == 1)
        plot(Time_out, pi/2 + thetaBound1Pos(:,ii));
        hold on
        plot(Time_out, -pi/2 + thetaBound1Neg(:,ii));
        plot(Time_out, phi(:,ii));
        plot(Time_out, theta_TS(:,ii));
        scatter(Time_out(vis_index), phi(vis_index, ii))
        yline(pi);
        yline(-pi);
    end
    scatter(Time_out(vis_index), rho(vis_index,ii), [],
 ColorSet(ii, :));
    ylabel('\rho^i [km]');
    set(gca, 'FontSize', 14)
end

nexttile
hold on
for ii = 1:12
    vis_index = find((phiCompare(:, ii) <= (pi/2 + thetaCompare(:,
 ii)) & phiCompare(:, ii) >= (-pi/2 + thetaCompare(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound1Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound1Neg(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound2Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound2Neg(:, ii))));
    scatter(Time_out(vis_index), rho_dot(vis_index,ii), [],
 ColorSet(ii, :));
    ylabel('$\dot{rho}^i$ [km/s]', 'Interpreter','latex');
    set(gca, 'FontSize', 14)
end
nexttile
hold on
for ii = 1:12
    vis_index = find((phiCompare(:, ii) <= (pi/2 + thetaCompare(:,
 ii)) & phiCompare(:, ii) >= (-pi/2 + thetaCompare(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound1Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound1Neg(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound2Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound2Neg(:, ii))));
    scatter(Time_out(vis_index), phi(vis_index,ii), [],
 ColorSet(ii, :));
    ylabel('\phi^i [rads]');
    set(gca, 'FontSize', 14)
end
nexttile
hold on
for ii = 1:12
    vis_index = find((phiCompare(:, ii) <= (pi/2 + thetaCompare(:,
 ii)) & phiCompare(:, ii) >= (-pi/2 + thetaCompare(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound1Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound1Neg(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound2Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound2Neg(:, ii))));
    scatter(Time_out(vis_index), visibleStation(vis_index,ii), [],
 ColorSet(ii, :));
    ylabel('Visible Station ID');
    set(gca, 'FontSize', 14)
end
```

1c) Linearized DT

```
pertX(:, 1) = perts';
for ii = 2:1401
    x1 = nomCon(1,ii-1);
    x2 = nomCon(2,ii-1);
    x3 = nomCon(3,ii-1);
    x4 = nomCon(4,ii-1);
    Atil = Atil_Solver([x1, x2, x3, x4]);

    Ftil = eye(n) + dt*Atil;
    pertX(:, ii) = Ftil*pertX(:, ii-1);
end

LinX(1, :) = nomCon(1,:) + pertX(1, :);
LinX(2, :) = nomCon(2,:) + pertX(2, :);
LinX(3, :) = nomCon(3,:) + pertX(3, :);
LinX(4, :) = nomCon(4,:) + pertX(4, :);
```

perturbations plot

```
figure()
subplot(4, 1, 1)
plot(Time_out, pertX(1, :), 'k')
% xlabel('Time [s]')
ylabel('\delta X [km]')
set(gca, 'FontSize', 14)

subplot(4, 1, 2)
plot(Time_out, pertX(2, :), 'k')
% xlabel('Time [s]')
ylabel('$\dot{\delta X}$ [km/s]', 'Interpreter','latex')
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
```

```
plot(Time_out, pertX(3, :), 'k')
% xlabel('Time [s]')
ylabel('\delta Y [km]')
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
plot(Time_out, pertX(4, :), 'k')
xlabel('Time [s]')
ylabel('$\dot{\delta Y}$ [km/s]', 'Interpreter','latex')
set(gca, 'FontSize', 14)

sgtitle('Linearized Approx Perturbations vs Time')

% state plot
figure()
subplot(4, 1, 1)
plot(Time_out, LinX(1, :), 'k')
% xlabel('Time [s]')
ylabel('X [km]')
set(gca, 'FontSize', 14)

subplot(4, 1, 2)
plot(Time_out, LinX(2, :), 'k')
% xlabel('Time [s]')
ylabel('$\dot{X}$ [km/s]', 'Interpreter','latex')
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
plot(Time_out, LinX(3, :), 'k')
% xlabel('Time [s]')
ylabel('Y [km]')
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
plot(Time_out, LinX(4, :), 'k')
xlabel('Time [s]')
ylabel('$\dot{Y}$ [km/s]', 'Interpreter','latex')
set(gca, 'FontSize', 14)

sgtitle('States vs Time, Linearized Approximate Dynamics Soluiton')
```

Lin MEasurements

```
for j = 1:12
    for i = 1:1401
        x(1) = nomCon(1,i);
        x(2) = nomCon(2,i);
        x(3) = nomCon(3,i);
        x(4) = nomCon(4,i);
        z(1) = TS_X(i,j);
        z(2) = TS_Xdot(i,j);
        z(3) = TS_Y(i,j);
        z(4) = TS_Ydot(i,j);
        Cnom = Ctil_Solver(x,z);

        H = Cnom;
        pertY(:, i) = H*pertX(:, i);
    end

    rhoLinPert(:,j) = pertY(1,:)';
    rho_dotLinPert(:,j) = pertY(2,:)';
    phiLinPert(:,j) = pertY(3,:)';

    rhoNom(:, j) = sqrt((nomCon(1,:)' - TS_X(:, j)).^2 + (nomCon(3,:)'
 - TS_Y(:, j)).^2);
    rho_dotNom(:, j) = ((nomCon(1,:)' - TS_X(:, j)).*(nomCon(2,:)'
 - TS_Xdot(:, j)) + (nomCon(3,:)' - TS_Y(:, j)).*(nomCon(4,:)' -
 TS_Ydot(:, j)))./rhoNom(:, j);
    phiNom(:, j) = atan2((nomCon(3,:)' - TS_Y(:, j)), (nomCon(1,:)' -
 TS_X(:, j)));

    rhoLinNom(:,j) = rhoNom(:,j) + rhoLinPert(:,j);
    rhoDotLinNom(:,j) = rho_dotNom(:,j) + rho_dotLinPert(:,j);
    phiLinNom(:,j) = phiNom(:,j) + phiLinPert(:,j);

    findAbovePiRho = find(rhoLinNom(:,j) > pi);
    findAbovePiPhi = find(phiLinNom(:,j) > pi);
    findBelowPiRho = find(rhoLinNom(:,j) < -pi);
    findBelowPiPhi = find(phiLinNom(:,j) < -pi);

%     rhoLinNom(findAbovePiRho,j) = rhoLinNom(findAbovePiRho,j) -
 2*pi;
    phiLinNom(findAbovePiPhi,j) = phiLinNom(findAbovePiPhi,j) - 2*pi;
%     rhoLinNom(findBelowPiRho,j) = rhoLinNom(findBelowPiRho,j) +
 2*pi;
    phiLinNom(findBelowPiPhi,j) = phiLinNom(findBelowPiPhi,j) + 2*pi;
end


figure('Position', [200, 200, 1200, 1000])
hold on
tl = tiledlayout(4,1);
title(tl, "Linearized Measurements");
xlabel(tl, "Time [s]");
nexttile
hold on
ylim([0 2500]);
DEBUG = 0;
for ii = 1:12
    vis_index = find((phiCompare(:, ii) <= (pi/2 + thetaCompare(:,
 ii)) & phiCompare(:, ii) >= (-pi/2 + thetaCompare(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound1Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound1Neg(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound2Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound2Neg(:, ii))));
    if(DEBUG == 1)
%         plot(Time_out, pi/2 + thetaBound1Pos(:,ii));
% %         hold on
%         plot(Time_out, -pi/2 + thetaBound1Neg(:,ii));
        plot(Time_out, phiLin(:,ii));
        plot(Time_out, theta_TS(:,ii));
        scatter(Time_out(vis_index), phiLin(vis_index, ii))
        yline(pi);
        yline(-pi);
    end
     scatter(Time_out(vis_index), rhoLinNom(vis_index,ii));
    ylabel('\rho^i [km]');
end

nexttile
hold on
for ii = 1:12
%     vis_index = find((phiLin(:, ii) <= (pi/2 + thetaCompare(:, ii))
 & phiLin(:, ii) >= (-pi/2 + thetaCompare(:, ii))) | ...
%             (phiLin(:, ii) <= (pi/2 + thetaBound1Pos(:, ii)) &
 phiLin(:, ii) >= (-pi/2 + thetaBound1Neg(:, ii))) | ...
%             (phiLin(:, ii) <= (pi/2 + thetaBound2Pos(:, ii)) &
 phiLin(:, ii) >= (-pi/2 + thetaBound2Neg(:, ii))));
    vis_index = find((phiCompare(:, ii) <= (pi/2 + thetaCompare(:,
 ii)) & phiCompare(:, ii) >= (-pi/2 + thetaCompare(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound1Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound1Neg(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound2Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound2Neg(:, ii))));
    scatter(Time_out(vis_index), rhoDotLinNom(vis_index,ii));
    ylabel('$\dot{\rho}^i$ [km/s]', 'Interpreter','latex');
end
nexttile
hold on
for ii = 1:12
    vis_index = find((phiCompare(:, ii) <= (pi/2 + thetaCompare(:,
 ii)) & phiCompare(:, ii) >= (-pi/2 + thetaCompare(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound1Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound1Neg(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound2Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound2Neg(:, ii))));
    scatter(Time_out(vis_index), phiLinNom(vis_index,ii));
    ylabel('\phi^i [rads]');
end
nexttile
hold on
for ii = 1:12
    vis_index = find((phiCompare(:, ii) <= (pi/2 + thetaCompare(:,
 ii)) & phiCompare(:, ii) >= (-pi/2 + thetaCompare(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound1Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound1Neg(:, ii))) | ...
            (phiCompare(:, ii) <= (pi/2 + thetaBound2Pos(:, ii)) &
 phiCompare(:, ii) >= (-pi/2 + thetaBound2Neg(:, ii))));
    scatter(Time_out(vis_index), visibleStation(vis_index,ii));
    ylabel('Visible Station ID');
end
```

Plotting section for report

We will plot the states on top of one another, as well as the measurements

```
figure('Position', [200, 200, 1600, 1000])
t1 = tiledlayout(4,1);
title(t1, "Simulated System State Perturbations");
xlabel(t1, "Time [s]");

nexttile;
hold on;
plot(Time_out, State_X_Pert, 'k')
plot(Time_out, pertX(1,:), 'r');
ylabel('$\delta X$ [km]', 'Interpreter','latex')
set(gca, 'FontSize', 14)
legend('Nonlinear Perturbations', 'Linearized
 Perturbations','location','bestoutside');

nexttile;
hold on;
plot(Time_out, State_Xdot_Pert, 'k')
plot(Time_out, pertX(2,:), 'r');
ylabel('$\delta \dot{X}$ [km/s]', 'Interpreter','latex')
set(gca, 'FontSize', 14)

nexttile;
hold on;
plot(Time_out, State_Y_Pert, 'k')
plot(Time_out, pertX(3,:), 'r');
ylabel('$\delta Y$ [km]', 'Interpreter','latex')
set(gca, 'FontSize', 14)

ax1 = nexttile;
hold on;
plot(Time_out, State_Ydot_Pert, 'k')
plot(Time_out, pertX(4,:), 'r');
ylabel('$\delta \dot{Y}$ [km/s]', 'Interpreter','latex')
set(gca, 'FontSize', 14)

% System States next
figure('Position', [200, 200, 1600, 1000])
t1 = tiledlayout(4,1);
title(t1, "Simulated System Dynamics");
xlabel(t1, "Time [s]");

nexttile;
hold on;
plot(Time_out, State_X, 'k')
plot(Time_out, LinX(1,:), 'r');
ylabel('$X$ [km]', 'Interpreter','latex')
set(gca, 'FontSize', 14)
legend('Nonlinear Dynamics', 'Linearized
 Dynamics','location','bestoutside');
```

```
nexttile;
hold on;
plot(Time_out, State_Xdot, 'k')
plot(Time_out, LinX(2,:), 'r');
ylabel('$\dot{X}$ [km/s]', 'Interpreter','latex')
set(gca, 'FontSize', 14)

nexttile;
hold on;
plot(Time_out, State_Y, 'k')
plot(Time_out, LinX(3,:), 'r');
ylabel('$Y$ [km]', 'Interpreter','latex')
set(gca, 'FontSize', 14)

ax1 = nexttile;
hold on;
plot(Time_out, State_Ydot, 'k')
plot(Time_out, LinX(4,:), 'r');
ylabel('$\dot{Y}$ [km/s]', 'Interpreter','latex')
set(gca, 'FontSize', 14)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
```

Provided Ydata

```
Gam = [0, 0; 1, 0; 0 0; 0 1];
Omega = dt*Gam;

ydata_TS_ID = NaN*ones(2, 1401);
ydata_data = NaN*ones(2*p, 1401);
c = NaN*ones(1, 1401);

for ii = 1:1401
    if ~isempty(ydata{ii})
        [~, c(ii)] = size(ydata{ii});
        if c(ii) == 1
            ydata_TS_ID(1, ii) = ydata{ii}(4, 1);
            ydata_data(1:3, ii) = ydata{ii}(1:3, 1);
        elseif c(ii) == 2
            ydata_TS_ID(1, ii) = ydata{ii}(4, 1);
            ydata_data(1:3, ii) = ydata{ii}(1:3, 1);
            ydata_TS_ID(2, ii) = ydata{ii}(4, 2);
            ydata_data(4:6, ii) = ydata{ii}(1:3, 2);
        end
    else
        ydata_TS_ID(:, ii) = NaN*ones(2, 1);
        ydata_data(:, ii) = NaN*ones(6, 1);
    end

end
```

```
ydata_TS_ID(1, 1) = 1;

TS_state(:, :, 1) = TS_X;
TS_state(:, :, 2) = TS_Xdot;
TS_state(:, :, 3) = TS_Y;
TS_state(:, :, 4) = TS_Ydot;

figure()
subplot(4, 1, 1)
hold on
for ii = 1:12
    [~, ID_index_c] = find(ydata_TS_ID(1, :) == ii);
    scatter(tvec(ID_index_c), ydata_data(1, ID_index_c), [],
 ColorSet(ii, :))
    [~, ID_index_c] = find(ydata_TS_ID(2, :) == ii);
    scatter(tvec(ID_index_c), ydata_data(4, ID_index_c), [],
 ColorSet(ii, :))
end
hold off
xlabel('Time [s]')
ylabel('\rho^i [km]')
set(gca, 'FontSize', 14)

subplot(4, 1, 2)
hold on
for ii = 1:12
    [~, ID_index_c] = find(ydata_TS_ID(1, :) == ii);
    scatter(tvec(ID_index_c), ydata_data(2, ID_index_c), [],
 ColorSet(ii, :))
    [~, ID_index_c] = find(ydata_TS_ID(2, :) == ii);
    scatter(tvec(ID_index_c), ydata_data(5, ID_index_c), [],
 ColorSet(ii, :))
end
hold off
xlabel('Time [s]')
ylabel('\rhodot^i [km/s]')
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
hold on
for ii = 1:12
    [~, ID_index_c] = find(ydata_TS_ID(1, :) == ii);
    scatter(tvec(ID_index_c), ydata_data(3, ID_index_c), [],
 ColorSet(ii, :))
    [~, ID_index_c] = find(ydata_TS_ID(2, :) == ii);
    scatter(tvec(ID_index_c), ydata_data(6, ID_index_c), [],
 ColorSet(ii, :))
end
hold off
xlabel('Time [s]')
ylabel('\phi^i [rad]')
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
```

```
hold on
for ii = 1:12
    [~, ID_index_c] = find(ydata_TS_ID(1, :) == ii);
    scatter(tvec(ID_index_c), ydata_TS_ID(1, ID_index_c), [],
 ColorSet(ii, :))
    [~, ID_index_c] = find(ydata_TS_ID(2, :) == ii);
    scatter(tvec(ID_index_c), ydata_TS_ID(2, ID_index_c), [],
 ColorSet(ii, :))
end
hold off
xlabel('Time [s]')
ylabel('Visible Station ID')
set(gca, 'FontSize', 14)

sgtitle('Provided ydata')
```

Part II - LKF Tunning

```
N = 50;

dx_true = [0, 0.075, 0, -0.021];
Px_true = 1e4*diag([0.1, 0.01, 0.1, 0.01]);
qp = 1e-6;

for jj = 1:N
    % TMT
    perts = mvnrnd(dx_true, qp*Px_true);

    MC_Initial_State = perts + initCon;

    TMT_X(1) = MC_Initial_State(1);
    TMT_Xdot(1) = MC_Initial_State(2);
    TMT_Y(1) = MC_Initial_State(3);
    TMT_Ydot(1) = MC_Initial_State(4);

    v = mvnrnd([0, 0, 0], Rtrue)';
    TMT_y_NL_out(:, 1) = StatOD_NLMeasurement([TMT_X(1), TMT_Xdot(1),
 TMT_Y(1), TMT_Ydot(1)], [TS_X(1, 1), TS_Xdot(1, 1), TS_Y(1, 1),
 TS_Ydot(1, 1)]);
    TMT_y_NL_noise_out(:, 1) = TMT_y_NL_out(:, 1) + v;
    TMT_ydata(1) = {[TMT_y_NL_noise_out(:, 1); ydata_TS_ID(1)]};
    TS_ID = NaN*ones(1401, 1);
    TS_ID(1) = ydata_TS_ID(1);


    for ii = 1:1400
        % Noise
        w = mvnrnd([0, 0], Qtrue);
        v = mvnrnd([0, 0, 0], Rtrue)';

        % State
        ODE45_InitialState = [TMT_X(ii), TMT_Xdot(ii), TMT_Y(ii),
 TMT_Ydot(ii), w(1), w(2)];
```

```
        [~, TMT_test] = ode45(@(Time, State) StatODNL_noise_ODE(Time,
State), [tvec(ii) tvec(ii+1)], ODE45_InitialState, options);

        TMT_X(ii+1) = TMT_test(end, 1);
        TMT_Xdot(ii+1) = TMT_test(end, 2);
        TMT_Y(ii+1) = TMT_test(end, 3);
        TMT_Ydot(ii+1) = TMT_test(end, 4);

        % Measurment
        for kk = 1:12 % compute measurements for each ground station
            yi = StatOD_NLMeasurement([TMT_X(ii+1), TMT_Xdot(ii+1),
TMT_Y(ii+1), TMT_Ydot(ii+1)], [TS_X(ii+1, kk), TS_Xdot(ii+1, kk),
TS_Y(ii+1, kk), TS_Ydot(ii+1, kk)]);
            TMT_y_ALL(ii, kk, 1) = yi(1) + v(1); % rho
            TMT_y_ALL(ii, kk, 2) = yi(2) + v(2); % rhodot
            TMT_y_ALL(ii, kk, 3) = yi(3) + v(3); % phi
        end
    end
    phiCompare = TMT_y_ALL(:, :, 3);

    for kk = 1:12 % compute the current visible ground station
        vis_index = find((phiCompare(:, kk) <= (pi/2 +
thetaCompare(2:end, kk)) & phiCompare(:, kk) >= (-pi/2 +
thetaCompare(2:end, kk))) | ...
            (phiCompare(:, kk) <= (pi/2 + thetaBound1Pos(2:end, kk)) &
phiCompare(:, kk) >= (-pi/2 + thetaBound1Neg(2:end, kk))) | ...
            (phiCompare(:, kk) <= (pi/2 + thetaBound2Pos(2:end, kk)) &
phiCompare(:, kk) >= (-pi/2 + thetaBound2Neg(2:end, kk))));

        TMT_y_NL_noise_out(1, vis_index+1) = TMT_y_ALL(vis_index, kk,
1);
        TMT_y_NL_noise_out(2, vis_index+1) = TMT_y_ALL(vis_index, kk,
2);
        TMT_y_NL_noise_out(3, vis_index+1) = TMT_y_ALL(vis_index, kk,
3);

        TS_ID(vis_index+1) = repmat(kk, length(vis_index), 1);
    end

    for ii = 2:1401
        TMT_ydata(ii) = {[TMT_y_NL_noise_out(:, ii); TS_ID(ii)]};
    end

    TMT_State = [TMT_X; TMT_Xdot; TMT_Y; TMT_Ydot];

    % NEES and NIS
    Q_LKF = 1500*Qtrue;
    R_LKF = Rtrue;

    dx0 = dx_true;
    P0 = Px_true;

    [P, dx, x_stds, eytil, S] = LKF_StatOD(dx0, P0, TMT_ydata, dt,
Q_LKF, R_LKF, Gam, TS_state, nomCon');
```

```
    dxtrue = TMT_State - nomCon;
    ex = dxtrue - dx.pos;
    for ii = 1:1401
        Ex(jj, ii) = ex(:, ii)'*(P.pos(:, :, ii))^-1*ex(:, ii);
        Ey(jj, ii) = eytil(1:3, ii)'*(S(1:3, 1:3, ii))^-1*eytil(1:3,
 ii);
    end

end
```

TMT Plots and Errors

```
x = nomCon + dx.pos;

figure()
subplot(4, 1, 1)
plot(tvec, TMT_State(1, :), 'k')
hold on
plot(tvec, nomCon(1, :), 'b')
plot(tvec, x(1, :), 'r')
hold off
xlabel('Time [s]')
ylabel('X [km]')
set(gca, 'FontSize', 14)
legend('TMT', 'Nominal', 'LKF Estimate')

subplot(4, 1, 2)
plot(tvec, TMT_State(2, :), 'k')
hold on
plot(tvec, nomCon(2, :), 'b')
plot(tvec, x(2, :), 'r')
hold off
xlabel('Time [s]')
ylabel('Xdot [km/s]')
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
plot(tvec, TMT_State(3, :), 'k')
hold on
plot(tvec, nomCon(3, :), 'b')
plot(tvec, x(3, :), 'r')
hold off
xlabel('Time [s]')
ylabel('Y [km]')
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
plot(tvec, TMT_State(4, :), 'k')
hold on
plot(tvec, nomCon(4, :), 'b')
plot(tvec, x(4, :), 'r')
hold off
```

```
xlabel('Time [s]')
ylabel('Ydot [km/s]')
set(gca, 'FontSize', 14)

sgtitle('TMT Simulated States')

figure() % TMT Measurement Plots
subplot(4, 1, 1)
hold on
for ii = 1:12
    ID_index = find(TS_ID == ii);
    scatter(tvec(ID_index), TMT_y_NL_noise_out(1, ID_index), [],
 ColorSet(ii, :))
end
hold off
xlabel('Time [s]')
ylabel('TMT \rho^i [km]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 2)
hold on
for ii = 1:12
    ID_index = find(TS_ID == ii);
    scatter(tvec(ID_index), TMT_y_NL_noise_out(2, ID_index), [],
 ColorSet(ii, :))
end
hold off
xlabel('Time [s]')
ylabel('TMT \rhodot^i [km/s]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
hold on
for ii = 1:12
    ID_index = find(TS_ID == ii);
    scatter(tvec(ID_index), TMT_y_NL_noise_out(3, ID_index), [],
 ColorSet(ii, :))
end
hold off
xlabel('Time [s]')
ylabel('TMT \phi^i [rad]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
hold on
for ii = 1:12
    ID_index = find(TS_ID == ii);
    scatter(tvec(ID_index), TS_ID(ID_index), [], ColorSet(ii, :))
end
hold off
xlabel('Time [s]')
```

```
ylabel('Visible Station ID')
grid on
set(gca, 'FontSize', 14)

sgtitle('TMT Simulated Measurements vs Time')

figure() % innovations
subplot(3, 1, 1)
scatter(tvec, eytil(1, :))
xlabel('Time [s]')
ylabel('\rho Innovation')
set(gca, 'FontSize', 14)

subplot(3, 1, 2)
scatter(tvec, eytil(2, :))
xlabel('Time [s]')
ylabel('\rhodot Innovation')
set(gca, 'FontSize', 14)

subplot(3, 1, 3)
scatter(tvec, eytil(3, :))
xlabel('Time [s]')
ylabel('\phi Innovation')
set(gca, 'FontSize', 14)

sgtitle('Innovations vs Time')


figure()
subplot(4, 1, 1)
plot(tvec, ex(1, :), 'k')
hold on
plot(tvec, 2*x_stds(1, :), 'r')
plot(tvec, -2*x_stds(1, :), 'r')
hold off
xlabel('Time [s]')
ylabel('X Error [km]')
set(gca, 'FontSize', 14)

subplot(4, 1, 2)
plot(tvec, ex(2, :), 'k')
hold on
plot(tvec, 2*x_stds(2, :), 'r')
plot(tvec, -2*x_stds(2, :), 'r')
hold off
xlabel('Time [s]')
ylabel('Xdot Error [km/s]')
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
plot(tvec, ex(3, :), 'k')
hold on
plot(tvec, 2*x_stds(3, :), 'r')
plot(tvec, -2*x_stds(3, :), 'r')
```

```
hold off
xlabel('Time [s]')
ylabel('Y Error [km]')
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
plot(tvec, ex(4, :), 'k')
hold on
plot(tvec, 2*x_stds(4, :), 'r')
plot(tvec, -2*x_stds(4, :), 'r')
hold off
xlabel('Time [s]')
ylabel('Ydot Error [km/s]')
set(gca, 'FontSize', 14)

sgtitle('States Estimation Error vs Time - LKF')

% Consitancy Plots
Ex_mean = mean(Ex);
Ey_mean = mean(Ey);

alpha = 0.05;
r1 = chi2inv(alpha/2, N*n)/N;
r2 = chi2inv(1-alpha/2, N*n)/N;

figure() % NEES
scatter(tvec, Ex_mean)
hold on
plot(tvec, repmat(r1, 1401, 1), 'r--')
plot(tvec, repmat(r2, 1401, 1), 'r--')
hold off
ylim([2 6])
xlabel('Time [s]')
ylabel('Mean \epsilon_x')
title('LKF NEES Plot')
legend('NEES @ t_k', 'r_1 Bound', 'r_2 Bound')
grid on
set(gca, 'FontSize', 14)

alpha = 0.05;
r1 = chi2inv(alpha/2, N*p)/N;
r2 = chi2inv(1-alpha/2, N*p)/N;

figure() % NIS
scatter(tvec, Ey_mean)
hold on
plot(tvec, repmat(r1, 1401, 1), 'r--')
plot(tvec, repmat(r2, 1401, 1), 'r--')
hold off
ylim([1 5])
xlabel('Time [s]')
ylabel('Mean \epsilon_y')
title('LKF NIS Plot')
legend('NIS @ t_k', 'r_1 Bound', 'r_2 Bound')
```

```
grid on
set(gca, 'FontSize', 14)
```

Part II - LKF Implementation

```
dx0 = [0, 0.075, 0, -0.021];
P0 = Px_true;
[P, dx, x_stds, eytil, S] = LKF_StatOD(dx0, P0, ydata, dt, Q_LKF,
 R_LKF, Gam, TS_state, nomCon');

x = nomCon + dx.pos;

figure()
subplot(4, 1, 1)
plot(tvec, x(1, :), 'k')
hold on
plot(tvec, x(1, :) + 2*x_stds(1, :), 'r--')
plot(tvec, x(1, :) - 2*x_stds(1, :), 'r--')
hold off
xlabel('Time [s]')
ylabel('Estimated X [km]')
legend('Estimated State', '2\sigma')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 2)
plot(tvec, x(2, :), 'k')
hold on
plot(tvec, x(2, :) + 2*x_stds(2, :), 'r--')
plot(tvec, x(2, :) - 2*x_stds(2, :), 'r--')
hold off
xlabel('Time [s]')
ylabel('Estimated Xdot [km/s]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
plot(tvec, x(3, :), 'k')
hold on
plot(tvec, x(3, :) + 2*x_stds(1, :), 'r--')
plot(tvec, x(3, :) - 2*x_stds(1, :), 'r--')
hold off
xlabel('Time [s]')
ylabel('Estimated Y [km]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
plot(tvec, x(4, :), 'k')
hold on
plot(tvec, x(4, :) + 2*x_stds(2, :), 'r--')
plot(tvec, x(4, :) - 2*x_stds(2, :), 'r--')
hold off
```

```
xlabel('Time [s]')
ylabel('Estimated Ydot [km/s]')
grid on
set(gca, 'FontSize', 14)

sgtitle('Implemented LKF Estimated States vs Time')
```

Part II - EKF Tunning

```
N = 50;

xtrue = initCon;
Ptrue = diag([0.05, 0.00025, 0.05, 0.00025]);
qp = 1e-6;

for jj = 1:N
    % TMT
    perts = mvnrnd(xtrue, Ptrue);

    MC_Initial_State = perts;

    TMT_X(1) = MC_Initial_State(1);
    TMT_Xdot(1) = MC_Initial_State(2);
    TMT_Y(1) = MC_Initial_State(3);
    TMT_Ydot(1) = MC_Initial_State(4);

    v = mvnrnd([0, 0, 0], Rtrue)';
    TMT_y_NL_out(:, 1) = StatOD_NLMeasurement([TMT_X(1), TMT_Xdot(1),
 TMT_Y(1), TMT_Ydot(1)], [TS_X(1, 1), TS_Xdot(1, 1), TS_Y(1, 1),
 TS_Ydot(1, 1)]);
    TMT_y_NL_noise_out(:, 1) = TMT_y_NL_out(:, 1) + v;
    TMT_ydata(1) = {[TMT_y_NL_noise_out(:, 1); ydata_TS_ID(1)]};
    TS_ID = NaN*ones(1401, 1);
    TS_ID(1) = ydata_TS_ID(1);


    for ii = 1:1400
        % Noise
        w = mvnrnd([0, 0], Qtrue);
        v = mvnrnd([0, 0, 0], Rtrue)';

        % State
        ODE45_InitialState = [TMT_X(ii), TMT_Xdot(ii), TMT_Y(ii),
 TMT_Ydot(ii), w(1), w(2)];
        [~, TMT_test] = ode45(@(Time, State) StatODNL_noise_ODE(Time,
 State), [tvec(ii) tvec(ii+1)], ODE45_InitialState, options);

        TMT_X(ii+1) = TMT_test(end, 1);
        TMT_Xdot(ii+1) = TMT_test(end, 2);
        TMT_Y(ii+1) = TMT_test(end, 3);
        TMT_Ydot(ii+1) = TMT_test(end, 4);

        % Measurment
        for kk = 1:12 % compute measurements for each ground station
            yi = StatOD_NLMeasurement([TMT_X(ii+1), TMT_Xdot(ii+1),
TMT_Y(ii+1), TMT_Ydot(ii+1)], [TS_X(ii+1, kk), TS_Xdot(ii+1, kk),
TS_Y(ii+1, kk), TS_Ydot(ii+1, kk)]);
            TMT_y_ALL(ii, kk, 1) = yi(1) + v(1); % rho
            TMT_y_ALL(ii, kk, 2) = yi(2) + v(2); % rhodot
            TMT_y_ALL(ii, kk, 3) = yi(3) + v(3); % phi
        end
    end
    phiCompare = TMT_y_ALL(:, :, 3);

    for kk = 1:12 % compute the current visible ground station
        vis_index = find((phiCompare(:, kk) <= (pi/2 +
thetaCompare(2:end, kk)) & phiCompare(:, kk) >= (-pi/2 +
thetaCompare(2:end, kk))) | ...
            (phiCompare(:, kk) <= (pi/2 + thetaBound1Pos(2:end, kk)) &
phiCompare(:, kk) >= (-pi/2 + thetaBound1Neg(2:end, kk))) | ...
            (phiCompare(:, kk) <= (pi/2 + thetaBound2Pos(2:end, kk)) &
phiCompare(:, kk) >= (-pi/2 + thetaBound2Neg(2:end, kk))));

        TMT_y_NL_noise_out(1, vis_index+1) = TMT_y_ALL(vis_index, kk,
1);
        TMT_y_NL_noise_out(2, vis_index+1) = TMT_y_ALL(vis_index, kk,
2);
        TMT_y_NL_noise_out(3, vis_index+1) = TMT_y_ALL(vis_index, kk,
3);

        TS_ID(vis_index+1) = repmat(kk, length(vis_index), 1);


    end

    for ii = 2:1401
        TMT_ydata(ii) = {[TMT_y_NL_noise_out(:, ii); TS_ID(ii)]};
    end

    TMT_State = [TMT_X; TMT_Xdot; TMT_Y; TMT_Ydot];

    % NEES and NIS
    Q_EKF = 0.95*Qtrue;
    R_EKF = Rtrue;

    x0 = initCon;
    P0 = 10*eye(n);

    [P, x, x_stds, eytil, S] = EKF_StatOD(x0, P0, TMT_ydata, dt, tvec,
Q_EKF, R_EKF, Gam, TS_state);

    ex = TMT_State - x.pos;
    for ii = 1:1401
        Ex(jj, ii) = ex(:, ii)'*(P.pos(:, :, ii))^-1*ex(:, ii);
        Ey(jj, ii) = eytil(1:3, ii)'*(S(1:3, 1:3, ii))^-1*eytil(1:3,
ii);
    end
```

```
end

figure()
subplot(4, 1, 1)
plot(tvec, TMT_State(1, :), 'k')
hold on
plot(tvec, nomCon(1, :), 'g')
plot(tvec, x.pos(1, :), 'r')
hold off
xlabel('Time [s]')
ylabel('X [km]')
set(gca, 'FontSize', 14)
legend('TMT', 'Nominal', 'LKF Estimate')

subplot(4, 1, 2)
plot(tvec, TMT_State(2, :), 'k')
hold on
plot(tvec, nomCon(2, :), 'g')
plot(tvec, x.pos(2, :), 'r')
hold off
xlabel('Time [s]')
ylabel('Xdot [km/s]')
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
plot(tvec, TMT_State(3, :), 'k')
hold on
plot(tvec, nomCon(3, :), 'g')
plot(tvec, x.pos(3, :), 'r')
hold off
xlabel('Time [s]')
ylabel('Y [km]')
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
plot(tvec, TMT_State(4, :), 'k')
hold on
plot(tvec, nomCon(4, :), 'g')
plot(tvec, x.pos(4, :), 'r')
hold off
xlabel('Time [s]')
ylabel('Ydot [km/s]')
set(gca, 'FontSize', 14)

sgtitle('TMT Simulated States')

figure() % state estimation errors
subplot(4, 1, 1)
plot(tvec, ex(1, :), 'k')
hold on
plot(tvec, 2*x_stds(1, :), 'r--')
plot(tvec, -2*x_stds(1, :), 'r--')
hold off
```

```
ylim([-0.5 0.5])
xlabel('Time [s]')
ylabel('X Error [km]')
legend('State Estimation Error', '2\sigma Bounds')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 2)
plot(tvec, ex(2, :), 'k')
hold on
plot(tvec, 2*x_stds(2, :), 'r--')
plot(tvec, -2*x_stds(2, :), 'r--')
hold off
ylim([-0.005 0.005])
xlabel('Time [s]')
ylabel('Xdot Error [km/s]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
plot(tvec, ex(3, :), 'k')
hold on
plot(tvec, 2*x_stds(3, :), 'r--')
plot(tvec, -2*x_stds(3, :), 'r--')
hold off
ylim([-0.6 0.6])
xlabel('Time [s]')
ylabel('Y Error [km]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
plot(tvec, ex(4, :), 'k')
hold on
plot(tvec, 2*x_stds(4, :), 'r--')
plot(tvec, -2*x_stds(4, :), 'r--')
hold off
ylim([-0.005 0.005])
xlabel('Time [s]')
ylabel('Ydot Error [km/s]')
grid on
set(gca, 'FontSize', 14)

sgtitle('States Estimation Error vs Time - EKF')

figure() % innovations
subplot(3, 1, 1)
scatter(tvec, eytil(1, :))
xlabel('Time [s]')
ylabel('\rho^i Error [km]');
grid on
set(gca, 'FontSize', 14)
ylim([-0.5 0.5])
```

```
subplot(3, 1, 2)
scatter(tvec, eytil(2, :))
xlabel('Time [s]')
ylabel('\rhodot^i Error [km/s]');
grid on
set(gca, 'FontSize', 14)

subplot(3, 1, 3)
scatter(tvec, eytil(3, :))
xlabel('Time [s]')
ylabel('\phi^i Error [rad]');
grid on
set(gca, 'FontSize', 14)

sgtitle('Inovations vs Time - EKF')

% NEES and NIS Plots

Ex_mean = mean(Ex);
Ey_mean = mean(Ey);

alpha = 0.05;
r1 = chi2inv(alpha/2, N*n)/N;
r2 = chi2inv(1-alpha/2, N*n)/N;

figure() % NEES
scatter(tvec, Ex_mean)
hold on
plot(tvec, repmat(r1, 1401, 1), 'r--')
plot(tvec, repmat(r2, 1401, 1), 'r--')
hold off
ylim([2 6])
xlabel('Time [s]')
ylabel('Mean \epsilon_x')
title('EKF NEES Plot')
legend('NEES @ t_k', 'r_1 Bound', 'r_2 Bound')
grid on
set(gca, 'FontSize', 14)

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

figure() % NIS
scatter(tvec, Ey_mean);
hold on
plot(tvec, repmat(r1, 1401, 1), 'r--')
plot(tvec, repmat(r2, 1401, 1), 'r--')
hold off
ylim([1 5])
xlabel('Time [s]')
ylabel('Mean \epsilon_y')
title('EKF NIS Plot')
legend('NIS @ t_k', 'r_1 Bound', 'r_2 Bound')
grid on
```

```
set(gca, 'FontSize', 14)

figure() % TMT state plots
subplot(4, 1, 1)
plot(tvec, TMT_X, 'k')
hold on
plot(tvec, nomCon(1, :), 'r')
hold off
xlabel('Time [s]')
ylabel('TMT X [km]')
legend('TMT State', 'Nominal Trajectory')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 2)
plot(tvec, TMT_Xdot, 'k')
hold on
plot(tvec, nomCon(2, :), 'r')
hold off
xlabel('Time [s]')
ylabel('TMT Xdot [km/s]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
plot(tvec, TMT_Y, 'k')
hold on
plot(tvec, nomCon(3, :), 'r')
hold off
xlabel('Time [s]')
ylabel('TMT Y [km]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
plot(tvec, TMT_Ydot, 'k')
hold on
plot(tvec, nomCon(4, :), 'r')
hold off
xlabel('Time [s]')
ylabel('TMT Ydot [km/s]')
grid on
set(gca, 'FontSize', 14)

sgtitle('TMT Simulated States vs Time')

figure() % TMT Measurement Plots
subplot(4, 1, 1)
hold on
for ii = 1:12
    ID_index = find(TS_ID == ii);
    scatter(tvec(ID_index), TMT_y_NL_noise_out(1, ID_index), [],
 ColorSet(ii, :))
end
```

```
hold off
xlabel('Time [s]')
ylabel('TMT \rho^i [km]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 2)
hold on
for ii = 1:12
    ID_index = find(TS_ID == ii);
    scatter(tvec(ID_index), TMT_y_NL_noise_out(2, ID_index), [],
 ColorSet(ii, :))
end
hold off
xlabel('Time [s]')
ylabel('TMT \rhodot^i [km/s]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
hold on
for ii = 1:12
    ID_index = find(TS_ID == ii);
    scatter(tvec(ID_index), TMT_y_NL_noise_out(3, ID_index), [],
 ColorSet(ii, :))
end
hold off
xlabel('Time [s]')
ylabel('TMT \phi^i [rad]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
hold on
for ii = 1:12
    ID_index = find(TS_ID == ii);
    scatter(tvec(ID_index), TS_ID(ID_index), [], ColorSet(ii, :))
end
hold off
xlabel('Time [s]')
ylabel('Visible Station ID')
grid on
set(gca, 'FontSize', 14)

sgtitle('TMT Simulated Measurements vs Time')
```

EKF Implementation

```
x0 = Initial_States;
P0 = 10*eye(n);

[~, x, x_stds, ~, ~] = EKF_StatOD(x0, P0, ydata, dt, tvec, Q_EKF,
 R_EKF, Gam, TS_state);
```

```
figure('Position', [200, 200, 1800, 1000])
subplot(4, 1, 1)
hold on
plot(tvec, TMT_X, 'k')
plot(tvec, nomCon(1, :), 'g')
plot(tvec, x.pos(1, :), 'r')
hold off
xlabel('Time [s]')
ylabel('X [km]')
grid on
set(gca, 'FontSize', 14)
legend('TMT', 'Nominal','EKF Estimate','location','bestoutside')

subplot(4, 1, 2)
hold on
plot(tvec, TMT_Xdot, 'k')
plot(tvec, nomCon(2, :), 'g')
plot(tvec, x.pos(2, :), 'r')
hold off
xlabel('Time [s]')
ylabel('Xdot [km/s]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 3)
hold on
plot(tvec, TMT_Y, 'k')
plot(tvec, nomCon(3, :), 'g')
plot(tvec, x.pos(3, :), 'r')
hold off
xlabel('Time [s]')
ylabel('Y [km]')
grid on
set(gca, 'FontSize', 14)

subplot(4, 1, 4)
hold on
plot(tvec, TMT_Ydot, 'k')
plot(tvec, nomCon(4, :), 'g')
plot(tvec, x.pos(4, :), 'r')
hold off
xlabel('Time [s]')
ylabel('Ydot [km/s]')
grid on
set(gca, 'FontSize', 14)

sgtitle('TMT Simulated States')
```

A Solver

```
function [Atil] = Atil_Solver(State)
```

```
mu = 398600; % Earth's standard gravitational paremters [km^3/s^2]

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

Atil = [0, 1, 0, 0;
    (3*mu*x1^2)/(x1^2 + x3^2)^(5/2) - mu/(x1^2 + x3^2)^(3/2), 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, (3*mu*x3^2)/(x1^2 +
 x3^2)^(5/2) - mu/(x1^2 + x3^2)^(3/2), 0];

end
```

C Solver

```
function [Ctil] = Ctil_Solver(State, TS_State)

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

z1 = TS_State(1);
z2 = TS_State(2);
z3 = TS_State(3);
z4 = TS_State(4);

Ctil = [(2*x1 - 2*z1)/(2*((x1 - z1)^2 + (x3 - z3)^2)^(1/2)), 0, (2*x3
 - 2*z3)/(2*((x1 - z1)^2 + (x3 - z3)^2)^(1/2)), 0;
    (x2 - z2)/((x1 - z1)^2 + (x3 - z3)^2)^(1/2) - ((2*x1 - 2*z1)*((x1
 - z1)*(x2 - z2) + (x3 - z3)*(x4 - z4)))/(2*((x1 - z1)^2 + (x3 -
 z3)^2)^(3/2)), (x1 - z1)/((x1 - z1)^2 + (x3 - z3)^2)^(1/2), (x4 -
 z4)/((x1 - z1)^2 + (x3 - z3)^2)^(1/2) - ((2*x3 - 2*z3)*((x1 - z1)*(x2
 - z2) + (x3 - z3)*(x4 - z4)))/(2*((x1 - z1)^2 + (x3 - z3)^2)^(3/2)),
 (x3 - z3)/((x1 - z1)^2 + (x3 - z3)^2)^(1/2);
    -(x3 - z3)/((x1 - z1)^2*((x3 - z3)^2/(x1 - z1)^2 + 1)), 0, 1/((x1
 - z1)*((x3 - z3)^2/(x1 - z1)^2 + 1)), 0];

end
```

EKF StatOD

```
function [P, x, x_stds, eytil, S] = EKF_StatOD(x0, P0, ydata, dt,
 tvec, Q, R, Gamma, TS_state)


% DEAL WITH MULT Measurement values!!!!!!!

% Matrix sizes and Steps
n = length(x0); % number of states
```

```
p = length(ydata{1}) - 1; % number of measurments, subtract one since
 it has GND station ID
steps = length(ydata); % number of steps for problem; step 1 is the
 zero time vec

%mu = 398600; % Earth's standard gravitational paremters [km^3/s^2]
Omega = dt*Gamma; % Since CT Gamma matrix is LTI we can compute Omega
 outside EKF loop

% ODE Tolerances
Rel_Tol = 1e-13;
Abs_Tol = Rel_Tol;
options = odeset('Stats', 'off', 'RelTol', Rel_Tol, 'AbsTol',
 Abs_Tol);

% initilaize variables for speed
x.neg(1:n, 1) = NaN*ones(n, 1);
x.pos(:, 1) = x0;

P.neg(1:n, 1:n, 1) = NaN*ones(n);
P.pos(:, :, 1) = P0;

x_stds(1:n, 1) = sqrt(diag(P0));

eytil = NaN*ones(2*p, 1);

S = NaN*ones(2*p, 2*p, steps);

y = NaN*ones(2*p, steps);
TS_ID = NaN*ones(2, steps);
c = NaN*ones(1, steps);

% Parse ydata into own data vector and GND station IDS
for ii = 1:steps
    if ~isempty(ydata{ii})
        [~, c(ii)] = size(ydata{ii});
        if c(ii) == 1
            y(1:3, ii) = ydata{ii}(1:3);
            TS_ID(1, ii) = ydata{ii}(4);
        elseif c(ii) == 2
            y(1:3, ii) = ydata{ii}(1:3, 1);
            TS_ID(1, ii) = ydata{ii}(4, 1);
            y(4:6, ii) = ydata{ii}(1:3, 2);
            TS_ID(2, ii) = ydata{ii}(4, 2);
        end
    end
end

No_Meas_index = isnan(TS_ID(1, :));

% EKF Loop
for ii = 2:steps
   % Prediction Step
   tspan = [tvec(ii-1) tvec(ii)];
```

```
  [~, NL_state] = ode45(@(Time, State) StatODNL_ODE(Time, State),
tspan, x.pos(:, ii-1)', options);
  x.neg(:, ii) = NL_state(end, :)';

  % NL and Jacobian Computaion (Part of Prediction Step)
  Atil = Atil_Solver(x.pos(:, ii-1));

  Ftil = eye(n) + dt*Atil;
  P.neg(:, :, ii) = Ftil*P.pos(:, :, ii-1)*Ftil' + Omega*Q*Omega';

  % Correction Step
  if No_Meas_index(ii) == 0 % There are Measurments
      if c(ii) == 1
          z1 = TS_state(ii, TS_ID(1, ii), 1);
          z2 = TS_state(ii, TS_ID(1, ii), 2);
          z3 = TS_state(ii, TS_ID(1, ii), 3);
          z4 = TS_state(ii, TS_ID(1, ii), 4);
          TS_stateK = [z1; z2; z3; z4];

          y_neg = StatOD_NLMeasurement(x.neg(:, ii), TS_stateK);

          Htil = Ctil_Solver(x.neg(:, ii), TS_stateK);

          eytil(1:3, ii) = y(1:3, ii) - y_neg;

          Ktil = P.neg(:, :, ii)*Htil'*(Htil*P.neg(:, :, ii)*Htil' +
R)^-1;

          x.pos(:, ii) = x.neg(:, ii) + Ktil*eytil(1:3, ii);
          P.pos(:, :, ii) = (eye(n) - Ktil*Htil)*P.neg(:, :, ii);

          S(1:p, 1:p, ii) = Htil*P.neg(:, :, ii)*Htil' + R;

      elseif c(ii) == 2
          % first measurment
          z1 = TS_state(ii, TS_ID(1, ii), 1);
          z2 = TS_state(ii, TS_ID(1, ii), 2);
          z3 = TS_state(ii, TS_ID(1, ii), 3);
          z4 = TS_state(ii, TS_ID(1, ii), 4);
          TS_stateK = [z1; z2; z3; z4];

          y_neg_1 = StatOD_NLMeasurement(x.neg(:, ii), TS_stateK);

          Htil_1 = Ctil_Solver(x.neg(:, ii), TS_stateK);

          eytil(1:3, ii) = y(1:3, ii) - y_neg_1;
          Ktil_1 = P.neg(:, :, ii)*Htil_1'*(Htil_1*P.neg(:, :,
ii)*Htil_1' + R)^-1;

          % second measurment
          z1 = TS_state(ii, TS_ID(2, ii), 1);
          z2 = TS_state(ii, TS_ID(2, ii), 2);
          z3 = TS_state(ii, TS_ID(2, ii), 3);
          z4 = TS_state(ii, TS_ID(2, ii), 4);
```

```
            TS_stateK = [z1; z2; z3; z4];

            y_neg_2 = StatOD_NLMeasurement(x.neg(:, ii), TS_stateK);

            Htil_2 = Ctil_Solver(x.neg(:, ii), TS_stateK);

            eytil(4:6, ii) = y(4:6, ii) - y_neg_2;

            Ktil_2 = P.neg(:, :, ii)*Htil_2'*(Htil_2*P.neg(:, :,
 ii)*Htil_2' + R)^-1;

            % Combined
            Pblock = blkdiag(P.neg(:, :, ii), P.neg(:, :, ii));
            Hblock = blkdiag(Htil_1, Htil_2);
            Rblock = blkdiag(R, R);

            x.pos(:, ii) = x.neg(:, ii) + Ktil_1*eytil(1:3, ii) +
 Ktil_2*eytil(4:6, ii);
            P.pos(:, :, ii) =  P.neg(:, :, ii) -
 Ktil_1*Htil_1*P.neg(:, :, ii) - Ktil_2*Htil_2*P.neg(:, :, ii);

            S(:, :, ii) = Hblock*Pblock*Hblock' + Rblock;
        end
    else % No Measurements
        x.pos(:, ii) = x.neg(:, ii);
        P.pos(:, :, ii) = P.neg(:, :, ii);
        eytil(:, ii) = NaN*ones(2*p, 1);
    end

    % Standard Deviations
    x_stds(:, ii) = sqrt(diag(P.pos(:, :, ii)));

end

end
```

LKF StatOD

```
function [P, dx, x_stds, eytil, S] = LKF_StatOD(dx0, P0, ydata, dt, Q,
 R, Gamma, TS_state, Nom_State)

% Matrix sizes and Steps
n = length(dx0); % number of states
p = length(ydata{1}) - 1; % number of measurments, subtract one since
 it has GND station ID
steps = length(ydata); % number of steps for problem; step 1 is the
 zero time vec

Omega = dt*Gamma; % Since CT Gamma matrix is LTI we can compute Omega
 outside EKF loop

% initilaize variables for speed
dx.neg(1:n, 1) = NaN*ones(n, 1);
dx.pos(:, 1) = dx0;

P.neg(1:n, 1:n, 1) = NaN*ones(n);
P.pos(:, :, 1) = P0;

x_stds(1:n, 1) = sqrt(diag(P0));

eytil = NaN*ones(2*p, 1);

S = NaN*ones(2*p, 2*p, steps);

y = NaN*ones(2*p, steps);
TS_ID = NaN*ones(2, steps);
c = NaN*ones(1, steps);

% Parse ydata into own data vector and GND station IDS
for ii = 1:steps
    if ~isempty(ydata{ii})
        [~, c(ii)] = size(ydata{ii});
        if c(ii) == 1
            y(1:3, ii) = ydata{ii}(1:3);
            TS_ID(1, ii) = ydata{ii}(4);
        elseif c(ii) == 2
            y(1:3, ii) = ydata{ii}(1:3, 1);
            TS_ID(1, ii) = ydata{ii}(4, 1);
            y(4:6, ii) = ydata{ii}(1:3, 2);
            TS_ID(2, ii) = ydata{ii}(4, 2);
        end
    end
end

No_Meas_index = isnan(TS_ID(1, :));

% LKF Loop
for ii = 2:steps
    % Prediction Step
    Atil = Atil_Solver(Nom_State(ii-1, :));
    Ftil = eye(n) + dt*Atil;

    dx.neg(:, ii) = Ftil*dx.pos(:, ii-1);

    P.neg(:, :, ii) = Ftil*P.pos(:, :, ii-1)*Ftil' + Omega*Q*Omega';

    % Correction Step %%% Deal with multiple measurments
    if No_Meas_index(ii) == 0 % There are Measurments
        if c(ii) == 1
        z1 = TS_state(ii, TS_ID(1, ii), 1);
        z2 = TS_state(ii, TS_ID(1, ii), 2);
        z3 = TS_state(ii, TS_ID(1, ii), 3);
        z4 = TS_state(ii, TS_ID(1, ii), 4);
        TS_stateK = [z1; z2; z3; z4];

        Htil = Ctil_Solver(Nom_State(ii, :), TS_stateK);
        Ktil = P.neg(:, :, ii)*Htil'*(Htil*P.neg(:, :, ii)*Htil' +
R)^-1;

        y_nom = StatOD_NLMeasurement(Nom_State(ii, :), TS_stateK);

        dy = y(1:3, ii) - y_nom;

        dx.pos(:, ii) = dx.neg(:, ii) + Ktil*(dy - Htil*dx.neg(:,
ii));

        P.pos(:, :, ii) = (eye(n) - Ktil*Htil)*P.neg(:, :, ii);

        S(1:3, 1:3, ii) = Htil*P.neg(:, :, ii)*Htil' + R;
        eytil(1:3, ii) = dy - Htil*dx.neg(:, ii);

        elseif c(ii) == 2
            % First measurment
            z1 = TS_state(ii, TS_ID(1, ii), 1);
            z2 = TS_state(ii, TS_ID(1, ii), 2);
            z3 = TS_state(ii, TS_ID(1, ii), 3);
            z4 = TS_state(ii, TS_ID(1, ii), 4);
            TS_stateK = [z1; z2; z3; z4];

            Htil_1 = Ctil_Solver(Nom_State(ii, :), TS_stateK);

            Ktil_1 = P.neg(:, :, ii)*Htil_1'*(Htil_1*P.neg(:, :,
ii)*Htil_1' + R)^-1;

            y_nom_1 = StatOD_NLMeasurement(Nom_State(ii, :),
TS_stateK);

            dy_1 = y(1:3, ii) - y_nom_1;

            eytil(1:3, ii) = dy_1 - Htil_1*dx.neg(:, ii);

            % Second measurment
            z1 = TS_state(ii, TS_ID(2, ii), 1);
            z2 = TS_state(ii, TS_ID(2, ii), 2);
            z3 = TS_state(ii, TS_ID(2, ii), 3);
            z4 = TS_state(ii, TS_ID(2, ii), 4);
            TS_stateK = [z1; z2; z3; z4];

            Htil_2 = Ctil_Solver(Nom_State(ii, :), TS_stateK);

            Ktil_2 = P.neg(:, :, ii)*Htil_2'*(Htil_2*P.neg(:, :,
ii)*Htil_2' + R)^-1;

            y_nom_2 = StatOD_NLMeasurement(Nom_State(ii, :),
TS_stateK);

            dy_2 = y(4:6, ii) - y_nom_2;

            eytil(4:6, ii) = dy_2 - Htil_2*dx.neg(:, ii);
            % Combined
            Pblock = blkdiag(P.neg(:, :, ii), P.neg(:, :, ii));
            Hblock = blkdiag(Htil_1, Htil_2);
            Rblock = blkdiag(R, R);

            dx.pos(:, ii) = dx.neg(:, ii) + Ktil_1*(dy_1 -
 Htil_1*dx.neg(:, ii)) + Ktil_2*(dy_2 - Htil_2*dx.neg(:, ii));

            P.pos(:, :, ii) = P.neg(:, :, ii) -
 Ktil_1*Htil_1*P.neg(:, :, ii) - Ktil_2*Htil_2*P.neg(:, :, ii);

            S(:, :, ii) = Hblock*Pblock*Hblock' + Rblock;

        end
    else
       dx.pos(:, ii) = dx.neg(:, ii);
       P.pos(:, :, ii) = P.neg(:, :, ii);
       eytil(:, ii) = NaN*ones(2*p, 1);
    end

    % Standard Deviations
    x_stds(:, ii) = sqrt(diag(P.pos(:, :, ii)));

end

end
```

StatOD NonLinear Measurements

```
function [y] = StatOD_NLMeasurement(Sat_state, TS_state)

X = Sat_state(1);
Xdot = Sat_state(2);
Y = Sat_state(3);
Ydot = Sat_state(4);

Xi = TS_state(1);
Xidot = TS_state(2);
Yi = TS_state(3);
Yidot = TS_state(4);

rho = sqrt((X - Xi)^2 + (Y - Yi)^2);
rhodot = ((X - Xi)*(Xdot - Xidot) + (Y - Yi)*(Ydot - Yidot))/sqrt((X -
 Xi)^2 + (Y - Yi)^2);
phi = atan2(Y - Yi, X - Xi);

y = [rho; rhodot; phi];

end
```

Stat OD NonLinear Noise ODE

```
function [State_Derivatives] = StatODNL_noise_ODE(Time, State)
```

```
u = 398600; % Earth's standard gravitational paremters [km^3/s^2]

X = State(1);
Xdot = State(2);
Y = State(3);
Ydot = State(4);

r = sqrt(X^2 + Y^2);

w1 = State(5);
w2 = State(6);

Xddot = -u*X/r^3 + w1;
Yddot = -u*Y/r^3 + w2;

State_Derivatives = [Xdot; Xddot; Ydot; Yddot; 0; 0];

end
```

StatOD NonLinear ODE

```
function [State_Derivatives] = StatODNL_ODE(Time, State)

u = 398600; % Earth's standard gravitational paremters [km^3/s^2]

X = State(1);
Xdot = State(2);
Y = State(3);
Ydot = State(4);

r = sqrt(X^2 + Y^2);

Xddot = -u*X/r^3;
Yddot = -u*Y/r^3;

State_Derivatives = [Xdot; Xddot; Ydot; Yddot];

end
```

Vary Color

```
function ColorSet=varycolor(NumberOfPlots)
% VARYCOLOR Produces colors with maximum variation on plots with
 multiple
% lines.
%
%     VARYCOLOR(X) returns a matrix of dimension X by 3.  The matrix
 may be
%     used in conjunction with the plot command option 'color' to vary
 the
%     color of lines.
%
```

```
%     Yellow and White colors were not used because of their poor
%     translation to presentations.
%
%     Example Usage:
%         NumberOfPlots=50;
%
%         ColorSet=varycolor(NumberOfPlots);
%
%         figure
%         hold on;
%
%         for m=1:NumberOfPlots
%             plot(ones(20,1)*m,'Color',ColorSet(m,:))
%         end
%Created by Daniel Helmick 8/12/2008
error(nargchk(1,1,nargin))%correct number of input arguements??
error(nargoutchk(0, 1, nargout))%correct number of output arguements??
%Take care of the anomolies
if NumberOfPlots<1
    ColorSet=[];
elseif NumberOfPlots==1
    ColorSet=[0 1 0];
elseif NumberOfPlots==2
    ColorSet=[0 1 0; 0 1 1];
elseif NumberOfPlots==3
    ColorSet=[0 1 0; 0 1 1; 0 0 1];
elseif NumberOfPlots==4
    ColorSet=[0 1 0; 0 1 1; 0 0 1; 1 0 1];
elseif NumberOfPlots==5
    ColorSet=[0 1 0; 0 1 1; 0 0 1; 1 0 1; 1 0 0];
elseif NumberOfPlots==6
    ColorSet=[0 1 0; 0 1 1; 0 0 1; 1 0 1; 1 0 0; 0 0 0];
else %default and where this function has an actual advantage
    %we have 5 segments to distribute the plots
    EachSec=floor(NumberOfPlots/5);

    %how many extra lines are there?
    ExtraPlots=mod(NumberOfPlots,5);

    %initialize our vector
    ColorSet=zeros(NumberOfPlots,3);

    %This is to deal with the extra plots that don't fit nicely into
 the
    %segments
    Adjust=zeros(1,5);
    for m=1:ExtraPlots
        Adjust(m)=1;
    end

    SecOne   =EachSec+Adjust(1);
    SecTwo   =EachSec+Adjust(2);
    SecThree =EachSec+Adjust(3);
    SecFour  =EachSec+Adjust(4);
    SecFive  =EachSec;
    for m=1:SecOne
        ColorSet(m,:)=[0 1 (m-1)/(SecOne-1)];
    end
    for m=1:SecTwo
        ColorSet(m+SecOne,:)=[0 (SecTwo-m)/(SecTwo) 1];
    end

    for m=1:SecThree
        ColorSet(m+SecOne+SecTwo,:)=[(m)/(SecThree) 0 1];
    end

    for m=1:SecFour
        ColorSet(m+SecOne+SecTwo+SecThree,:)=[1 0 (SecFour-m)/
(SecFour)];
    end
    for m=1:SecFive
        ColorSet(m+SecOne+SecTwo+SecThree+SecFour,:)=[(SecFive-m)/
(SecFive) 0 0];
    end

end
```

Published with MATLAB® R2020b