Ann and H.J. Smead
Aerospace Engineering Sciences
UNIVERSITY OF COLORADO **BOULDER**

ASEN 5044 Final Project: Statistical Orbit Determination

Brady Sivey
Jared Steffen
Nathan Whittenburg

December 11, 2025

Author Contributions:

Brady Sivey

- Part 1 Question 1
- Created the EKF algorithm code
- Tuned the LKF

Jared Steffen

- Simulated the nonlinear and linear dynamics and measurements
- Implemented the code for the LKF
- Implemented the code for and tuned the UKF

Nathan Whittenburg

- Part 1 Question 2
- Implemented the code for Monte Carlo simulations with NEES/NIS tests
- Tuned the EKF

1 Introduction

Space missions that monitor Earth's oceans rely on extremely precise knowledge of a satellite's position in orbit. Modern altimetry satellites can measure global sea-level height to within about an inch, which means the spacecraft's orbit must be known with comparable accuracy. Achieving this level of precision falls under the field of statistical orbit determination. This project explores a greatly simplified example to illustrate the fundamentals of this process.

2 System Description

Assuming a gravity point mass model which obeys the inverse square law, the full nonlinear equations of motion are:

$$\ddot{X} = \frac{-\mu X}{r^3} + u_1 + \tilde{w}_1$$
$$\ddot{Y} = \frac{-\mu Y}{r^3} + u_2 + \tilde{w}_2$$

Where $r = \sqrt{X^2 + Y^2}$, $\mu = 398,600 \frac{km^3}{s^2}$, $u_{1,2}$ are control accelerations, and $\tilde{w}_{1,2}$ are disturbances. In a circular orbit at 300 km, $r_0 = 6678$ km, and the states, inputs, and disturbances of the spacecraft can be described by:

$$\vec{x}(t) = [X \;\; \dot{X} \;\; Y \;\; \dot{Y}]^T$$
$$\vec{x}_{nom}(0) = \left[r_0 \quad 0 \;\; 0 \quad r_0\sqrt{\frac{\mu}{r_0^3}}\right]^T$$
$$\vec{u}(t) = [u_1 \;\; u_2]^T$$
$$\vec{u}_{nom}(t) = [0 \;\; 0]^T$$
$$\vec{\tilde{w}}(t) = [\tilde{w}_1 \;\; \tilde{w}_2]^T$$

Nonlinear measurements are given by range, range rate, and elevation angle from i tracking stations located at $(X_s^i(t), Y_s^i(t))$. These station coordinates are known for all time and can be described by:

$$X_s^i(t) = R_E \cos(\omega_E t + \theta^i(0)) \qquad Y_s^i(t) = R_E \sin(\omega_E t + \theta^i(0)),$$

$$\theta^i(0) = (i-1)\frac{\pi}{6} \qquad i = 1, 2, ..., 12$$

The nonlinear measurement equations are as follows:

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

Where $\vec{\tilde{v}}^i(t)$ is the measurement error vector at each station

$$\rho^i(t) = \sqrt{(X(t) - X_s^i(t))^2 + (Y(t) - Y_s^i(t))^2}$$

$$\dot{\rho}^i(t) = \frac{[X(t) - X_s^i(t)]\,[\dot{X}(t) - \dot{X}_s^i(t)] + [Y(t) - Y_s^i(t)]\,[\dot{Y}(t) - \dot{Y}_s^i(t)]}{\rho^i(t)}$$

$$\phi^i(t) = \tan^{-1}\left(\frac{Y(t) - Y_s^i(t)}{X(t) - X_s^i(t)}\right).$$

And measurements for a station are only recorded if the spacecraft is visible to the station, i.e.:

$$\phi^i(t) \in \left[-\frac{\pi}{2} + \theta^i(t),\ \frac{\pi}{2} + \theta^i(t)\right] \qquad \theta^i(t) = \tan^{-1}\left(\frac{Y_s^i(t)}{X_s^i(t)}\right)$$

3 Part I: Deterministic System Analysis

3.1 Question 1: Continuous Time Jacobians

The first step in deterministic system analysis for the statistical orbit determination is deriving the required CT Jacobians. Eq. (1) was derived utilizing the equations defined in the system description. From here, the CT Jacobians were derived as seen in eqs. (2)-(5). The size for all these matrices are listed below. It is important to note that the size for the H matrix will be dependent on which satellites are visible. Each satellite will have a $3 * 4$ H matrix corresponding to it that will than be stacked to create the full H matrix for the system. This H matrix will be a different size at different times.

$$\mathbf{x} = \begin{bmatrix} X \\ \dot{X} \\ Y \\ \dot{Y} \end{bmatrix}, \qquad \dot{\mathbf{x}} = \begin{bmatrix} \dot{X} \\ \ddot{X} \\ \dot{Y} \\ \ddot{Y} \end{bmatrix} = \begin{bmatrix} \dot{X} \\ -\mu\dfrac{X}{r^3} + u_1 + \tilde{w}_1 \\ \dot{Y} \\ -\mu\dfrac{Y}{r^3} + u_2 + \tilde{w}_2 \end{bmatrix}, \qquad r = \sqrt{X^2 + Y^2}. \tag{1}$$

$$\frac{\partial F}{\partial x} = \begin{bmatrix} \frac{\partial \dot{x}}{\partial x} & \frac{\partial \dot{x}}{\partial \dot{x}} & \frac{\partial \dot{x}}{\partial y} & \frac{\partial \dot{x}}{\partial \dot{y}} \\ \frac{\partial \ddot{x}}{\partial x} & \frac{\partial \ddot{x}}{\partial \dot{x}} & \frac{\partial \ddot{x}}{\partial y} & \frac{\partial \ddot{x}}{\partial \dot{y}} \\ \frac{\partial \dot{y}}{\partial x} & \frac{\partial \dot{y}}{\partial \dot{x}} & \frac{\partial \dot{y}}{\partial y} & \frac{\partial \dot{y}}{\partial \dot{y}} \\ \frac{\partial \ddot{y}}{\partial x} & \frac{\partial \ddot{y}}{\partial \dot{x}} & \frac{\partial \ddot{y}}{\partial y} & \frac{\partial \ddot{y}}{\partial \dot{y}} \end{bmatrix} = A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\mu\dfrac{(y^2 - 2x^2)}{(x^2 + y^2)^{5/2}} & 0 & \mu\dfrac{3xy}{(x^2 + y^2)^{5/2}} & 0 \\ 0 & 0 & 0 & 1 \\ \mu\dfrac{3xy}{(x^2 + y^2)^{5/2}} & 0 & -\mu\dfrac{(x^2 - 2y^2)}{(x^2 + y^2)^{5/2}} & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}. \tag{2}$$

$$\frac{\partial F}{\partial u} = \begin{bmatrix} \frac{\partial \dot{x}}{\partial u_1} & \frac{\partial \dot{x}}{\partial u_2} \\ \frac{\partial \ddot{x}}{\partial u_1} & \frac{\partial \ddot{x}}{\partial u_2} \\ \frac{\partial \dot{y}}{\partial u_1} & \frac{\partial \dot{y}}{\partial u_2} \\ \frac{\partial \ddot{y}}{\partial u_1} & \frac{\partial \ddot{y}}{\partial u_2} \end{bmatrix} = B = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 2}. \tag{3}$$

$$\frac{\partial h}{\partial x} = \begin{bmatrix} \frac{\partial \rho}{\partial x} & \frac{\partial \rho}{\partial \dot{x}} & \frac{\partial \rho}{\partial y} & \frac{\partial \rho}{\partial \dot{y}} \\ \frac{\partial \dot{\rho}}{\partial x} & \frac{\partial \dot{\rho}}{\partial \dot{x}} & \frac{\partial \dot{\rho}}{\partial y} & \frac{\partial \dot{\rho}}{\partial \dot{y}} \\ \frac{\partial \phi}{\partial x} & \frac{\partial \phi}{\partial \dot{x}} & \frac{\partial \phi}{\partial y} & \frac{\partial \phi}{\partial \dot{y}} \end{bmatrix} = H_i = \begin{bmatrix} \frac{\Delta X}{\rho} & 0 & \frac{\Delta Y}{\rho} & 0 \\ \frac{\rho\,\Delta\dot{X} - A\left(\frac{\Delta X}{\rho}\right)}{\rho^2} & \frac{\Delta X}{\rho} & \frac{\rho\,\Delta\dot{Y} - A\left(\frac{\Delta Y}{\rho}\right)}{\rho^2} & \frac{\Delta Y}{\rho} \\ -\frac{\Delta Y}{\rho^2} & 0 & \frac{\Delta X}{\rho^2} & 0 \end{bmatrix} \in \mathbb{R}^{3\times 4}. \tag{4}$$

$$\begin{aligned} \Delta X &= X(t) - X_s(t), & \Delta \dot{X} &= \dot{X}(t) - \dot{X}_s(t), \\ \Delta Y &= Y(t) - Y_s(t), & \Delta \dot{Y} &= \dot{Y}(t) - \dot{Y}_s(t), \\ A &= \Delta X\,\Delta\dot{X} + \Delta Y\,\Delta\dot{Y}, \\ \rho &= \sqrt{(\Delta X)^2 + (\Delta Y)^2}. \end{aligned}$$

$$\frac{\partial g}{\partial u} = \begin{bmatrix} \frac{\partial \rho}{\partial u_1} & \frac{\partial \rho}{\partial u_2} \\ \frac{\partial \dot{\rho}}{\partial u_1} & \frac{\partial \dot{\rho}}{\partial u_2} \\ \frac{\partial \phi}{\partial u_1} & \frac{\partial \phi}{\partial u_2} \end{bmatrix} = D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 2}. \tag{5}$$

3.2 Question 2: Linearization and Discrete Time Model

Now equipped with the CT Jacobians, we are able to obtain a linearized model of the system. Since the system does not have an equilibrium state, we must linearize around a nominal trajectory instead. This has the implication that the linearized system will be time-varying instead of time-invariant (LTV instead of LTI). We can linearize the system with the Jacobians given above by performing a first-order Taylor Series expansion of the nonlinear system evaluated at the nominal trajectory. This provides us with linearized perturbation dynamics for the system (a CT LTV system). Once the system is linearized around its nominal trajectory, we can then discretize the system to obtain the discrete-time linear time-varying system (a DT LTV system).

3.2.1 Nominal Trajectory

Given that the system has a nominal orbit radius $r_{nom} = 6678$ km, we can derive the following parametric equations for X and Y that describe the nominal trajectory of the system:

Nominal position:

$$r_{nom} = \sqrt{X_{nom}^2 + Y_{nom}^2}, \quad X_{nom}(t=0) = r_{nom}, \quad Y_{nom}(t=0) = 0$$

$$\hookrightarrow X_{nom} = r_{nom}\cos(\omega t)$$

$$\hookrightarrow Y_{nom} = r_{nom}\sin(\omega t)$$

Nominal velocity:
Let

$$\omega = \sqrt{\frac{\mu}{r_0^3}}$$

and given

$$\sqrt{\dot{X}_{nom}^2 + \dot{Y}_{nom}^2} = r_0\omega, \quad \dot{X}_{nom}(t=0) = 0, \quad \dot{Y}_{nom}(t=0) = r_0\omega$$

Then, taking the derivative of X_{nom} and Y_{nom} gives

$$\dot{X}_{nom} = -\omega r_{nom}\sin(\omega t)$$

$$\dot{Y}_{nom} = \omega r_{nom}\cos(\omega t)$$

Thus,

$$x_{nom} = \begin{bmatrix} X_{nom} \\ \dot{X}_{nom} \\ Y_{nom} \\ \dot{Y}_{nom} \end{bmatrix} = \begin{bmatrix} r_{nom}\cos(\omega t) \\ -\omega r_{nom}\sin(\omega t) \\ r_{nom}\sin(\omega t) \\ \omega r_{nom}\cos(\omega t) \end{bmatrix}$$

3.2.2 Linearization (CT LTV system)

Evaluating the Jacobians $\frac{\partial F}{\partial x}$, $\frac{\partial F}{\partial u}$, $\frac{\partial h}{\partial x}$, and $\frac{\partial h}{\partial u}$ at the nominal trajectory x_{nom} provides us with the following perturbation dynamics:

$$\delta\dot{x} = \frac{\partial F}{\partial x}|_{x_{nom}}\delta x + \frac{\partial F}{\partial u}|_{x_{nom}}\delta u$$

$$\delta y = \frac{\partial h}{\partial x}|_{x_{nom}}\delta x + \frac{\partial h}{\partial u}|_{x_{nom}}\delta u$$

$$\boxed{\frac{\partial F}{\partial x}|_{x_{nom}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\omega^2(1-3\cos^2(\omega t)) & 0 & \frac{3}{2}\omega^2\sin(2\omega t) & 0 \\ 0 & 0 & 0 & 1 \\ \frac{3}{2}\omega^2\sin(2\omega t) & 0 & -\omega^2(1-3\sin^2(\omega t)) & 0 \end{bmatrix}}$$

$$\boxed{\frac{\partial F}{\partial u}|_{x_{nom}} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}}$$

$$\boxed{\frac{\partial h}{\partial x} = \begin{bmatrix} \frac{\Delta X_{nom}}{\rho_{nom}} & 0 & \frac{\Delta Y_{nom}}{\rho_{nom}} & 0 \\ \frac{\rho_{nom}\Delta \dot{X}_{nom} - A(\frac{\Delta X_{nom}}{\rho_{nom}})}{\rho_{nom}^2} & \frac{\Delta X_{nom}}{\rho_{nom}} & \frac{\rho_{nom}\Delta \dot{Y}_{nom} - A_{nom}(\frac{\Delta Y_{nom}}{\rho})}{\rho_{nom}^2} & \frac{\Delta Y_{nom}}{\rho_{nom}} \\ -\frac{\Delta Y_{nom}}{\rho_{nom}^2} & 0 & \frac{\Delta X_{nom}}{\rho_{nom}^2} & 0 \end{bmatrix}}$$

where

$$\Delta X_{nom} = r_{nom}\cos(\omega t) - X_s(t), \qquad \Delta \dot{X}_{nom} = -\omega r_{nom}\sin(\omega t) - \dot{X}_s(t),$$

$$\Delta Y_{nom} = r_{nom}\sin(\omega t) - Y_s(t), \qquad \Delta \dot{Y}_{nom} = \omega r_{nom}\cos(\omega t) - \dot{Y}_s(t),$$

$$A_{nom} = \Delta X_{nom}\,\Delta \dot{X}_{nom} + \Delta Y_{nom}\,\Delta \dot{Y}_{nom},$$

$$\rho_{nom} = \sqrt{(\Delta X_{nom})^2 + (\Delta Y_{nom})^2}.$$

$$\omega = \sqrt{\frac{\mu}{r_{nom}^3}}$$

$$\boxed{\frac{\partial h}{\partial u} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}}$$

3.2.3 Discretization

Finally, we can obtain the DT LTV system

$$x_{k+1} = F(t)x_k + Gu_k$$
$$y_k = H(t)x_k + Mu_k$$

where in the LTV system, the DT matrices are calculated at each time step from:

$$F(t) = I + \Delta T \cdot A(t)$$
$$G = \Delta T \cdot B$$
$$H(t) = \frac{\partial F}{\partial u}|_{x_{nom}}$$
$$M = \frac{\partial h}{\partial u}_{x_{nom}}$$

3.3 Question 3: Discrete Time Dynamics and Measurement Simulation

3.3.1 Nonlinear Simulation

The nonlinear equations of motion were simulated through MATLAB's ode45() function. The initial conditions were perturbed from the nominal initial conditions so that the full initial conditions were:

$$\vec{x}(0) = \vec{x}_{nom}(0) + [0 \;\; 0.075 \;\; 0 \;\; -0.021]^T$$

The nonlinear measurement outputs were calculated from a custom MATLAB function that determines measurements from the equations highlighted earlier. Figs. 1 and 2 shows the dynamics and measurements from the full nonlinear simulation with no process or measurement noise.

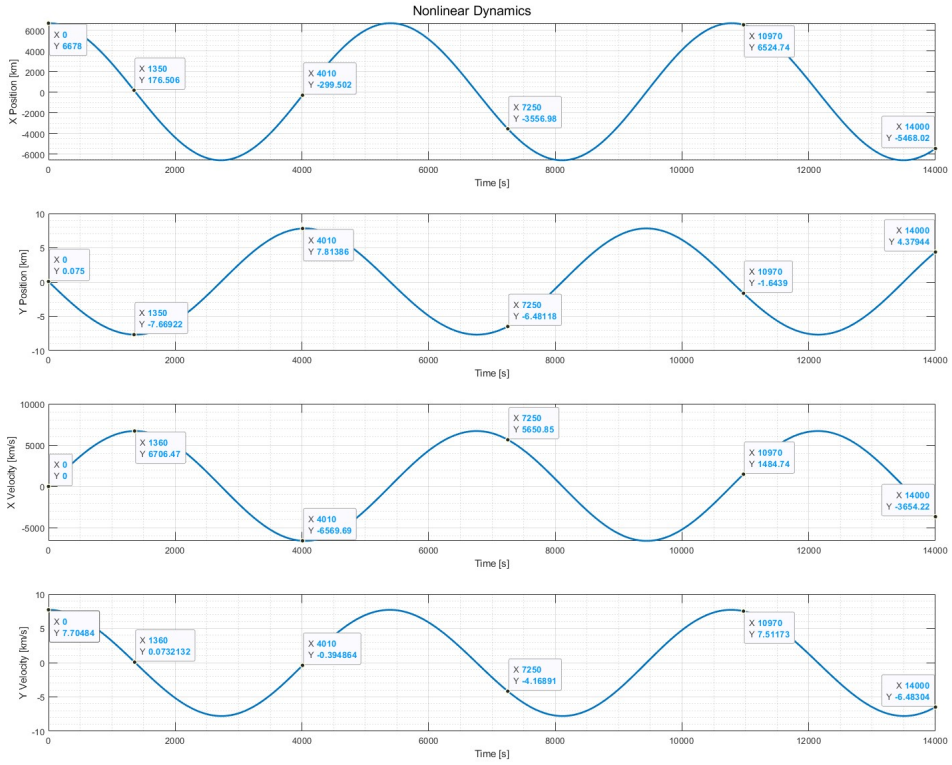

Figure 1: Full Nonlinear Dynamics

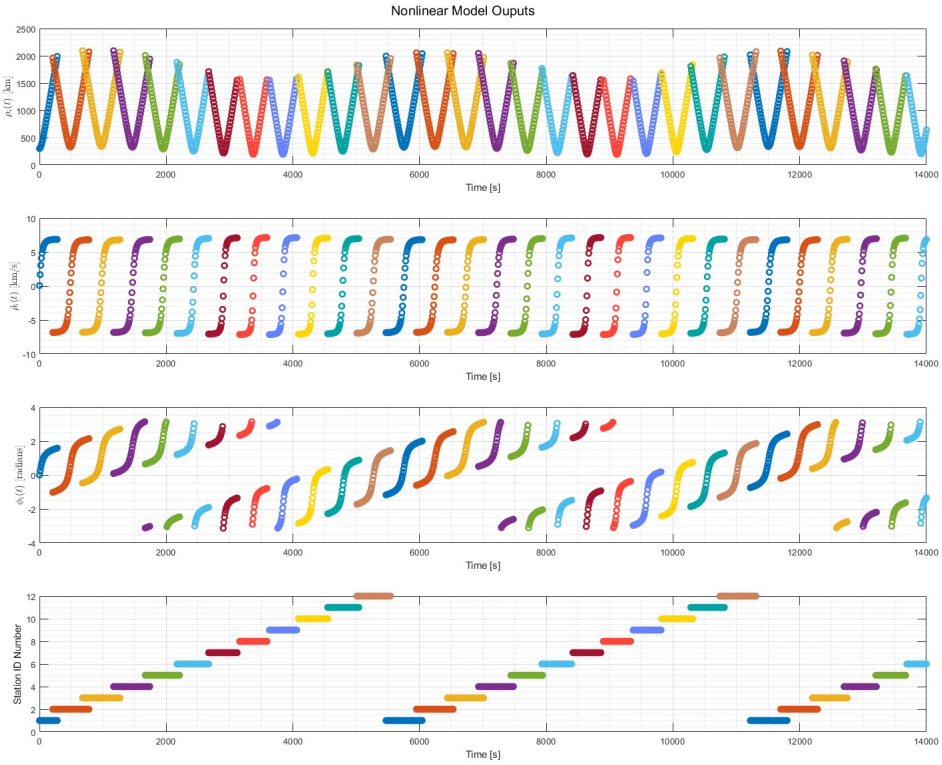

Figure 2: Full Nonlinear Outputs

3.3.2 Linearized Simulation

Utilizing the linearization of the dynamics highlighted in problems 1 and 2 above, the linearized perturbation dynamics and measurements were simulated through the DT state space equations:

$$\delta x_{k+1} = \tilde{F}_k \delta x_k + \tilde{G} \delta u_k$$
$$\delta y_{k+1} = \tilde{H}_k \delta x_{k+1}$$

The perturbation dynamics and measurements were then added back to the full, unperturbed, nominal trajectory and measurements to get the full linearized dynamics and measurements. Fig. 3 shows the resulting linearized perturbation dynamics, and figs. 4 and 5 show the full linearized outputs and measurements respectively.

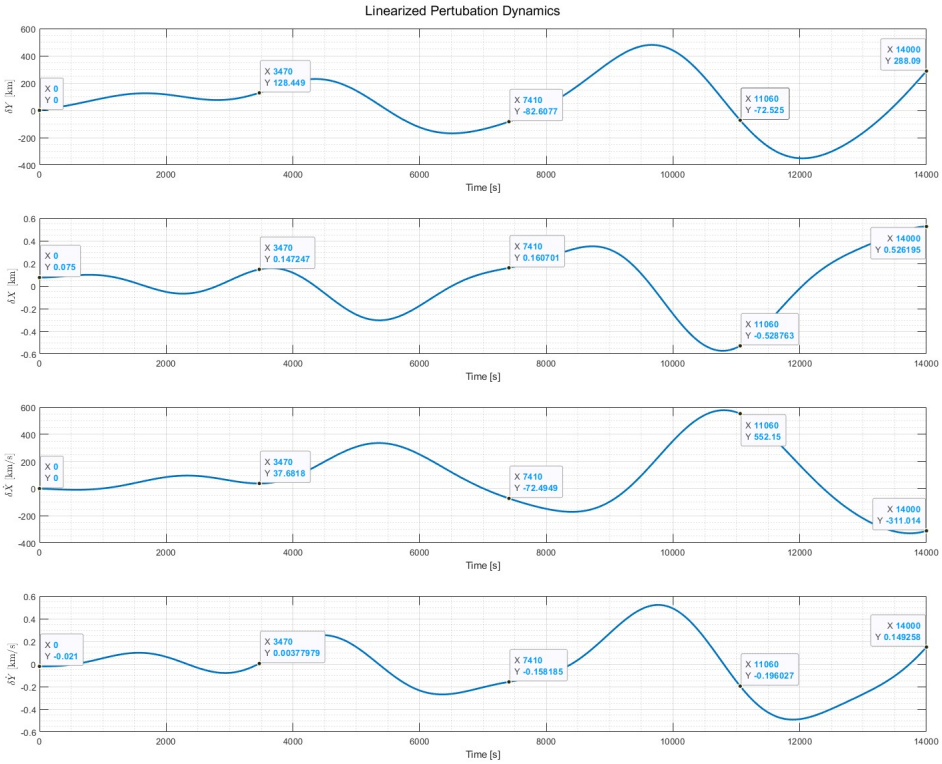

Figure 3: Linearized Perturbation Dynamics

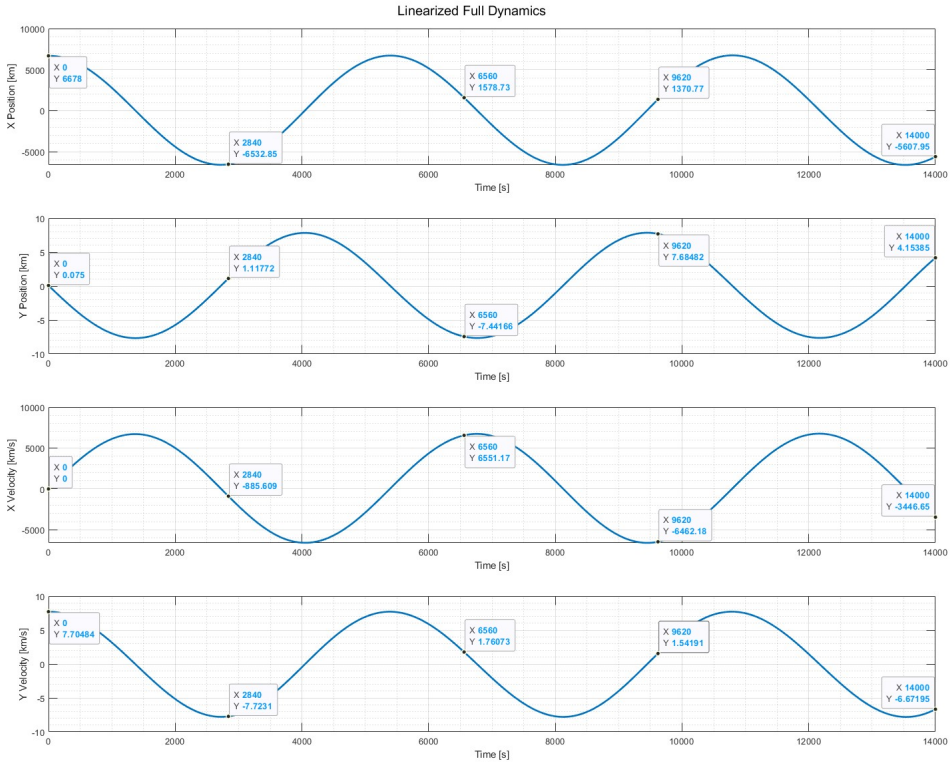

Figure 4: Linearized Full Dynamics

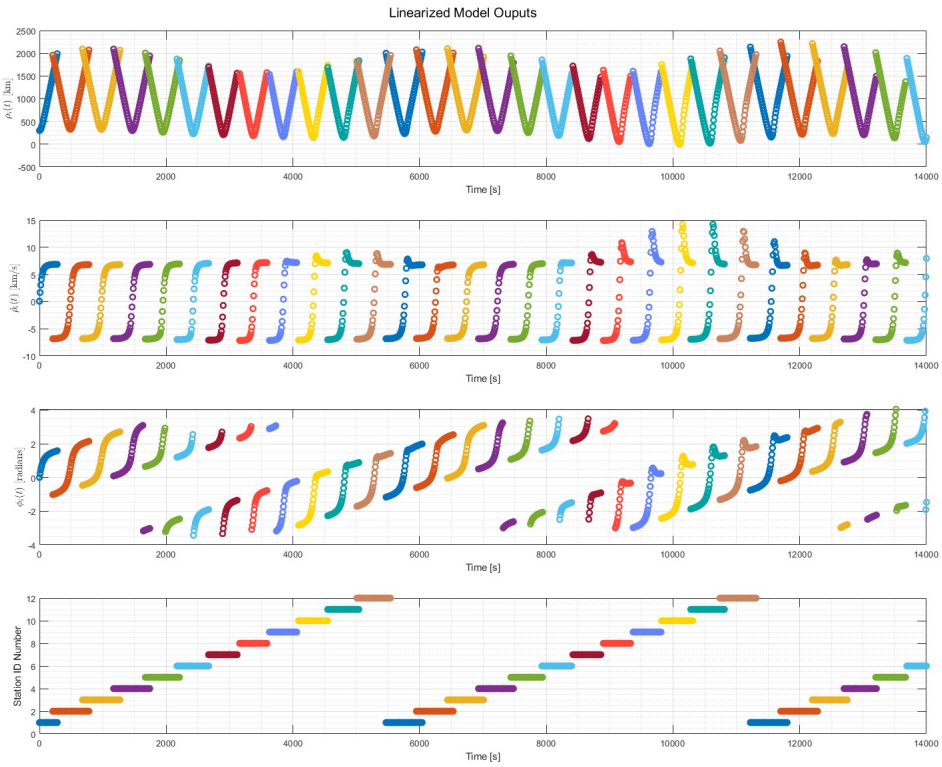

Figure 5: Full Linearized Outputs

4 Part II: Stochastic Nonlinear Filtering

To begin stochastic nonlinear filtering problems, process and measurement noise must now be considered. Looking at the nonlinear equations of motion, we can show that the continuous and discrete time process noise matrices are:

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

$$\tilde{\Omega} = \Delta T \cdot \Gamma$$

Then the simulated process noise is generated via:

$$S_k = chol(Q, 'lower') = \sqrt{Q}$$

$$x_{k+1} = x_{pert_{k+1}} + \tilde{\Omega} S_k$$

And the simulated noisy measurement data is generated via:

$$S_v = chol(R_{k+1}, 'lower') = \sqrt{R_{k+1}}$$

$$q_k \sim \mathcal{N}(0, I_{pM \times pM})$$

$$y_{k+1} = y_{pert_{k+1}} + S_v q_k$$

Where $x_{pert_{k+1}}$ and $y_{pert_{k+1}}$ are the true perturbed states and measurements respectively. Figs. 6 and 7 show the resulting error in the perturbed state due to process noise and the resulting output measurements including measurement noise respectively for a single simulation.

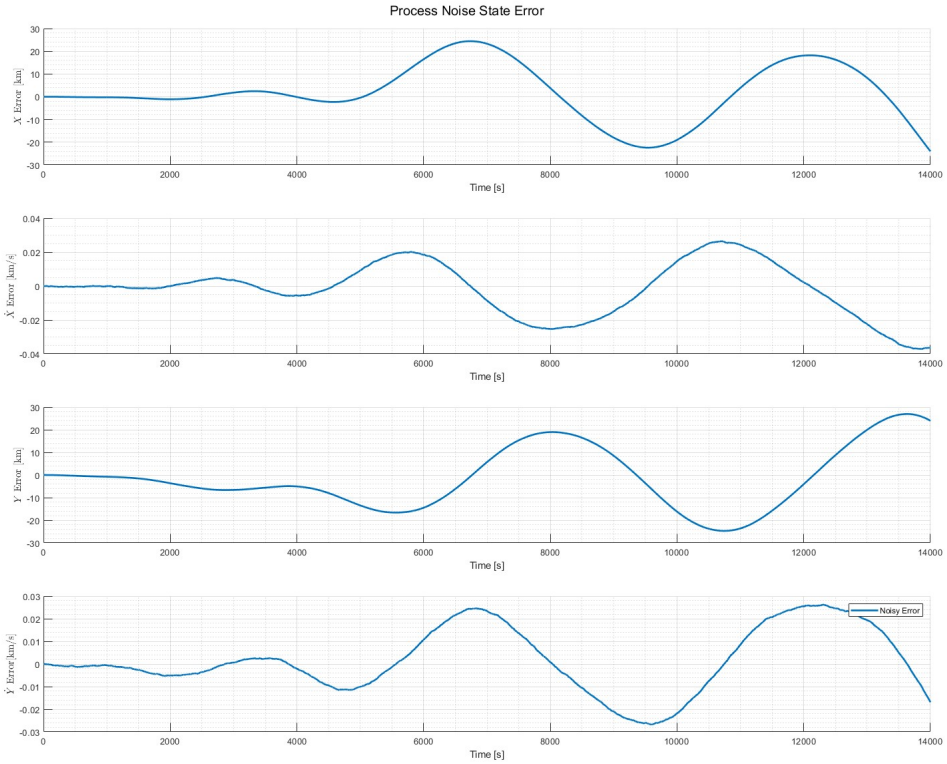

Figure 6: Simulated Process Noise State Error

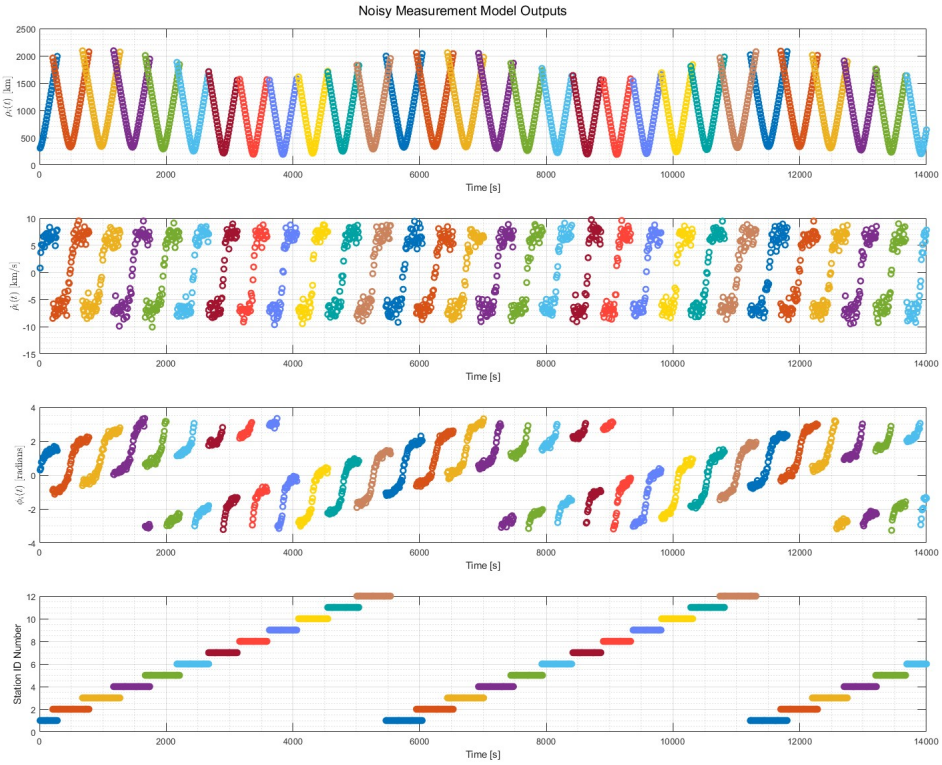

Figure 7: Simulated Measurement Noise Outputs

4.1 NEES/NIS Methodology

The Normalized Estimation Error Squared (NEES) and Normalized Innovation Squared (NIS) Chi-Squared metrics are statistical tests that are used to ensure that a KF's state errors/measurement residuals make sense for the filter's given system, measurements, and noise models. The NEES and NIS statistics provide us with a method to determine if our KFs are statistically consistent and therefore working properly.

The NEES and NIS metrics are derived as follows. Let

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

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

where $e_{x,k}$ is a vector of state estimation errors with respect to ground truth x_k, and $e_{y,k}$ is a vector of measurement residuals w.r.t. observations y_k for each time step k.

The normalized magnitudes of these vectors over time are then calculated as

$$\epsilon_{x,k} = e_{x,k}^T (P_k^+)^{-1} e_{x,k} \rightarrow \textbf{NEES at time k}$$

$$\epsilon_{y,k} = e_{y,k}^T (S_k)^{-1} e_{y,k} \rightarrow \textbf{NIS at time k}$$

It then follows that if the KF works properly as per the state space model and noise specifications, then we must have

1. $\epsilon_{x,k} \sim \chi_n^2 \; \forall \; k$, where $E[\epsilon_{x,k}] = n, \text{var}(\epsilon_{x,k}) = 2n$
2. $\epsilon_{y,k} \sim \chi_p^2 \; \forall \; k$, where $E[\epsilon_{y,k}] = p, \text{var}(\epsilon_{y,k}) = 2p$

where n is the total number of states and p is the total number of measurements at time step k.

In order to assess whether the above two conditions hold for our KF, we utilize Monte Carlo testing. With our Truth Model (TM) simulator, we perform N Monte Carlo simulations. The TM provides us with the input measurements y_k to our KF and with the ground truth values x_k at each time step k. Then, let

$$\bar{\epsilon}_{x,k} = \frac{1}{N} \sum_{i=1}^{N} \epsilon_{x,k}^i$$

$$\bar{\epsilon}_{y,k} = \frac{1}{N} \sum_{i=1}^{N} \epsilon_{y,k}^i$$

where $\epsilon_{x,k}^i$ is NEES and $\epsilon_{y,k}^i$ is NIS for Monte Carlo sim i at time step k. Then, if $\bar{\epsilon}_{x,k} \rightarrow n$ and $\bar{\epsilon}_{y,k} \rightarrow p$ as $N \rightarrow \text{inf}$, then the NEES and NIS tests support that the KF is working properly.

Finally, we can assess that $\bar{\epsilon}_{x,k} \rightarrow n$ and $\bar{\epsilon}_{y,k} \rightarrow p$ as $N \rightarrow \text{inf}$ (i.e. that NEES and NIS follow a χ^2 distribution) with a significance level α as follows:

For a two-sided hypothesis test at significance level α, we compute bounds r_1 and r_2 such that:

$$P\left(r_1 \leq \bar{\epsilon}_{x,k} \leq r_2\right) = 1 - \alpha$$

where

$$r_1 = \frac{1}{N} \chi^2_{(N \cdot n, \alpha/2)}$$

and

$$r_2 = \frac{1}{N} \chi^2_{(N \cdot n, 1-\alpha/2)}$$

Then, if the filter is consistent and working correctly, $(1-\alpha) \cdot 100\%$ of the $\bar{\epsilon}_{x,k}$ values over time should fall within $[r_1, r_2]$.

4.2 Linearized Kalman Filter (LKF)

4.2.1 Algorithm

The LKF estimates the state perturbations around a priori nominal state trajectory. These perturbations can then be added to the nominal state trajectory to get the full perturbed state trajectory. The same applies to the measurements. As mentioned in problem 2, the linearized Jacobians are calculated at each time step in the simulation using an Euler iteration scheme. The algorithm for the LKF is as follows:

1. Initialize:

$$\hat{\delta x}^{+}(0) = [0 \quad 0.075 \quad 0 \quad -0.021]^T$$

$$P^{+}(0) = diag([100 \quad 1 \quad 100 \quad 1])$$

2. Time Update/Prediction Step:

$$\hat{\delta x}_{k+1}^{-} = \tilde{F}_k \hat{\delta x}_k^{+} + \tilde{G}\delta u_k$$

$$P_{k+1}^{-} = \tilde{F}_k P_k^{+} \tilde{F}_k^T + \tilde{\Omega} Q \tilde{\Omega}^T$$

3. Measurement Update/Correction Step:

$$R_{k+1} = \begin{bmatrix} R & 0 & \dots & 0 \\ 0 & R & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & R \end{bmatrix}_{pM \times pM}$$

Where M is the number of visible stations at time t_k

$$K_{k+1} = P_{k+1}^{-} \tilde{H}_{k+1}^T \left(\tilde{H}_{k+1} P_{k+1}^{-} \tilde{H}_{k+1}^T + R_{k+1} \right)^{-1}$$

$$\hat{\delta x}_{k+1}^{+} = \hat{\delta x}_{k+1}^{-} + K_{k+1} \left(\delta y_{k+1} - \tilde{H}_{k+1} \hat{\delta x}_{k+1}^{-} \right)$$

$$P_{k+1}^{+} = \left(I - K_{k+1} \tilde{H}_{k+1} \right) P_{k+1}^{-}$$

$$\delta y_{k+1} = y_{k+1} - y_{k+1}^{*} = y_{k+1} - h(x_{k+1}^{*})$$

Figs. 8 and 9 show the state and error outputs after implementing the Linearized Kalman Filter on the perturbed nonlinear truth trajectory, using noisy measurements derived from the perturbed state propagation. The estimates follow the sinusoidal motion, but the error exhibits some oscillations as time goes along. This is due to the LKF being based on first-order linearizations of the dynamics and measurement models about the nominal orbit, so any deviation of the true state from this linearization point introduces modeling error that appears as those oscillatory spikes on the plots.

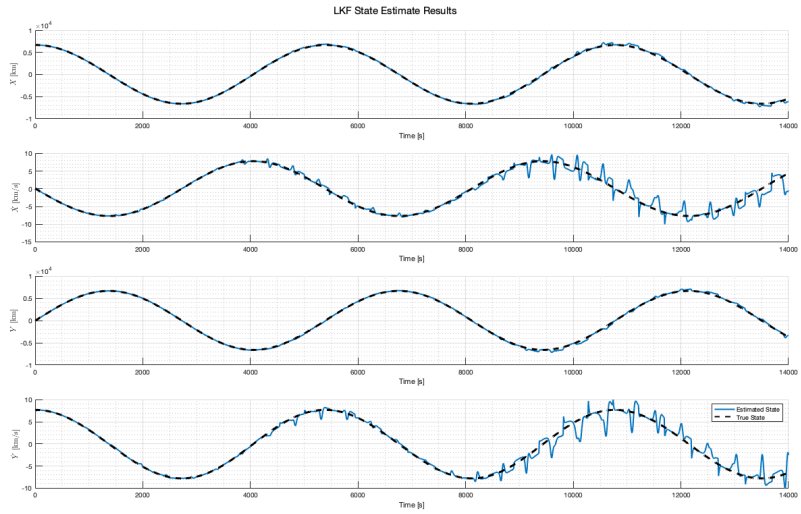

Figure 8: LKF State Estimation Errors.

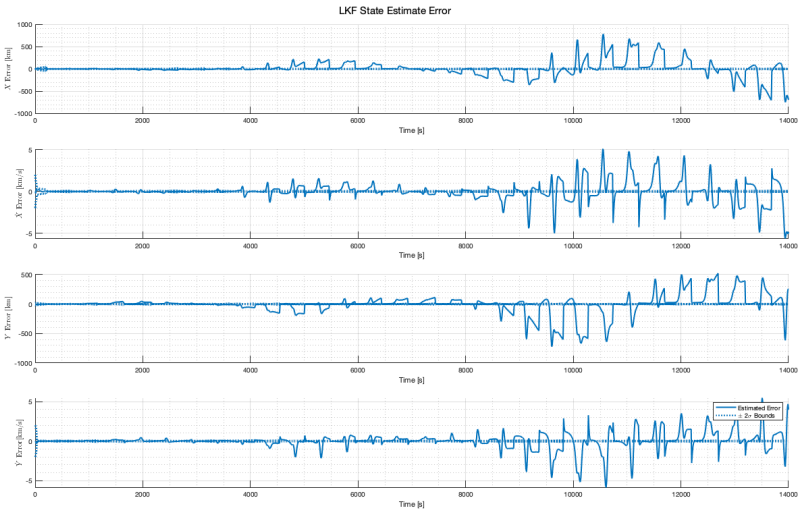

Figure 9: LKF State Error Norm $\|e(t)\|$.

4.2.2 Tuning and Truth Model Testing

Tuning the LKF was a difficult process. The tuning process for the LKF was done through the NEES/NIS tests as described in section 4.1. The goal was to adjust Q to the point where the

filter's predicted covariance produced NEES and NIS values that lay within the corresponding chi-square bounds, indicating that the assumed uncertainty in the filter matched the actual estimation error observed during Monte-Carlo simulation. This proved to be impossible. We started with $Q = 10 * Q_{true}$ and increased it from there. No matter how large we made Q, the true error was always above the chi-square bounds. Figs. 10 and 11 show that even with an extremely high Q value (Like $Q = 10000 * Q_{true}$ the NEES and NIS metrics remain orders of magnitude above the admissible region. This indicates that the LKF is extremely over confident due to its predicted covariance being far too small compared to the actual errors.

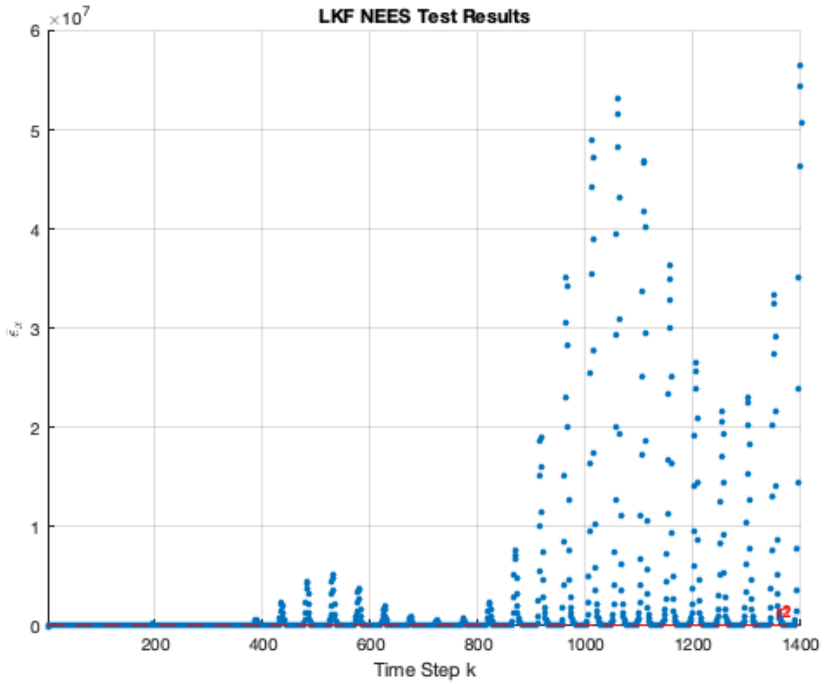

Figure 10: LKF NEES Test: $Q = 10,000 * Q_{true}$

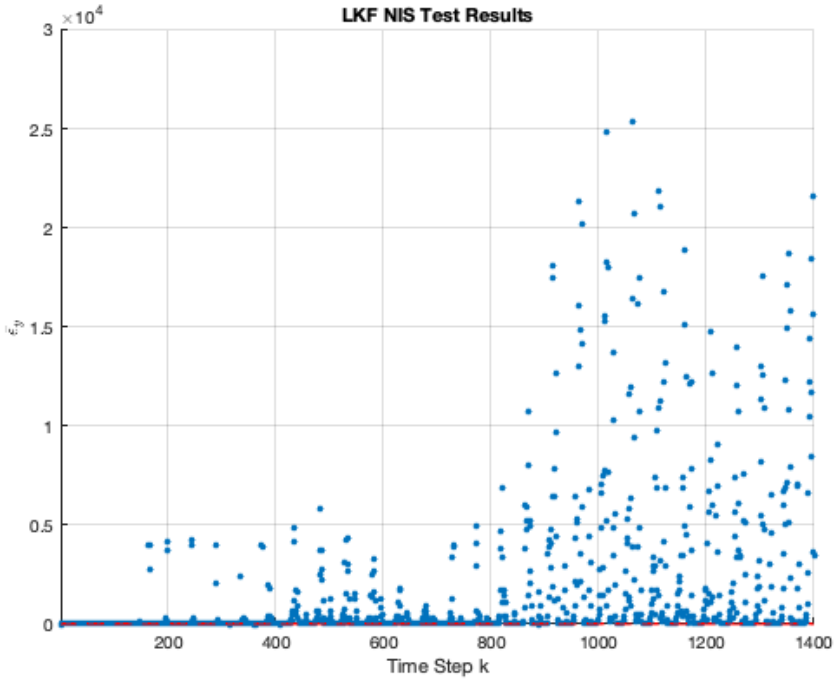

Figure 11: LKF NIS Test: $Q = 10,000 * Q_{true}$

4.3 Extended Kalman Filter (EKF)

4.3.1 Algorithm

The EKF estimates the full nonlinear state by propagating the mean of the state through the nonlinear dynamics while locally linearizing the system and measurement models at every time step. The EKF algorithm integrates the nonlinear equations of motion at each time step in order to obtain the prior measurement. These measurements are then evaluated in derived Jacobians. See eqs. (6) - (19) for the full implementation of this algorithm.

$$\dot{x}(t) = f\big(x(t), u(t), w(t)\big), \qquad y(t) = h\big(x(t), v(t)\big), \tag{6}$$

with $x = [X, \dot{X}, Y, \dot{Y}]^T \in \mathbb{R}^4$.

$$\hat{x}_0^+ = \hat{x}_0, \qquad P_0^+ = P_0. \tag{7}$$

For each time step $k = 0, \ldots, N-1$ with $t_{k+1} = t_k + \Delta t$:

$$\hat{x}_{k+1}^- \approx \text{nonlinear integration of } \dot{x} = f(x, u, w) \text{ with } w = 0, \tag{8}$$

Compute continuous-time Jacobians at the predicted state:

$$A_{c,k} = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k+1}^-}, \qquad \Gamma_{c,k} = \left.\frac{\partial f}{\partial w}\right|_{\hat{x}_{k+1}^-}. \tag{9}$$

First–order discrete-time approximations:

$$F_k = I_4 + \Delta t\, A_{c,k}, \qquad \Omega_k = \Delta t\, \Gamma_{c,k}. \tag{10}$$

Predicted (a priori) covariance:

$$P_{k+1}^{-} = F_k P_k^{+} F_k^T + \Omega_k Q \Omega_k^T. \tag{11}$$

For the set of visible stations at time t_{k+1}, let m be the number of stations and stack their measurements:

$$y_{k+1} = \begin{bmatrix} y_{k+1}^{(s_1)} \\ \vdots \\ y_{k+1}^{(s_m)} \end{bmatrix}, \qquad \hat{y}_{k+1}^{-} = \begin{bmatrix} h^{(s_1)}\big(\hat{x}_{k+1}^{-}, 0\big) \\ \vdots \\ h^{(s_m)}\big(\hat{x}_{k+1}^{-}, 0\big) \end{bmatrix}. \tag{12}$$

The corresponding stacked measurement Jacobian is

$$H_k = \begin{bmatrix} H_{s_1} \\ \vdots \\ H_{s_m} \end{bmatrix}, \qquad H_{s_i} = \left.\frac{\partial h^{(s_i)}}{\partial x}\right|_{\hat{x}_{k+1}^{-}} \in \mathbb{R}^{3\times 4}, \tag{13}$$

and the stacked measurement-noise covariance is

$$R_k = I_m \otimes R \in \mathbb{R}^{3m\times 3m}. \tag{14}$$

Innovation:

$$\tilde{e}_{y,k+1} = y_{k+1} - \hat{y}_{k+1}^{-}. \tag{15}$$

Innovation covariance:

$$S_{k+1} = H_k P_{k+1}^{-} H_k^T + R_k. \tag{16}$$

Kalman gain:

$$K_{k+1} = P_{k+1}^{-} H_k^T S_{k+1}^{-1}. \tag{17}$$

Updated (a posteriori) state estimate:

$$\hat{x}_{k+1}^{+} = \hat{x}_{k+1}^{-} + K_{k+1}\, \tilde{e}_{y,k+1}. \tag{18}$$

Updated covariance:

$$P_{k+1}^{+} = \big(I_4 - K_{k+1} H_k\big) P_{k+1}^{-}. \tag{19}$$

Figs. 12 and 13 show the state estimate and corresponding estimation error obtained using the Extended Kalman Filter (EKF). In this case, the EKF utilizes the full nonlinear orbital dynamics and measurement models during both the prediction and correction steps, allowing the filter to more accurately capture the true system behavior. As a result, the EKF rapidly drives the initial estimation error to zero and maintains a tightly bounded error for the remainder of the simulation. The state trajectories closely track the nonlinear truth model despite the presence of process and measurement noise.

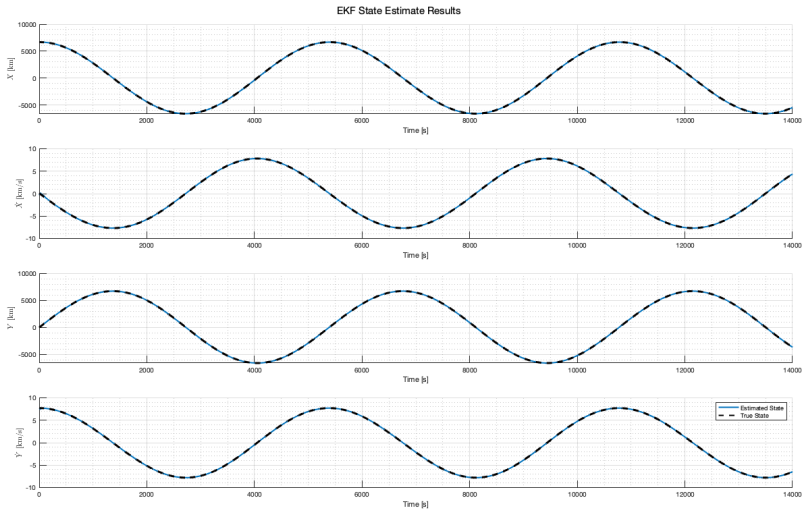

Figure 12: EKF State Estimation Errors.

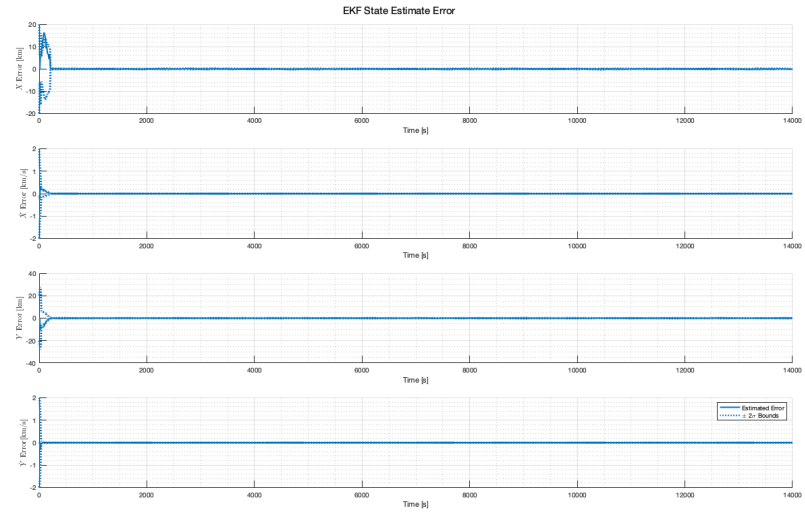

Figure 13: EKF State Error Norm $\|e(t)\|$.

4.3.2 Tuning and Truth Model Testing

The tuning process for the EKF was similar to the LKF. Truth model tests with 50 Monte Carlo simulations were run, calculating NEES and NIS values for each test to evaluate the validity of the

EKF. The values that were tuned were $P^+(0)$ and Q. First, $P^+(0)$ was tuned to remove the large spike in the NEES plot (highly overconfident) at the beginning of the simulation (k=0). Once this initial NEES spike was fixed with the tuning of $P^+(0)$, different Q values were tested. An α of 0.05 was chosen in order to represent a 95% significance level. Initially, with Q set to the true Q value provided in the assignment, the NEES test showed a bias to above the upper bound r_2, and NIS was too small, suggesting that the model was overconfident.

After many iterations of tuning Q, the best result achieved was with $P^+(0) = diag([1.35 \;\; 1.35 \times 10^{-3} \;\; 1.35 \;\; 1.35 \times 10^{-3}])$ and $Q = 0.9 \cdot Q_{true}$. This tuning configuration achieved a NEES value of 0.905 and a NIS value of 0.906. These values do not reach the 95% threshold necessary to declare the KF consistent with the significance level α. Deviation from the theoretical 95% could possibly be attributed to linearization errors inherent in the EKF and measurement model approximations during station visibility transitions.

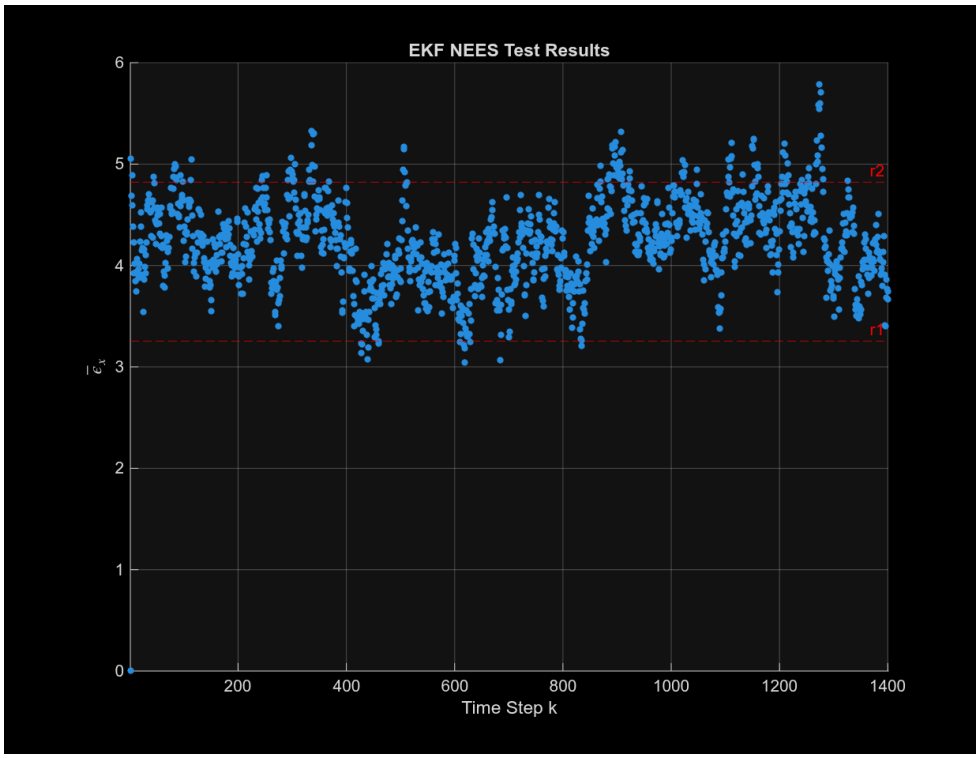

Figure 14: EKF NEES Test: $Q = 0.9 \cdot Q_{true}$

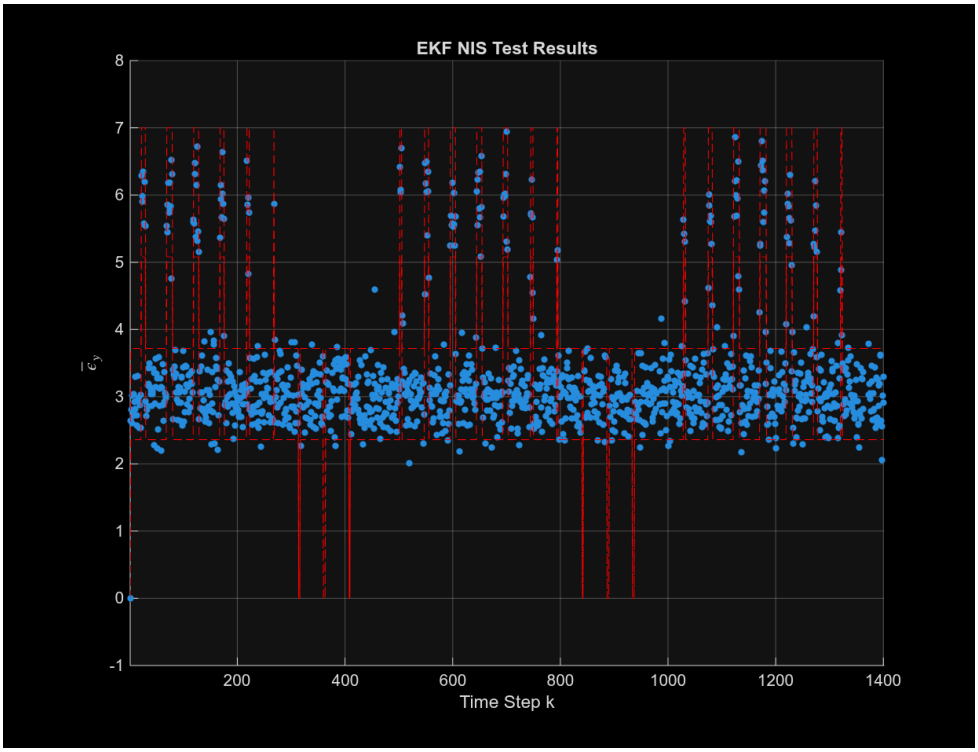

Figure 15: EKF NIS Test: $Q = 0.9 \cdot Q_{true}$

4.4 LKF and EKF Comparison

As seen in Fig. 16, the 2σ curves are essentially right on top of the state estimate curves. To show the extent of the magnitudes of these 2σ values, we also included plots with 100σ values. The 2σ values are so tight with the state estimate curves due to the extreme overconfidence of the LKF as explained in section 4.2.2. This goes to show that the LKF is not the proper model to use for this problem because the LKF thinks it knows the state almost perfectly, even though the actual estimation error is much larger.

The EKF vastly outperforms the LKF, especially as time increases as seen in fig. 17. The EKF shows $\pm 2\sigma$ bounds much smaller than the LKF that stay consistent throughout the entire simulation. This is due to the EKF's ability to handle the system's nonlinearities by re-linearizing around its current state estimate at each time step, rather than relying on a fixed nominal trajectory. Therefore, the EKF does a much better job than the LKF at estimating the state trajectory.

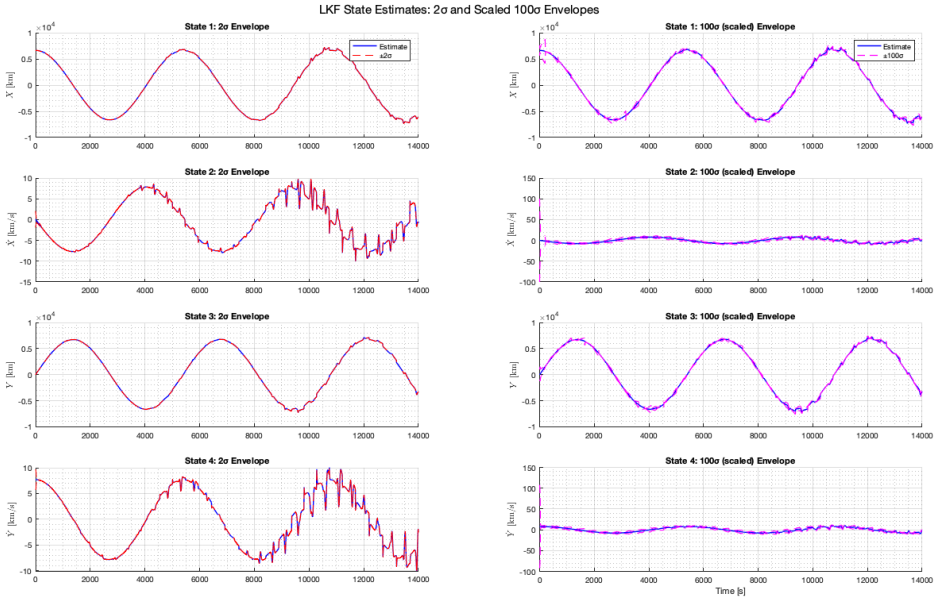

Figure 16: LKF σ Plots

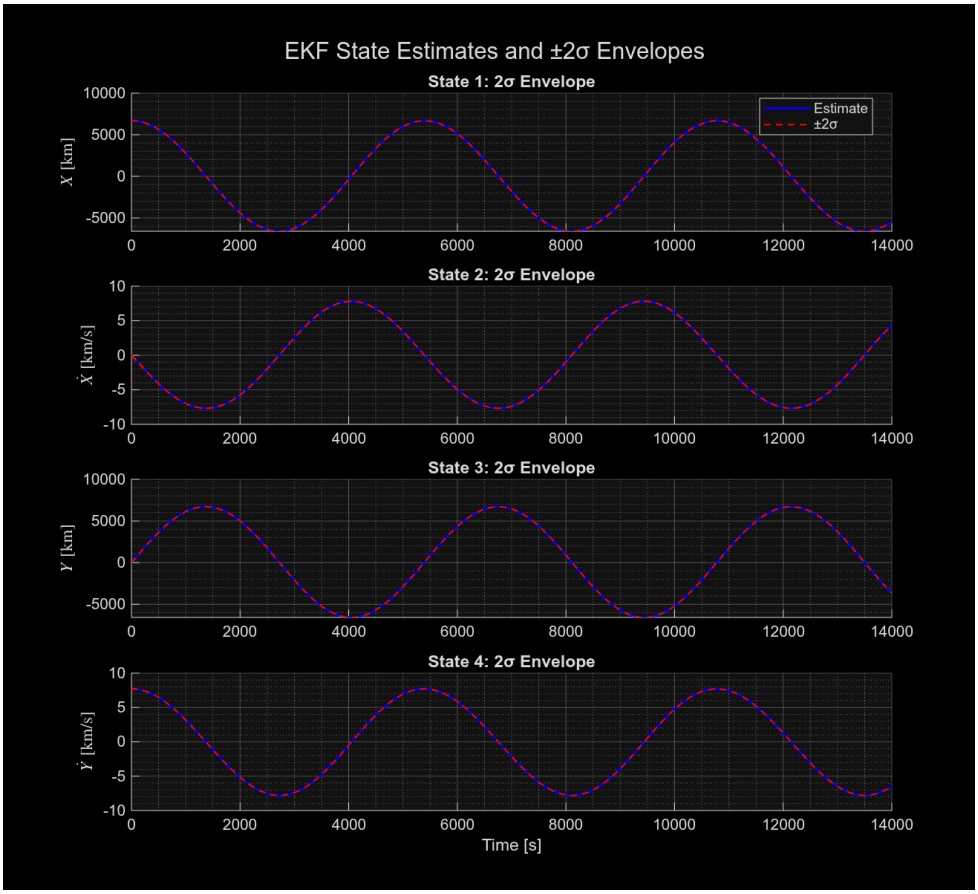

Figure 17: EKF σ Plots

5 Part III: Advanced Questions

5.1 State Estimation Haiku

What is my true state?
Kalman will estimate it.
My best observer.

5.2 Unscented Kalman Filter (UKF)

5.2.1 Algorithm

Unlike the LKF and EKFs, the UKF does not utilize Jacobians to determine state estimations, it utilizes simulation-based sampling of sigma points that are propagated through nonlinear dynamic and measurement equations. This means that the higher order terms of the Taylor-series expansions that are discarded in linearization are accounted for. The UKF algorithm is as follows:

1. Calculate Constants:

$$w_m^0 = \frac{\lambda}{n+\lambda}, \qquad w_m^i = \frac{1}{2(n+\lambda)}, \quad \text{for } i = 1, \ldots, 2n,$$

$$w_c^0 = \frac{\lambda}{n+\lambda} + 1 - \alpha^2 + \beta, \qquad w_c^i = w_m^i, \qquad \text{for } i = 1, \ldots, 2n,$$

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

Where typically $\kappa = 0$, $\beta = 2$ and $\alpha \in [10^{-4} \;\; 1]$

2. Initialize:

$$\hat{x}^+(0) = [x_{nom}(0) + \delta x(0)]^T$$

$$P^+(0) = diag([2 \;\; 2 \times 10^{-3} \;\; 2 \;\; 2 \times 10^{-3}])$$

3. Time Update/Prediction Step:

   (a) Generate 2n+1 sigma points:

$$S_k = chol(P_k^+, 'lower') = \sqrt{P_k^+}$$

$$\chi_k^0 = \hat{x}_k^+$$

$$\chi_k^i = \begin{cases} \hat{x}_k^+ + \left(\sqrt{n+\lambda}\right) \; S_k^{j,T}, & \text{for } i = 1, \ldots, n, \; j = 1, \ldots, n, \\ \hat{x}_k^+ - \left(\sqrt{n+\lambda}\right) \; S_k^{j,T}, & \text{for } i = n+1, \ldots, 2n, \; j = 1, \ldots, n. \end{cases}$$

Where $S_k^j k$ is the j^{th} row of S_k

   (b) Propagate each χ_k^i through the nonlinear dynamics to get $\bar{\chi}_{k+1}^0$ and $\bar{\chi}_{k+1}^i$.

   (c) Recombine resultant points to get predicted mean and covariance:

$$\hat{x}_{k+1}^- \approx \sum_{i=0}^{2n} w_m^i \cdot \bar{\chi}_{k+1}^i$$

$$P_{k+1}^- \approx \sum_{i=0}^{2n} w_c^i \cdot (\bar{\chi}_{k+1}^i - \hat{x}_{k+1}^-)(\bar{\chi}_{k+1}^i - \hat{x}_{k+1}^-)^T + \tilde{\Omega} Q_k \tilde{\Omega}^T$$

4. Measurement Update/Correction Step:

   (a) Determine size of R_{k+1}:

$$R_{k+1} = \begin{bmatrix} R & 0 & \dots & 0 \\ 0 & R & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & R \end{bmatrix}_{pM \times pM}$$

   Where M is the number of visible stations at time t_k

   (b) Generate 2n+1 (new) sigma points:

$$\bar{S}_{k+1} = chol(P^-_{k+1}, ' lower') = \sqrt{P^-_{k+1}}$$

$$\chi^0_{k+1} = \hat{x}^-_{k+1}$$

$$\chi^i_{k+1} = \begin{cases} \hat{x}^-_{k+1} + \left(\sqrt{n+\lambda}\right)\ \bar{S}^{j,T}_{k+1}, & \text{for } i = 1, \dots, n,\ j = 1, \dots, n, \\ \hat{x}^-_{k+1} - \left(\sqrt{n+\lambda}\right)\ \bar{S}^{j,T}_{k+1}, & \text{for } i = n+1, \dots, 2n,\ j = 1, \dots, n. \end{cases}$$

   Where $\bar{S}^j_{k+1}$ is the j^{th} row of $\bar{S}_{k+1}$

   (c) Propagate each χ^i_{k+1} through the nonlinear measurements to get γ^0_{k+1} and γ^i_{k+1}.

   (d) Recombine resultant points to get predicted mean and covariance:

$$\hat{y}^-_{k+1} \approx \sum_{i=0}^{2n} w^i_m \cdot \gamma^i_{k+1}$$

$$P^-_{yy,k+1} \approx \sum_{i=0}^{2n} w^i_c \cdot (\gamma^i_{k+1} - \hat{y}^-_{k+1})(\gamma^i_{k+1} - \hat{y}^-_{k+1})^T + R_{k+1}$$

   (e) Get state-measurement cross-covariance matrix:

$$P_{xy,k+1} \approx \sum_{i=0}^{2n} w^i_c \cdot (\chi^i_{k+1} - \hat{x}^-_{k+1})(\gamma^i_{k+1} - \hat{y}^-_{k+1})^T$$

   (f) Estimate Kalman gain matrix:

$$K_{k+1} \approx P_{xy,k+1}[P^-_{yy,k+1}]^{-1}$$

   (g) Perform state and covariance update:

$$\hat{x}^+_{k+1} = \hat{x}^-_{k+1} + K_{k+1}(y_{k+1} - \hat{y}^-_{k+1})$$

$$P^+_{k+1} = P^-_{k+1} - K_{k+1} P_{yy,k+1} K^T_{k+1}$$

5.2.2 Tuning and Truth Model Testing

Unlike in the LKF and EKF, the UKF has an additional tuning knob in α. For this section, to distinguish between the α in the UKF and the α in the NESS/NIS tests, the α in the NEES/NIS tests will be designated as α_{TMT}, and $\alpha_{TMT} = 0.05$ was used for the UKF. The tuning process started with setting $Q_{KF} = Q_{true}$, $P^+(0) = diag([10\ \ 1\ \ 10\ \ 1])$, and $\alpha = 1 \times 10^{-4}$, which makes it almost like the EKF. Truth model tests with 50 Monte Carlo simulations were ran and NEES/NIS test were performed to evaluate the tuning of the filter. The first thing tuned was $P^+(0)$, as there were large spikes of values of $\bar{\epsilon}_x$ and $\bar{\epsilon}_y$ due to an overestimation of initial covariance. Once the spike was reduced heavily, Q_{KF} was tuned to get the majority of the $\bar{\epsilon}_x$ within the $r_{1,2}$ bounds.The final step was slowing raising α until the best possible results were achieved. The final tune was given by $P^+(0) = diag([2\ \ 2 \times 10^{-3}\ \ 2\ \ 2 \times 10^{-3}])$, $Q_{KF} = 1.025 \cdot Q_{true}$ and $\alpha = 0.05$. Figs. 18 and 19 show the tuned UKF NEES and NIS test respectively. The tuned NEES plot had 95.5% of points lie within $r_{1,2}$ bounds and the NIS plot had 92.4% of points lie within the $r_{1,2}$ bounds.

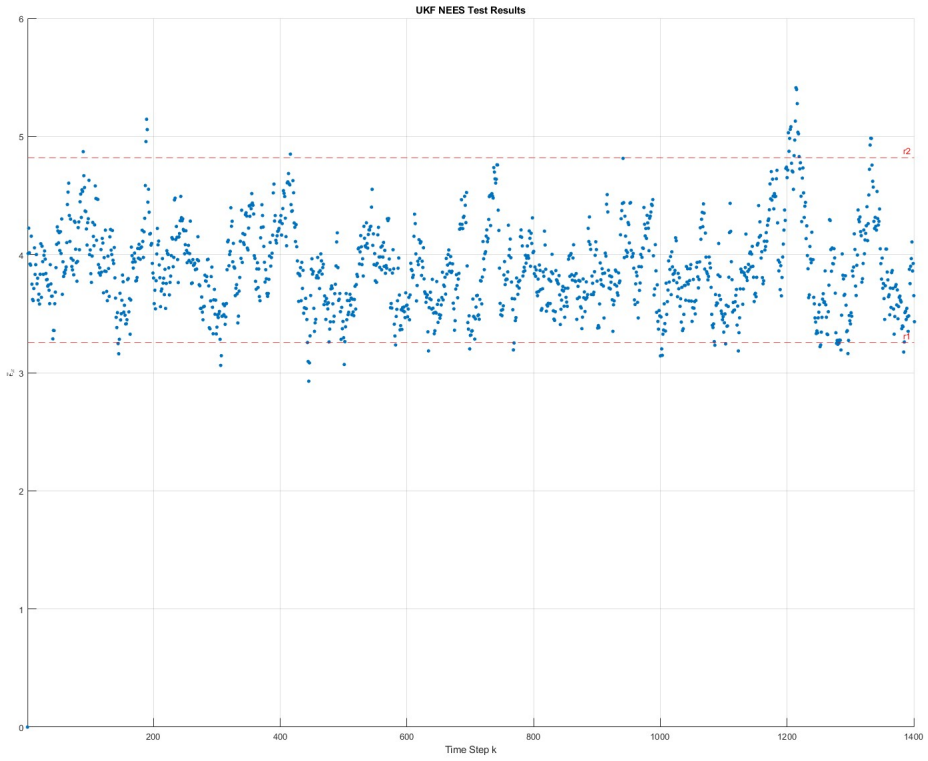

Figure 18: Tuned UKF NEES Test Results

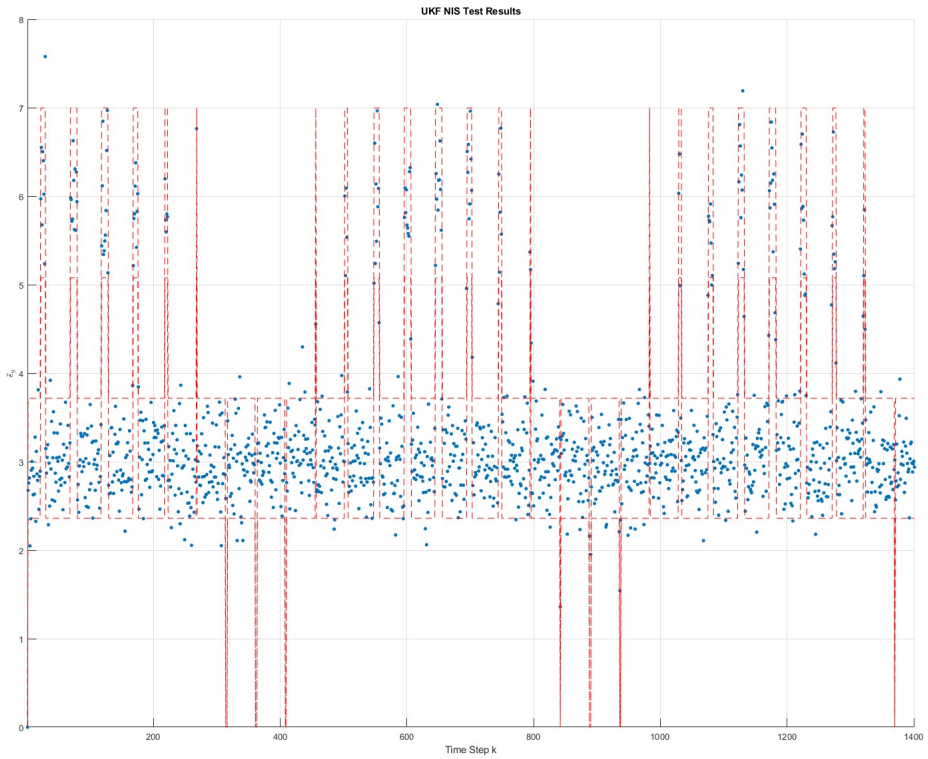

Figure 19: Tuned UKF NIS Test Results

5.2.3 UKF Comparison to LKF and EKF

It has already been stated that the LKF does not work well when the orbit starts to heavily deviate from its nominal trajectory, so the main comparison of interest here is between the UKF and EKF. Utilizing the given data log, 20 shows the state estimation and the $\pm 2\sigma$ bounds. When comparing fig. ... (the EKF results) to the fig.20, we can see that the tune is very similar between the two. This is to be expected with this problem due to the fact that the higher order terms that are disregarded by the EKF and accounted for in the UKF are small, so thier total impact on the filter's estimated states is small. However, through the statistics on the EKF and UKF NEES and NIS testing, it can be seen that the tune on the UKF is slightly better, so it is expected to have slightly higher confidence in the state estimations for the UKF than the EKF.

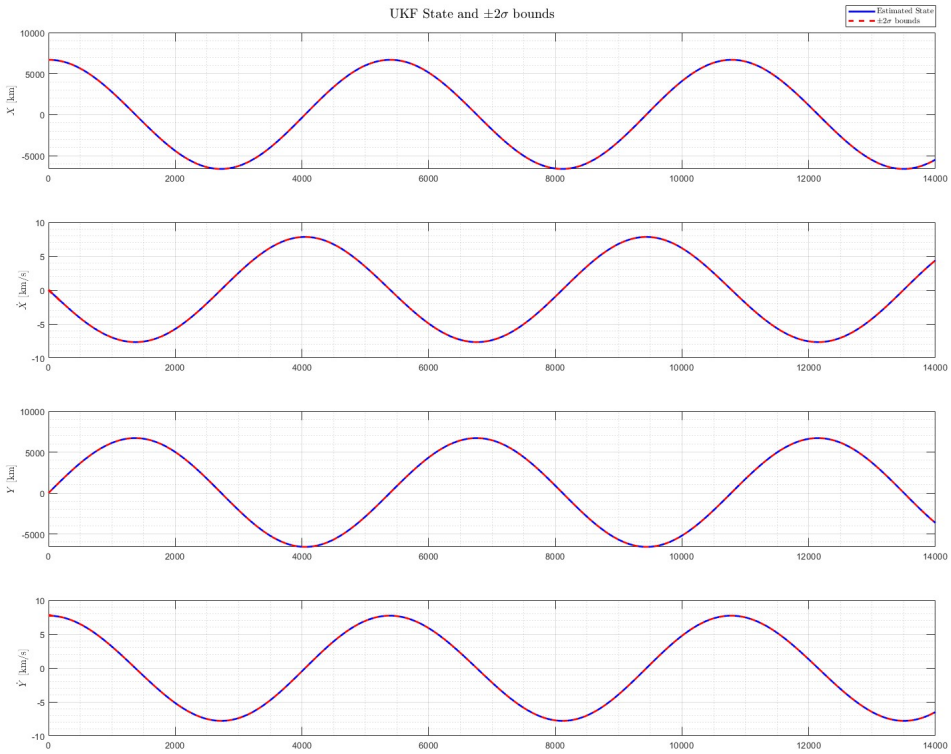

Figure 20: UKF State Estimates and $\pm 2\sigma$ bounds

6 Appendix

6.1 Code: Main Script

```
clc; clear; close all

%% Simulate Non-Linear ODEs and Measured Outputs

% Constants
mu = 398600; % km^3/s^2
r0 = 6678; % km
Ydot0 = r0 * sqrt(mu/r0^3); % km/s
RE = 6378; % km
wE = (2*pi)/86400; % rad/s
w = sqrt(mu/r0^3);

% ICs
nom_var0 = [r0 0 0 Ydot0]';
delta_x0 = [0 0.075 0 -0.021]';
pert_var0 = nom_var0 + delta_x0;

% Simulation time
delta_t = 10; %s
% T = (2*pi)/(sqrt(mu/r0^3));
% tspan = 0:delta_t:T; % s
tspan = 0:delta_t:14000; % s

% ode45 call
options = odeset('RelTol',1e-6,'AbsTol',1e-9);
[~,x_nom] = ode45(@(tspan,var0) OrbitEOM(tspan,var0,mu),tspan,nom_var0,options);

% ode45 call - pert
[t,x_pert] = ode45(@(tspan,var) OrbitEOM(tspan,var,mu),tspan,pert_var0,options);

% Get outputs
[output_var,station_vis] = MeasuredOutput(t,x_pert,RE,wE,true);
[output_var_nom,~] = MeasuredOutput(t,x_nom,RE,wE,false);

% Convert into cell array - filter nominal measurements to match NL idxs
N = length(t);
ycell_nom = cell(N,1);
ycell_pert = cell(N,1);
station_vis_cell = cell(N,1);
for k = 1:N-1
    yrowsKeep = ~any(isnan(output_var(:,k+1)),2); % rows that contain NO NaNs
    stationrowsKeep = ~any(isnan(station_vis(:,k+1)),2); % rows that contain NO NaNs
```

```
    y1 = output_var_nom(yrowsKeep,k+1);
    y2 = output_var(yrowsKeep,k+1);
    station = station_vis(stationrowsKeep,k+1);
    ycell_nom{k+1} = y1;
    ycell_pert{k+1} = y2;
    station_vis_cell{k+1} = station;
end

y_pert = ycell_pert;
y_nom = ycell_nom;
station_vis = station_vis_cell;

% Concatanate Outputs/Stations
NL_outputs = [y_nom station_vis];

%% CT Dynamics
Abar = @(t) [0 1 0 0
        −w^2∗(1−3∗cos(w∗t)^2) 0 (3/2)∗w^2∗sin(2∗w∗t) 0
        0 0 0 1
        (3/2)∗w^2∗sin(2∗w∗t) 0 −w^2∗(1−3∗sin(w∗t)^2) 0];

Bbar = [0 0
        1 0
        0 0
        0 1];

thetai0 = @(i) (i − 1) ∗ pi/6;
Xi = @(t,i) RE ∗ cos(wE∗t + thetai0(i));
Yi = @(t,i) RE ∗ sin(wE∗t + thetai0(i));
thetai = @(t,i) atan2(Yi(t,i),Xi(t,i));
Xidot = @(t,i) −wE ∗ RE ∗ sin(wE∗t + thetai0(i));
Yidot = @(t,i) wE ∗ RE ∗ cos(wE∗t + thetai0(i));

delta_Xnom = @(t,i) r0∗cos(w∗t) − Xi(t,i);
delta_Ynom = @(t,i) r0∗sin(w∗t) − Yi(t,i);
delta_Xdotnom = @(t,i) −w∗r0∗sin(w∗t) − Xidot(t,i);
delta_Ydotnom = @(t,i) w∗r0∗cos(w∗t) − Yidot(t,i);
Anom = @(t,i) delta_Xnom(t,i)∗delta_Xdotnom(t,i)...
    + delta_Ynom(t,i)∗delta_Ydotnom(t,i);
rhonom = @(t,i) sqrt(delta_Xnom(t,i)^2 + delta_Ynom(t,i)^2);

Cbar = @(t,i) [delta_Xnom(t,i)./rhonom(t,i), 0, delta_Ynom(t,i)./rhonom(t,i)...
    , 0; % row 1
(rhonom(t,i)∗delta_Xdotnom(t,i) − Anom(t,i)∗(delta_Xnom(t,i)/rhonom(t,i)))...
/ rhonom(t,i)^2,... row 2 col 1
delta_Xnom(t,i)/rhonom(t,i),... row 2 col 2
```

```
(rhonom(t,i)*delta_Ydotnom(t,i) - Anom(t,i)*(delta_Ynom(t,i)/rhonom(t,i)))...
/ rhonom(t,i)^2,... row 2 col 3
delta_Ynom(t,i)/rhonom(t,i); % row 2 col 4
-delta_Ynom(t,i)/rhonom(t,i)^2, 0, delta_Xnom(t,i)/rhonom(t,i)^2, 0]; % row 3

Dbar = zeros(3,2);

%% CT -> DT and Simulate Perturbation Dynamics
% Initialize
delta_xk = zeros(4, N);
delta_xk(:,1) = delta_x0;
delta_yk = cell(N,1);
Fk = zeros(4,4,N);
Hk = cell(N,1);
yk = cell(N,1);

for k = 1:N-1

    % Build H at time t_k
    num_vis = station_vis{k+1};
    H = zeros(3*length(num_vis),4);
    for i = 1:length(num_vis)
        H(3*i-3+(1:3),:) = Cbar(tspan(k+1),num_vis(i));
    end
    Hk{k+1} = H;

    % Propagate perturbation to next step (except at the last time)
    if k < N
        Fk(:,:,k) = eye(4) + delta_t.*Abar(tspan(k));
        G = delta_t.*Bbar;
        delta_xk(:,k+1) = Fk(:,:,k)*delta_xk(:,k);
    end

    % Linearized measurement at time t_k
    delta_yk{k+1} = Hk{k+1}*delta_xk(:,k+1);
    yk{k+1} = y_nom{k+1} + delta_yk{k+1};
end

% Reconstruct the linearized full state
state_lin = x_nom + delta_xk.';

% Concatanate Outputs/Stations
L_outputs = [yk,station_vis];

%% Load in Data Logs and Define Inputs
% Load in Data Logs and Q/R
```

```
data = load('orbitdeterm_finalproj_KFdata.mat');
Qtrue = data.Qtrue;
R = data.Rtrue;
tvec_datalog = data.tvec;
ydatalog = data.ydata';
ydatalog(1) = cell(1,1);

% Modify ydatalog for plotting fxn
ydatalog_mod = cell(N,1);
station_vis_datalog = cell(N,1);
for k = 1:N
    current_y = cell2mat(ydatalog(k));
    if isempty(current_y)
        continue
    else
        station_vis_datalog{k} = current_y(end,:)';
        ydatalog_mod{k} = current_y(1:end-1,:); % take the top 3 rows
        ydatalog_mod{k} = ydatalog_mod{k}(:); % column-wise vectorization
    end
end

% Inputs
u_nom = zeros(2,length(tspan));
u = zeros(2,length(tspan));

%% Tuning Knobs
Q_LKF = 1*Qtrue;
Q_EKF = 1*Qtrue;
Q_UKF = 1.025*Qtrue;
alpha = 0.05;
beta = 2;
kappa = 0;

%% Noise and Covariance
% Process Noise Matrix
Gamma = Bbar; % w1,2 has same mapping as u1,2
Omegabar = delta_t.*Gamma;

% Generate Noisy Measurements;
y_pert_noise = cell(N,1);
for k = 1:N-1
    K = length(y_pert{k+1});
    Sv = chol(kron(eye(K/3),R),'lower');
    qk = randn(K,1);
    y_pert_noise{k+1} = y_pert{k+1} + Sv*qk;
end
```

```
noisy_ouputs = [y_pert_noise station_vis];

% ode45 call - pert w/ noise
x_true_pert = pert_var0;
x_pert_noisy = zeros(N,4);
x_pert_noisy(1,:) = pert_var0';
for k = 1:N-1
    current_tspan = [tspan(k) tspan(k+1)];
    [~,x_pert_k] = ode45(@(current_tspan,x_true_pert)...
        OrbitEOM(current_tspan,x_true_pert,mu),current_tspan,x_true_pert,options);
    xtrue_kp1 = x_pert_k(end,:);
    w_k = chol(Qtrue,"lower")*randn(2,1);
    x_true_pert = xtrue_kp1' + Omegabar*w_k;
    x_pert_noisy(k+1,:) = x_true_pert;
end

% Initialize Covariance
Pp0 = diag([2,2e-3,2,2e-3]);

% %% LKF
% [x_LKF,y_LKF,P_LKF,innov_LKF,delta_x_LKF,Sv_LKF] = LKF...
% (Fk,G,Hk,Q_LKF,R,Omegabar,delta_x0,Pp0,x_nom,u_nom,u,y_nom,y_pert_noise);
%
% LKF_outputs = [y_LKF station_vis];
% LKF_state_err = x_LKF-x_pert;
%
% %% EKF
% [x_EKF,P_EKF,y_EKF,innov_EKF,Sv_EKF] = EKF(Q_EKF,R,y_pert_noise,t,mu,RE,wE,nom_var0,Pp0,station_vis);
%
% EKF_outputs = [y_EKF station_vis];
% EKF_state_err = x_EKF'-x_pert;

%% UKF
[x_UKF,P_UKF,y_UKF,innov_UKF,Sv_UKF] = UKF(t,mu,RE,wE,nom_var0,Pp0,Q_UKF,...
    R,Omegabar,alpha,beta,kappa,ydatalog_mod,station_vis_datalog);

UKF_outputs = [y_UKF station_vis];
UKF_state_err = x_UKF'-x_pert;

%% Plots
% Dynamics Labels
Full_Dynamics_Labels = {'$X$ [km]','$\dot{X}$ [km/s]','$Y$ [km]',...
    '$\dot{Y}$ [km/s]'};
Perturbation_Dynamics_Labels = {'$\delta X$ [km]',...
    '$\delta\dot{X}$ [km/s]','$\delta Y$ [km]','$\delta\dot{Y}$ [km/s]'};
```

```
Error_Dynamics_Labels = {'$X$ Error [km]','$\dot{X}$ Error [km/s]',...
    '$Y$ Error [km]','$\dot{Y}$ Error[km/s]'};

% NL System
Plot_Dynamics(t,x_pert,Full_Dynamics_Labels,'Nonlinear Dynamics')
Plot_Outputs(t,NL_outputs,'Nonlinear Model Outputs')

% Linearized System
Plot_Dynamics(t,delta_xk',Perturbation_Dynamics_Labels,...
    'Linearized Perturbation Dynamics')
Plot_Dynamics(t,state_lin,Full_Dynamics_Labels,'Linearized Full Dynamics')
Plot_Outputs(t,L_outputs,'Linearized Model Outputs')

% Noisy Measurements
Plot_Outputs(t,noisy_ouputs,'Noisy Measurement Model Outputs')

% % LKF Results
% Plot_Outputs(t,LKF_outputs,'LKF Outputs')
%
% Plot_KFState_Results(t,x_pert,x_LKF,LKF_state_err,P_LKF,'LKF State Estimate Results',...
% 'LKF State Estimate Error',Full_Dynamics_Labels,Error_Dynamics_Labels)
%
% % EKF Results
% Plot_Outputs(t,EKF_outputs,'EKF Outputs')
%
% Plot_KFState_Results(t,x_pert,x_EKF',EKF_state_err,P_EKF,'EKF State Estimate Results',...
% 'EKF State Estimate Error',Full_Dynamics_Labels,Error_Dynamics_Labels)

% UKF Results
Plot_Outputs(t,UKF_outputs,'UKF Outputs')

Plot_KFState_Results(t,x_pert,x_UKF',UKF_state_err,P_UKF,'UKF State Estimate Results',...
    'UKF State Estimate Error',Full_Dynamics_Labels,Error_Dynamics_Labels)

% Extract Variances
X_var = squeeze(P_UKF(1,1,:));
X_dot_var = squeeze(P_UKF(2,2,:));
Y_var = squeeze(P_UKF(3,3,:));
Y_dot_var = squeeze(P_UKF(4,4,:));

% Calculate Sigma Bound Magnitude
X_sigma(:,1) = sqrt(X_var);
X_dot_sigma(:,1) = sqrt(X_dot_var);
Y_sigma(:,1) = sqrt(Y_var);
Y_dot_sigma(:,1) = sqrt(Y_dot_var);
```

```
figure();
subplot(411)
plot(t,x_UKF(1,:),'b','LineWidth',2)
hold on; grid on;grid minor
plot(t,x_UKF(1,:)+2.*X_sigma','--r','LineWidth',2)
plot(t,x_UKF(1,:)-2.*X_sigma','--r','LineWidth',2)
ylabel('$X$ [km]','Interpreter','latex')
subplot(412)
plot(t,x_UKF(2,:),'b','LineWidth',2)
hold on; grid on;grid minor
plot(t,x_UKF(2,:)+2.*X_dot_sigma','--r','LineWidth',2)
plot(t,x_UKF(2,:)-2.*X_dot_sigma','--r','LineWidth',2)
ylabel('$\dot{X}$ [km/s]','Interpreter','latex')
subplot(413)
plot(t,x_UKF(3,:),'b','LineWidth',2)
hold on; grid on;grid minor
plot(t,x_UKF(3,:)+2.*Y_sigma','--r','LineWidth',2)
plot(t,x_UKF(3,:)-2.*Y_sigma','--r','LineWidth',2)
ylabel('$Y$ [km]','Interpreter','latex')
subplot(414)
plot(t,x_UKF(4,:),'b','LineWidth',2)
hold on; grid on;grid minor
plot(t,x_UKF(4,:)+2.*Y_dot_sigma','--r','LineWidth',2)
plot(t,x_UKF(4,:)-2.*Y_dot_sigma','--r','LineWidth',2)
ylabel('$\dot{Y}$ [km/s]','Interpreter','latex')
sgtitle('UKF State and $\pm 2\sigma$ bounds','Interpreter','latex')
```

6.2 Code: Measured Outputs

```
function [output_var,station_vis] = MeasuredOutput(t,state,RE,wE,NaN_bool)
    % Goal: Output measured outputs (rho, rho_dot, phi) for each visible station
    % t: time vector
    % state: satellite state vector
    % RE: radius of Earth
    % wE: rotational rate of Earth
    % NaN_bool: if true, hides non visible measurements with NaN

    % Extract state info
    N = length(t);
    x_t = state(:,1); % km
    x_dot_t = state(:,2); % km/s
    y_t = state(:,3); % km
    y_dot_t = state(:,4); % km/s

    % Get positions and viewing angle for each station
    for i = 1:12
        theta_i0 = (i-1)*pi/6; % rad
```

```
        Xs_it = RE*cos(wE.*t+theta_i0); % km
        Ys_it = RE*sin(wE.*t+theta_i0); % km
        Xsdot_it = -RE*wE*sin(wE.*t+theta_i0); % km/s
        Ysdot_it = RE*wE*cos(wE.*t+theta_i0); % km/s
        theta_it = atan2(Ys_it,Xs_it); % rad

        % Determine visibility and outputs
        phi_it(:,i) = atan2(y_t-Ys_it,x_t-Xs_it);
        rho_it(:,i) = sqrt((x_t-Xs_it).^2 + (y_t-Ys_it).^2);
        rhodot_it(:,i) = (((x_t - Xs_it).*(x_dot_t - Xsdot_it)) +...
            ((y_t-Ys_it).*(y_dot_t-Ysdot_it)))./rho_it(:,i);
        station_vis(:,i) = i.*ones(length(rho_it(:,i)),1);
        if NaN_bool == true
            for k = 1:N
                if abs(wrapToPi(phi_it(k,i) - theta_it(k))) < pi/2
                    phi_it(k,i) = phi_it(k,i);
                    rho_it(k,i) = rho_it(k,i);
                    rhodot_it(k,i) = rhodot_it(k,i);
                    station_vis(k,i) = station_vis(k,i);
                else
                    phi_it(k,i) = NaN;
                    rho_it(k,i) = NaN;
                    rhodot_it(k,i) = NaN;
                    station_vis(k,i) = NaN;
                end
            end
        else
            phi_it(:,i) = phi_it(:,i);
            rho_it(:,i) = rho_it(:,i);
            rhodot_it(:,i) = rhodot_it(:,i);
            station_vis(:,i) = station_vis(:,i);
        end
        yi(3*i-3 + (1:3),:) = [rho_it(:,i)';rhodot_it(:,i)';phi_it(:,i)'];
    end
    station_vis = station_vis';

    % Allows for KF alogrithms to pull one output at a time
    if width(yi) > 1
        station_vis(:,1) = NaN;
        yi(:,1) = NaN;
    end

    output_var = yi;

end
```

6.3 Code: Equations of Motion

```
function [var_dot] = OrbitEOM(~,var,mu)
    % Goal: Output ODEs for ode45 where var is a 4x1 state vector (2D pos and vel)

    % Extract state variables
    x = var(1);
    u = var(2);
    y = var(3);
    v = var(4);

    % Calculate radius
    r = sqrt(x^2 + y^2);

    % Assign t.r.o.c variables
    x_dot = u;
    y_dot = v;
    u_dot = (-mu*x)/r^3;
    v_dot = (-mu*y)/r^3;

    % Final state derivative
    var_dot = [x_dot;u_dot;y_dot;v_dot];
end
```

6.4 Code: LKF

```
function [x_full,y_full,Ppkp1,innov,delta_xpkp1,Skp1] = LKF...
    (F,G,H,Q,R,Omega,delta_x0,Pp0,x_nom,u_nom,u,y_nom,y_noisy)

    % Preallocate/initialize
    N = size(F,3);
    delta_xpkp1 = zeros(4,N);
    delta_xpkp1(:,1) = delta_x0;
    innov = cell(N,1);
    y_full = cell(N,1);
    Ppkp1(:,:,1) = Pp0;
    delta_uk = u - u_nom;
    Skp1 = cell(N,1);
    Skp1{1} = zeros(length(y_noisy{1}), length(y_noisy{1}));

    % LKF Algorithm
    for k = 1:N-1
        % Time update/prediction step
        delta_xmkp1 = F(:,:,k)*delta_xpkp1(:,k) + G*delta_uk(:,k);
        Pmkp1 = F(:,:,k)*Ppkp1(:,:,k)*F(:,:,k)' + Omega*Q*Omega';

        % Measurement update/correction step
```

```
        K = height(H{k+1})/3;
        Rk = kron(eye(K),R);
        Skp1{k+1} = (H{k+1}*Pmkp1*H{k+1}'+Rk);
        Kkp1 = Pmkp1*H{k+1}'/Skp1{k+1};
        delta_ykp1 = y_noisy{k+1} - y_nom{k+1};
        Ppkp1(:,:,k+1) = (eye(4)-Kkp1*H{k+1})*Pmkp1;
        delta_xpkp1(:,k+1) = delta_xmkp1+Kkp1*(delta_ykp1-H{k+1}*delta_xmkp1);

        % Innovation
        innov{k+1} = delta_ykp1-H{k+1}*delta_xmkp1;

        % Add Outputs
        y_full{k+1} = y_nom{k+1} + H{k+1}*delta_xmkp1;
    end

    % Add to nominal state
    x_full = x_nom + delta_xpkp1';
end
```

6.5 Code: EKF

```
%Written by Brady Sivey

%This function implements the Extended Kalman Filter as learned in ASEN
%5044
function [xhatp, Pp, yhat,innov, Sv] = EKF(Q, R, ydata, tvec, mu, rE, wE, xhat0, P0, station_vis)
%inputs
%1. Q - Process noise covariance
%2. R - Measurement Noise Covariance
%3. ydata - Measurements
%4. tvec - Time vector
%5. mu - Gravitational Parameter
%6. rE - Radius of Earth
%7. wE - Earth rotation rate
%8. xhat0 - Initial state estimate
%9. P0 - Initial Covariance
%10. station_vis - Vector of visible station numbers

%outputs
%1. xhatp - state estimates
%2. Pp - Covariance
%3. yhat - Predicted Measurements

% ode45 options
options = odeset('RelTol',1e-6,'AbsTol',1e-9);

N = length(tvec);
```

```
%preallocating
xhatp = zeros(4, N);
Pp = zeros(4, 4, N);
yhat = cell(N,1);
Sv = cell(N,1);
innov = cell(N,1);

%step 1
%initializing
xhatp(:, 1) = xhat0;
Pp(:, :, 1) = P0;

for k = 1:N−1
    %step 2
    tk = tvec(k);
    tkn = tvec(k+1);
    dt = tkn − tk;

    %step 3
    x0 = xhatp(:, k);
    %deterministic nonlinear DT function eval
    [˜, xtraj] = ode45(@(t,x) OrbitEOM(t,x,mu), [tk tkn], x0, options);
    xhatm = xtraj(end,:).';

    vis_idx = station_vis{k+1};
    [Hk, Ac, Gamc] = EKFJacobians(xhatm, tkn, rE, wE, mu, vis_idx);

    Fk = eye(4) + dt ∗ Ac;
    Omegak = dt ∗ Gamc;

    %approx predicted covariance
    Pm = Fk∗Pp(:,:,k)∗Fk.' + Omegak∗Q∗Omegak.';

    %step 4
    %deterministic nonlinear function eval
    [yfa_unfiltered, ˜] = MeasuredOutput(tkn, xhatm', rE, wE, false);;

    m = length(vis_idx);
    yfa = [];
    for j = 1:m
        yfa(3∗j−3+(1:3),:) = yfa_unfiltered(3∗vis_idx(j)−3+(1:3),:);
    end
    if isempty(yfa)
        y_hat{k+1} = [];
        xhatp(:,k+1) = xhatm;
```

```
        Pp(:,:,k+1) = Pm;
    else
        yhat{k+1} = yfa;

        %innovation
        innov{k+1} = ydata{k+1} - yhat{k+1};

        Rk = kron(eye(m), R);

        %kalman gain
        Sv{k+1} = (Hk * Pm * Hk.' + Rk);
        K = Pm * Hk.' / Sv{k+1};

        %updated total state estimate
        xhatp(:,k+1) = xhatm + K*innov{k+1};

        %updated covariance approximation
        Pp(:,:,k+1) = (eye(4) - K*Hk) * Pm;
    end
end

%Written by Brady Sivey

%The purpose of this function is to compute the necessary Jacobians
%to be used in the EKF.
function [Hk, Ac, Gamc] = EKFJacobians(xhatm, t, rE, wE, mu, stationvisindex)
%inputs
%1. xhatm - predicted state from dynamics
%2. t - time
%3. rE - Radius of Earth
%4. wE - Earth rotation rate
%5. mu - Gravitational Parameter
%6. stationvisindex - Vector of visible station numbers

%outputs
%1. Hk - Linearized measurement matrix for each station
%2. Ac - df/dx Jacobian
%3. Gamc - df/dw Jacobian

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

N = length(stationvisindex);
```

```
Hk = zeros(3*N, 4);

%for loop to construct Hk
for i = 1:N
s = stationvisindex(i);

%givens
thetai0 = (s - 1) * pi/6;
Xs = rE*cos(wE*t + thetai0);
Ys = rE*sin(wE*t + thetai0);
Xsdot = -rE*wE*sin(wE*t + thetai0);
Ysdot = rE*wE*cos(wE*t + thetai0);

%These equations come from my derivations
dX = X - Xs;
dXdot = Xdot - Xsdot;
dY = Y - Ys;
dYdot = Ydot - Ysdot;

rho = sqrt(dX^2 + dY^2);
A = dX*dXdot + dY*dYdot;

% 3x4 H matrix
H21 = (rho*dXdot - A*(dX/rho))/(rho^2);
H23 = (rho*dYdot - A*(dY/rho))/(rho^2);
Hi = [dX/rho 0 dY/rho 0;
     H21 dX/rho H23 dY/rho;
     -dY/(rho^2) 0 dX/(rho^2) 0];

idx = 3*(i-1) + (1:3);
Hk(idx,:) = Hi;

end

%constructing Ac and Gamc matrices

%These come from my derivations
Ac21 = -mu * (Y^2 - 2*X^2)/(X^2 + Y^2)^(5/2);
Ac23 = mu * (3*X*Y)/(X^2 + Y^2)^(5/2);
Ac41 = mu * (3*X*Y)/(X^2 + Y^2)^(5/2);
Ac43 = -mu * (X^2 - 2*Y^2)/(X^2 + Y^2)^(5/2);

Ac = [0 1 0 0;
     Ac21 0 Ac23 0
     0 0 0 1;
     Ac41 0 Ac43 0];
```

```
Gamc = [0 0;
    1 0;
    0 0;
    0 1];

end
```

6.6 Code: Monte Carlo Simulations/Truth Model Testing

```
clc; clear all; close all;

% Constants
mu = 398600; % km^3/s^2
r0 = 6678; % km
Ydot0 = r0 * sqrt(mu/r0^3); % km/s
RE = 6378; % km
wE = (2*pi)/86400; % rad/s
w = sqrt(mu/r0^3);

% ICs
nom_var0 = [r0 0 0 Ydot0]';
delta_x0 = [0 0.075 0 -0.021]';
pert_var0 = nom_var0 + delta_x0;

% Simulation time
delta_t = 10; %s
tspan = 0:delta_t:14000; % s
N = length(tspan);

% Initialize Covariance
Pp0 = diag([2,2e-3,2,2e-3]);

% ode45 options
options = odeset('RelTol',1e-6,'AbsTol',1e-9);

% Loading Qtrue and Rtrue
data = load('orbitdeterm_finalproj_KFdata.mat');
Qtrue = data.Qtrue;
R = data.Rtrue;
tvec_datalog = data.tvec;
ydatalog = data.ydata';
ydatalog(1) = cell(1,1);

% Inputs
u_nom = zeros(2,length(tspan));
u = zeros(2,length(tspan));
```

```
% Getting nominal trajectory and measurements
[~,x_nom] = ode45(@(tspan,var0) OrbitEOM(tspan,var0,mu),tspan,nom_var0,options);
[output_var_nom,~] = MeasuredOutput(tspan',x_nom,RE,wE,false);

%% KF Tuning Knob
% LKF
Q_LKF = 80*Qtrue;

% EKF
Q_EKF = 80*Qtrue;

% UKF
alpha = 0.05;% 0.05
beta = 2;
kappa = 0;
Q_UKF = 1.025*Qtrue; % 1.03


%% CT Dynamics
Abar = @(t) [0 1 0 0
        -w^2*(1-3*cos(w*t)^2) 0 (3/2)*w^2*sin(2*w*t) 0
        0 0 0 1
        (3/2)*w^2*sin(2*w*t) 0 -w^2*(1-3*sin(w*t)^2) 0];

Bbar = [0 0
        1 0
        0 0
        0 1];

thetai0 = @(i) (i - 1) * pi/6;
Xi = @(t,i) RE * cos(wE*t + thetai0(i));
Yi = @(t,i) RE * sin(wE*t + thetai0(i));
thetai = @(t,i) atan2(Yi(t,i),Xi(t,i));
Xidot = @(t,i) -wE * RE * sin(wE*t + thetai0(i));
Yidot = @(t,i) wE * RE * cos(wE*t + thetai0(i));

delta_Xnom = @(t,i) r0*cos(w*t) - Xi(t,i);
delta_Ynom = @(t,i) r0*sin(w*t) - Yi(t,i);
delta_Xdotnom = @(t,i) -w*r0*sin(w*t) - Xidot(t,i);
delta_Ydotnom = @(t,i) w*r0*cos(w*t) - Yidot(t,i);
Anom = @(t,i) delta_Xnom(t,i)*delta_Xdotnom(t,i)...
    + delta_Ynom(t,i)*delta_Ydotnom(t,i);
rhonom = @(t,i) sqrt(delta_Xnom(t,i)^2 + delta_Ynom(t,i)^2);

Cbar = @(t,i) [delta_Xnom(t,i)./rhonom(t,i), 0, delta_Ynom(t,i)./rhonom(t,i)...
```

```
    , 0; % row 1
(rhonom(t,i)*delta_Xdotnom(t,i) - Anom(t,i)*(delta_Xnom(t,i)/rhonom(t,i)))...
/ rhonom(t,i)^2,... row 2 col 1
delta_Xnom(t,i)/rhonom(t,i),... row 2 col 2
(rhonom(t,i)*delta_Ydotnom(t,i) - Anom(t,i)*(delta_Ynom(t,i)/rhonom(t,i)))...
/ rhonom(t,i)^2,... row 2 col 3
delta_Ynom(t,i)/rhonom(t,i); % row 2 col 4
-delta_Ynom(t,i)/rhonom(t,i)^2, 0, delta_Xnom(t,i)/rhonom(t,i)^2, 0]; % row 3

Dbar = zeros(3,2);

Gamma = Bbar; % w1,2 has same mapping as u1,2
Omegabar = delta_t.*Gamma;

% Compute Fk once outside MC loop
Fk = zeros(4,4,N);
for k = 1:N-1
    Fk(:,:,k) = eye(4) + delta_t.*Abar(tspan(k));
end
G = delta_t.*Bbar;

%% MC
num_sims = 50;
epsilon_x = zeros(N,num_sims);
epsilon_y = zeros(N,num_sims);
for sim_num = 1:num_sims
    %% Simulating nonlinear EOMs with perturbed initial condition and process noise
    % ode45 call - pert w/ noise
    x_true_pert = pert_var0;
    x_pert_noisy = zeros(N,4);
    x_pert_noisy(1,:) = x_true_pert';
    e_x_EKF = zeros(N,4);
    for k = 1:N-1
        % Propagate perturbed true state with noise using ode45
        current_tspan = [tspan(k) tspan(k+1)];
        [~,x_pert_k] = ode45(@(current_tspan,pert_var0)...
            OrbitEOM(current_tspan,pert_var0,mu),current_tspan,x_true_pert,options);
        xtrue_kp1 = x_pert_k(end,:);
        w_k = chol(Qtrue,"lower")*randn(2,1);
        x_true_pert = xtrue_kp1' + Omegabar*w_k;
        x_pert_noisy(k+1,:) = x_true_pert;
    end

    %% Simulating measurements for noisy NL simulation, adding measurment noise
    % Also, building H matrices at each k time step
    [station_meas,station_vis] = MeasuredOutput(tspan',x_pert_noisy,RE,wE,true);
```

```
% Convert into cell array − filter nominal measurements to match NL idxs
y_nom = cell(N,1);
y_pert_noisy = cell(N,1);
station_vis_cell = cell(N,1);

delta_xk = zeros(4, N);
delta_xk(:,1) = delta_x0;
Hk = cell(N,1);
p(1) = 0;
for k = 2:N
    y_pert = station_meas(~any(isnan(station_meas(:,k)),2), k);
    y_nom{k} = output_var_nom(~any(isnan(station_meas(:,k)),2), k);
    K = length(y_pert);
    Sv = chol(kron(eye(K/3),R),'lower');
    qk = randn(K,1);
    y_pert_noisy{k} = y_pert + Sv*qk;
    station_vis_cell{k} = station_vis(~any(isnan(station_vis(:,k)),2), k);

    % Build H at time t_k
    num_vis = station_vis_cell{k};
    H = zeros(3*length(num_vis),4);
    for i = 1:length(num_vis)
        H(3*i−3+(1:3),:) = Cbar(tspan(k),num_vis(i));
    end
    Hk{k} = H;

    % p keeps track of the number of available measurements at step k
    p(k) = size(H,1);
end

% % Run LKF and calculate NEES and NIS
% delta_x0_KF = zeros(4,1);
% [x_LKF,y_LKF,Ppkp1_LKF,innov_LKF,delta_x_LKF,Skp1] = LKF...
% (Fk,G,Hk,Q_LKF,R,Omegabar,delta_x0_KF,Pp0,x_nom,u_nom,u,y_nom,y_pert_noisy);
% % e_y_LKF = cell(N,1);
% for k=2:N
% % calculate epsilon_x for each time step k
% e_x_LKF(k,:) = x_pert_noisy(k,:) − x_LKF(k,:);
% epsilon_x_LKF(k,sim_num) = e_x_LKF(k,:) * (Ppkp1_LKF(:,:,k) \ e_x_LKF(k,:)');
% % calculate epsilon_y for each time step k
% epsilon_y_LKF(k,sim_num) = innov_LKF{k}' * (Skp1{k} \ innov_LKF{k});
% end
%
% % Run EKF and calculate NEES and NIS
% [x_EKF, P_EKF, y_EKF, innov_EKF, Sv_EKF] = EKF...
% (Q_EKF, R, y_pert_noisy, tspan', mu, RE, wE, nom_var0, Pp0, station_vis_cell);
```

```
    % e_y_EKF = cell(N,1);
    % for k=2:N
    % % calculate epsilon_x for each time step k
    % e_x_EKF(k,:) = x_pert_noisy(k,:) - x_EKF(:,k)';
    % epsilon_x_EKF(k,sim_num) = e_x_EKF(k,:) * (P_EKF(:,:,k) \ e_x_EKF(k,:)');
    % % calculate epsilon_y for each time step k
    % if isempty(innov_EKF{k})
    % epsilon_y_EKF(k,sim_num) = NaN;
    % else
    % epsilon_y_EKF(k,sim_num) = innov_EKF{k}' * (Sv_EKF{k} \ innov_EKF{k});
    % end
    % end

    % Run UKF and calculate NEES and NIS
    [x_UKF,P_UKF,y_UKF,innov_UKF,Sv_UKF] = UKF(tvec_datalog,mu,RE,wE,nom_var0,Pp0,Q_UKF,...
        R,Omegabar,alpha,beta,kappa,y_pert_noisy,station_vis_cell);
    for k=2:N
        % calculate epsilon_x for each time step k
        e_x_UKF(k,:) = x_pert_noisy(k,:) - x_UKF(:,k)';
        epsilon_x_UKF(k,sim_num) = e_x_UKF(k,:) * (P_UKF(:,:,k) \ e_x_UKF(k,:)');
        % calculate epsilon_y for each time step k
        if isempty(innov_UKF{k})
            epsilon_y_UKF(k,sim_num) = NaN;
        else
            epsilon_y_UKF(k,sim_num) = innov_UKF{k}' * (Sv_UKF{k} \ innov_UKF{k});
        end
    end
end
% Calculate NEES and NIS
% epsilonbar_x_LKF = (1/num_sims) .* sum(epsilon_x_LKF,2);
% epsilonbar_y_LKF = (1/num_sims) .* sum(epsilon_y_LKF,2);

% Getting upper and lower bounds for NEES and NIS
alpha = 0.05;
r1_NEES = chi2inv(alpha/2,num_sims*4)./num_sims;
r2_NEES = chi2inv(1-alpha/2,num_sims*4)./num_sims;
r1_NIS = chi2inv(alpha/2,num_sims*p)./num_sims;
r2_NIS = chi2inv(1-alpha/2,num_sims*p)./num_sims;

%% Plots of NEES and NIS for LKF and EKF
% figure;
% scatter(1:N, epsilonbar_x_LKF, 16, 'filled'); grid on; hold on;
% xlabel('Time Step k');
% ylabel('$\bar{\epsilon}_x$', 'Interpreter', 'latex');
% title('LKF NEES Test Results');
% xlim([1,N]);
```

```
% yline(r1_NEES,'--',"r1","Color","red");
% yline(r2_NEES,'--',"r2","Color","red");
%
% figure;
% scatter(1:N, epsilonbar_y_LKF, 16, 'filled'); grid on; hold on;
% xlabel('Time Step k');
% ylabel('$\bar{\epsilon}_y$', 'Interpreter', 'latex');
% title('LKF NIS Test Results');
% xlim([1,N]);
% plot(1:N,r1_NIS,'--',"Color","red");
% plot(1:N,r2_NIS,'--',"Color","red");
%
% epsilonbar_x_EKF = (1/num_sims) .* sum(epsilon_x_EKF,2);
% epsilonbar_y_EKF = (1/num_sims) .* sum(epsilon_y_EKF,2);
%
% figure;
% scatter(1:N, epsilonbar_x_EKF, 16, 'filled'); grid on; hold on;
% xlabel('Time Step k');
% ylabel('$\bar{\epsilon}_x$', 'Interpreter', 'latex');
% title('EKF NEES Test Results');
% xlim([1,N]);
% yline(r1_NEES,'--',"r1","Color","red");
% yline(r2_NEES,'--',"r2","Color","red");
%
% figure;
% scatter(1:N, epsilonbar_y_EKF, 16, 'filled'); grid on; hold on;
% xlabel('Time Step k');
% ylabel('$\bar{\epsilon}_y$', 'Interpreter', 'latex');
% title('EKF NIS Test Results');
% xlim([1,N]);
% plot(1:N,r1_NIS,'--',"Color","red");
% plot(1:N,r2_NIS,'--',"Color","red");

epsilonbar_x_UKF = (1/num_sims) .* sum(epsilon_x_UKF,2);
epsilonbar_y_UKF = mean(epsilon_y_UKF, 2, 'omitnan');

figure;
scatter(1:N, epsilonbar_x_UKF, 16, 'filled'); grid on; hold on;
xlabel('Time Step k');
ylabel('$\bar{\epsilon}_x$', 'Interpreter', 'latex');
title('UKF NEES Test Results');
xlim([1,N]);
yline(r1_NEES,'--',"r1","Color","red");
yline(r2_NEES,'--',"r2","Color","red");

figure;
```

```
scatter(1:N, epsilonbar_y_UKF, 16, 'filled'); grid on; hold on;
xlabel('Time Step k');
ylabel('$\bar{\epsilon}_y$', 'Interpreter', 'latex');
title('UKF NIS Test Results');
xlim([1,N]);
plot(1:N,r1_NIS,'--',"Color","red");
plot(1:N,r2_NIS,'--',"Color","red");


UKF_NEES_counter = 0;
UKF_NIS_counter = 0;
for i = 1:length(epsilonbar_y_UKF)
    if epsilonbar_x_UKF(i) > r1_NEES && epsilonbar_x_UKF(i) < r2_NEES
        UKF_NEES_counter = UKF_NEES_counter + 1;
    end
    if epsilonbar_y_UKF(i) > r1_NIS(i) && epsilonbar_y_UKF(i) < r2_NIS(i)
        UKF_NIS_counter = UKF_NIS_counter + 1;
    end
end

% NEES count (all time steps are valid for state)
UKF_NEES_counter = sum( epsilonbar_x_UKF > r1_NEES & epsilonbar_x_UKF < r2_NEES );
fprintf('There are %d/%d NEES points within the bounds\n', ...
        UKF_NEES_counter, N);

% NIS count: only where we actually have measurements (p > 0) and finite NIS
valid_NIS_idx = (p(:) > 0) & ~isnan(epsilonbar_y_UKF);

UKF_NIS_counter = sum( epsilonbar_y_UKF(valid_NIS_idx) > r1_NIS(valid_NIS_idx)' & ...
                       epsilonbar_y_UKF(valid_NIS_idx) < r2_NIS(valid_NIS_idx)' );

num_valid_NIS = sum(valid_NIS_idx);
fprintf('There are %d/%d NIS points within the bounds (where measurements exist)\n', ...
        UKF_NIS_counter, num_valid_NIS);
```

6.7 Code: UKF

```
function [xpkp1,Ppkp1,ymkp1,innov,Pyykp1] = UKF...
    (t,mu,RE,wE,xp0,P0,Q,R,Omega,alpha,beta,kappa,y_data,station_vis)
    % Inputs:
    % t: time vector
    % mu: EArth's gravitational parameter
    % RE: radius of Earth
    % wE: rotational rate of Earth
    % xp0: initial state condition
    % P0: inital covariance matrix
    % Q: process noise covariance matrix
```

```
% R: measurement noise covariance matrix
% Omega: process noise mapping matrix
% alpha: tuning parameter between [1e-4,1]
% beta: tuning parameter(?), typically = 2
% kappa: tuning paramater(?), typically = 0
% y_data: noisy measurements (cell array)
% station_vis: station visibilities

% Outputs:
% xpkp1: state estimates
% Ppkp1: state covariance
% ymkp1: predicted measurements
% innov: innovation term
% Pyykp1: innovation covariance matrix

% ode45 options
options = odeset('RelTol',1e-6,'AbsTol',1e-9);

% Initialize/Preallocate
N = length(t);
xpkp1 = zeros(length(P0),N);
xpkp1(:,1) = xp0;
n = length(xp0);
Ppkp1 = zeros(length(P0),length(P0),N);
Ppkp1(:,:,1) = P0;
chik = zeros(n,2*n+1);
chikp1 = zeros(n,2*n+1);
wm = zeros(1,2*n+1);
wc = zeros(1,2*n+1);
ymkp1 = cell(N,1);
innov = cell(N,1);
Pyykp1 = cell(N,1);

% Calulate necessary constants
lambda = alpha^2*(n+kappa)-n;
wm(1:2*n) = 1/(2*(n+lambda));
wm(2*n+1) = lambda/(n+lambda);
wc(1:2*n) = wm(1:2*n);
wc(2*n+1) = wm(2*n+1)+1-alpha^2+beta;

% UKF Algorithm
for k = 1:N-1

    % Step 1: Dynamics Prediction
    % Generate 2n+1 Sigma Points for Dynamics Prediciton
    chik(:,2*n+1) = xpkp1(:,k); % mean in 9th position
```

```
Sk = chol(Ppkp1(:,:,k),'lower');
for i = 1:n
    chik(:,i) = xpkp1(:,k) + (sqrt(n+lambda))*Sk(:,i); % 1-4 of 2nd moment
end
for i = n+1:2*n
    chik(:,i) = xpkp1(:,k) - (sqrt(n+lambda))*Sk(:,i-4); % 5-8 of 2nd moment
end

% Propogate Sigma Points through NL Dynamics, get prior info
current_tspan = [t(k) t(k+1)];
for i = 1:2*n+1
    current_chi = chik(:,i);
    [~,chik_prop] = ode45(@(current_tspan,current_chi)...
        OrbitEOM(current_tspan,current_chi,mu),current_tspan,current_chi,options);
    chibarkp1(:,i) = chik_prop(end,:)';
end
xmkp1 = chibarkp1 * wm';
Pmkp1 = zeros(size(P0));
for i = 1:2*n+1
    Pmkp1 = wc(i)*(chibarkp1(:,i)-xmkp1)*(chibarkp1(:,i)-xmkp1)' + Pmkp1;
end
Pmkp1 = Pmkp1 + Omega*Q*Omega';

% Step 2: Measurement Update
% Generate 2n+1 Sigma Points for Measurement Update
chikp1(:,2*n+1) = xmkp1; % pred mean in 9th position
Sbarkp1 = chol(Pmkp1,'lower');
for i = 1:n
    chikp1(:,i) = xmkp1 + (sqrt(n+lambda))*Sbarkp1(:,i); % 1-4 of 2nd moment
end
for i = n+1:2*n
    chikp1(:,i) = xmkp1 - (sqrt(n+lambda))*Sbarkp1(:,i-4); % 5-8 of 2nd moment
end

% Propogate Sigma Points through NL Measurements, get prior info
for i = 1:2*n+1
    current_chikp1 = chikp1(:,i);
    [output_var,~] = MeasuredOutput...
        (t(k+1),current_chikp1',RE,wE,false);
    gammabar_unfiltered(:,i) = output_var;
end
vis_stations = station_vis{k+1};

% Filter Visible Stations
gammabar_kp1 = [];
for j = 1:length(vis_stations)
```

```
            gammabar_kp1(3*j-3+(1:3),:) = gammabar_unfiltered(3*vis_stations(j)-3+(1:3),:);
        end
        if isempty(gammabar_kp1)
            ymkp1{k+1} = [];
            xpkp1(:,k+1) = xmkp1;
            Ppkp1(:,:,k+1) = Pmkp1;
            innov{k+1} = [];
        else
            ymkp1{k+1} = gammabar_kp1 * wm';
            K = length(ymkp1{k+1});
            Pyykp1_ = zeros(size(kron(eye(K/3),R)));
            for i = 1:2*n+1
                Pyykp1_ = wc(i)*(gammabar_kp1(:,i)-ymkp1{k+1})...
                    *(gammabar_kp1(:,i)-ymkp1{k+1})' + Pyykp1_;
            end
            Pyykp1{k+1} = Pyykp1_ + kron(eye(K/3),R);

            % Get state-measurement cross covariance matrix
            Pxykp1 = zeros(n,size(kron(eye(K/3),R),1));
            for i = 1:2*n+1
                Pxykp1 = wc(i)*(chikp1(:,i)-xmkp1)...
                    *(gammabar_kp1(:,i)-ymkp1{k+1})' + Pxykp1;
            end

            % Estimate Kalman Gain
            Kkp1 = Pxykp1/Pyykp1{k+1};

            % Update State and Covariace
            innov{k+1} = y_data{k+1}-ymkp1{k+1};
            xpkp1(:,k+1) = xmkp1+Kkp1*(innov{k+1});
            Ppkp1(:,:,k+1) = Pmkp1-Kkp1*Pyykp1{k+1}*Kkp1';
        end
    end

end
```