

Final Project - Statistical orbit determination

ASEN 5044 - Statistical Estimation for Dynamical Systems (Fall 2025)
Prof. Khosro Ghobadi-Far

Jack Huston
Ryan Hyatt
Philippe Kruettli

December 12, 2025

Contents

1	Team member contributions	3
1.1	Ryan	3
1.2	Philippe	3
1.3	Jack	3
2	Part I - Deterministic System Analysis	4
2.1	Continuous-Time Dynamics	4
2.2	Discrete-Time Linearization	6
2.3	Dynamics Simulation	7
2.3.1	Station Visibility	7
2.3.2	Linearized DT simulation	7
2.3.3	Full nonlinear simulation	8
2.3.4	Results and Plots	9
2.3.5	DT Linearized Simulation	10
2.3.6	Nonlinear Simulation	11
2.3.7	Nonlinear Simulated Measurements vs. Time	12
2.3.8	Comparison	13
3	Part II - Stochastic Nonlinear Filtering	14
3.1	Stochastic Nonlinear Filter Validation via Monte Carlo Analysis	14
3.1.1	Ground Truth Simulation	14
3.1.2	Filter Consistency Testing	14
3.1.3	Filter Tuning Methodology	15
3.2	Linearized Kalman Filter (LKF)	16
3.2.1	Initialization	16
3.2.2	Time update/prediction step	16
3.2.3	Index Convention	16
3.2.4	Measurement update/correction step	16
3.2.5	Output	17
3.2.6	Results	17
3.3	Extended Kalman Filter (EKF)	22
3.3.1	Initialization and EKF setup	22
3.3.2	Time update (prediction)	22
3.3.3	Measurement update	23
3.3.4	Results	24
3.4	State trajectory estimation and comparison	28
4	Estimation Haiku - Advanced Question 13	31
5	Appendix	33
5.1	Appendix A: Equation Derivation	33
5.2	Appendix B: MATLAB Code	39

1 Team member contributions

1.1 Ryan

1. Linearized DT dynamics and measurement simulation
2. Part 3 linearized DT dynamics figure generation
3. Document writeup for part 3 linearized simulation
4. Implementation and tuning of linearized Kalman Filter (Part 4a)
5. Part 6 - Linearized Kalman filter implementation
6. Document writeup for part 4a, and part 6

1.2 Philippe

1. Nonlinear ode45 dynamics and measurement simulation
2. Part 3 non-linear dynamics figure generation
3. Document writeup for part 3 non-linear simulation
4. Non-linear simulation running as monte-carlo for parts 4 and 5
5. Plots of the NEES test statistics for 4.b and 5.b
6. Plots of the NIS test statistics for 4.c and 5.c
7. Part 6 comparison
8. Document writeup for NEES and NIS statistics for 4.b/c and 5.b/c, and part 6 comparison.

1.3 Jack

1. Part 1 Jacobian derivation for CT model
2. Part 2 Linearization and associated Matlab Script
3. Document writeup for parts 1 and 2
4. Implementation and tuning of Extended Kalman Filter (Part 5a)
5. Part 6 - Filter implementation
6. Document writeup for part 5a, and part 6

2 Part I - Deterministic System Analysis

2.1 Continuous-Time Dynamics

The spacecraft motion is modeled in an Earth-centered inertial frame with state vector

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} = \begin{bmatrix} X(t) \\ \dot{X}(t) \\ Y(t) \\ \dot{Y}(t) \end{bmatrix},$$

control input

$$u(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix},$$

and process disturbance

$$\tilde{w}(t) = \begin{bmatrix} \tilde{w}_1(t) \\ \tilde{w}_2(t) \end{bmatrix}.$$

The current orbital radius is

$$r(t) = \sqrt{X^2(t) + Y^2(t)} = \sqrt{x_1^2(t) + x_3^2(t)},$$

and the continuous-time nonlinear dynamics (including small control and disturbance accelerations) are

$$\begin{aligned} \dot{x}_1(t) &= \dot{X}(t) = x_2(t), \\ \dot{x}_2(t) &= \ddot{X}(t) = -\mu \frac{x_1(t)}{r^3(t)} + u_1(t) + \tilde{w}_1(t), \\ \dot{x}_3(t) &= \dot{Y}(t) = x_4(t), \\ \dot{x}_4(t) &= \ddot{Y}(t) = -\mu \frac{x_3(t)}{r^3(t)} + u_2(t) + \tilde{w}_2(t), \end{aligned}$$

with the standard gravitational parameter $\mu = 398600 \text{ km}^3/\text{s}^2$.

These equations can be written as

$$\dot{x}(t) = f(x(t)) + B u(t) + \Gamma \tilde{w}(t),$$

where $f \in \mathbb{R}^4$ collects the gravitational acceleration terms, and the input and disturbance Jacobians are

$$B = \Gamma = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4 \times 2}.$$

Dynamics Jacobian $A(x)$

The continuous-time dynamics Jacobian $A(x) = \partial f / \partial x$ (dimensions 4×4) is obtained by differentiating the components of $f(x)$ with respect to the state variables x_1, x_2, x_3 , and x_4 . Using $r = \sqrt{x_1^2 + x_3^2}$ we find

$$\begin{aligned} \frac{\partial f_1}{\partial x} &= [0 \quad 1 \quad 0 \quad 0], \\ \frac{\partial f_2}{\partial x_1} &= -\mu \frac{\partial}{\partial x_1} \left(\frac{x_1}{r^3} \right) = \mu \frac{2x_1^2 - x_3^2}{r^5}, \\ \frac{\partial f_2}{\partial x_3} &= -\mu \frac{\partial}{\partial x_3} \left(\frac{x_1}{r^3} \right) = 3\mu \frac{x_1 x_3}{r^5}, \\ \frac{\partial f_3}{\partial x} &= [0 \quad 0 \quad 0 \quad 1], \\ \frac{\partial f_4}{\partial x_1} &= -\mu \frac{\partial}{\partial x_1} \left(\frac{x_3}{r^3} \right) = 3\mu \frac{x_1 x_3}{r^5}, \\ \frac{\partial f_4}{\partial x_3} &= -\mu \frac{\partial}{\partial x_3} \left(\frac{x_3}{r^3} \right) = \mu \frac{2x_3^2 - x_1^2}{r^5}, \end{aligned}$$

while the remaining partial derivatives are zero. Collecting these terms gives

$$A(x) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \mu \frac{2x_1^2 - x_3^2}{r^5} & 0 & 3\mu \frac{x_1 x_3}{r^5} & 0 \\ 0 & 0 & 0 & 1 \\ 3\mu \frac{x_1 x_3}{r^5} & 0 & \mu \frac{2x_3^2 - x_1^2}{r^5} & 0 \end{bmatrix}, \quad A(x) \in \mathbb{R}^{4 \times 4}.$$

Measurement model and Jacobian $H^i(x, t)$

Each ground station i provides a 3-dimensional measurement

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

where ρ^i is the range, $\dot{\rho}^i$ the range rate, and ϕ^i the elevation angle from station i to the spacecraft. Using the station position $(X_s^i(t), Y_s^i(t))$ and velocity $(\dot{X}_s^i(t), \dot{Y}_s^i(t))$, we define relative position and velocity components

$$\begin{aligned} \Delta X_i &= X(t) - X_s^i(t), & \Delta Y_i &= Y(t) - Y_s^i(t), \\ \Delta \dot{X}_i &= \dot{X}(t) - \dot{X}_s^i(t), & \Delta \dot{Y}_i &= \dot{Y}(t) - \dot{Y}_s^i(t), \end{aligned}$$

and

$$\rho_i = \sqrt{\Delta X_i^2 + \Delta Y_i^2}, \quad a_i = \Delta X_i \Delta \dot{X}_i + \Delta Y_i \Delta \dot{Y}_i.$$

The nonlinear measurement functions are then

$$\begin{aligned} \rho^i(t) &= \rho_i, \\ \dot{\rho}^i(t) &= \frac{a_i}{\rho_i}, \\ \phi^i(t) &= \tan^{-1} \left(\frac{\Delta Y_i}{\Delta X_i} \right). \end{aligned}$$

Linearizing $y^i(t) = h^i(x(t), t)$ with respect to the spacecraft state yields the measurement Jacobian

$$H^i(x, t) = \frac{\partial h^i}{\partial x}(x, t) \in \mathbb{R}^{3 \times 4},$$

with rows

$$\begin{aligned} \frac{\partial \rho^i}{\partial x} &= \begin{bmatrix} \frac{\Delta X_i}{\rho_i} & 0 & \frac{\Delta Y_i}{\rho_i} & 0 \end{bmatrix}, \\ \frac{\partial \dot{\rho}^i}{\partial x} &= \begin{bmatrix} \frac{\rho_i \Delta \dot{X}_i - a_i \Delta X_i / \rho_i}{\rho_i^2} & \frac{\Delta X_i}{\rho_i} & \frac{\rho_i \Delta \dot{Y}_i - a_i \Delta Y_i / \rho_i}{\rho_i^2} & \frac{\Delta Y_i}{\rho_i} \end{bmatrix}, \\ \frac{\partial \phi^i}{\partial x} &= \begin{bmatrix} -\frac{\Delta Y_i}{\rho_i^2} & 0 & \frac{\Delta X_i}{\rho_i^2} & 0 \end{bmatrix}. \end{aligned}$$

Stacking $H^i(x, t)$ over all visible stations at time t gives the full measurement Jacobian $H(x, t)$ used in the linearized output model.

A full handwritten derivation is available in the appendix below.

2.2 Discrete-Time Linearization

Continuous-time Linearized Model

We linearize the nonlinear dynamics about a time-varying nominal (noise-free) trajectory $x_{nom}(t)$ driven by a nominal input $u_{nom}(t)$. The perturbation variables are defined as

$$\delta x(t) = x(t) - x_{nom}(t), \quad \delta u(t) = u(t) - u_{nom}(t), \quad \tilde{w}(t) = \tilde{w}(t),$$

and for each station i ,

$$\delta y^i(t) = y^i(t) - y_{nom}^i(t), \quad y_{nom}^i(t) = h^i(x_{nom}(t), t).$$

We can therefore find that the continuous time linearized perturbation model is the following:

$$\delta \dot{x}(t) = A(x_{nom}(t)) \delta x(t) + B \delta u(t) + \Gamma \tilde{w}(t),$$

where $A(x)$, B , and Γ are the Jacobians derived in Part I(a). Importantly, as the nominal linearization point is continuously changing and A is evaluated *along* the nominal trajectory, A is time-varying:

$$A_k = A(x_{nom}(t_k)), \quad t_k = k\Delta T.$$

We also find that the linearized measurement model $y^i(t)$ about $x_{nom}(t)$ is equal to the following:

$$\delta y^i(t) = H^i(x_{nom}(t), t) \delta x(t),$$

where $H^i(x, t) = \partial h^i / \partial x$ is the 3×4 measurement Jacobian for station i . Stacking the rows for all visible stations at time t gives the full measurement Jacobian $H(x_{nom}(t), t)$, and the CT linearized output equation becomes

$$\delta y(t) = H(x_{nom}(t), t) \delta x(t).$$

The resulting CT linearized system thus has state dimension $n = 4$, input dimension $m = 2$, disturbance dimension 2, and output dimension $p = 3N_{\text{vis}}(t)$, where $N_{\text{vis}}(t)$ is the number of ground stations that can see the spacecraft at time t .

Eulerized Discrete-time Linearization

With sampling period $\Delta T = 10$ s, we can find a discrete-time linear perturbation model by applying a Euler approximation to the CT linearized dynamics. At time t_k we have

$$\delta \dot{x}(t_k) = A_k \delta x(t_k) + B \delta u_k + \Gamma \tilde{w}_k, \quad \text{where } A_k = A(x_{nom}(t_k)),$$

and can approximate the time derivative as

$$\delta \dot{x}(t_k) \approx \frac{\delta x_{k+1} - \delta x_k}{\Delta T}.$$

Solving for δx_{k+1} and collecting terms gives

$$\delta x_{k+1} \approx (I + \Delta T A_k) \delta x_k + \Delta T B \delta u_k + \Delta T \Gamma \tilde{w}_k.$$

We therefore define the Eulerized DT Jacobian matrices

$$\tilde{F}_k = I + \Delta T A_k, \quad \tilde{G}_k = \Delta T B, \quad \tilde{\Omega}_k = \Delta T \Gamma,$$

so that the discrete-time linearized perturbation dynamics can be written as

$$\delta x_{k+1} = \tilde{F}_k \delta x_k + \tilde{G}_k \delta u_k + \tilde{\Omega}_k \tilde{w}_k.$$

In our implementation, A_k is recomputed at each time step using the current nominal state $x_{nom}(t_k)$, and the corresponding $\tilde{F}_k$, $\tilde{G}_k$, and $\tilde{\Omega}_k$ are updated accordingly.

Evaluating the measurement Jacobian along the nominal trajectory gives

$$H_k = H(x_{nom}(t_k), t_k),$$

and the DT linearized measurement model for the stacked measurements of all visible stations is

$$\delta y_k = H_k \delta x_k.$$

2.3 Dynamics Simulation

Initial Conditions and Perturbation

The total initial state is the sum of the nominal state x_{nom} and the initial state perturbation $x_{0,perturb}$ as provided by the problem statement and solution sketch:

$$x_{nom} = \begin{bmatrix} 6678 \\ 0 \\ 0 \\ r_0 \cdot \sqrt{\frac{\mu}{r_0^3}} \end{bmatrix} = \begin{bmatrix} 6678 \\ 0 \\ 0 \\ 7.72583519755957 \end{bmatrix}$$

$$x_{0,perturb} = \begin{bmatrix} 0 \\ 0.075 \\ 0 \\ -0.021 \end{bmatrix}$$

$$x_0 = x_{nom} + x_{0,perturb}$$

We assume no process noise, no measurement noise, and no control input perturbations.

2.3.1 Station Visibility

In determining visibility of the vehicle from a given ground station, both linear and nonlinear simulations use the full simulated state to determine visibility, rather than simply the perturbation. To ensure the vehicle's angle is within $\pi/2$, the following expression is used, to find the shortest possible angle between the two (vehicle and ground station), regardless of the quadrant between the two. This works at least when ϕ_i and θ_i are defined in the range $[-\pi, \pi]$.

$$((\phi_i - \theta_i) + \pi \mod 2\pi) - \pi$$

The visibility condition is equivalent to determining if the angle between the position vector of the vehicle relative to the ground station and the ground station and the origin of the earth centered reference are within $\pi/2$ radians, so alternatively, a method involving solving for the dot product of these two vectors can be used. This visibility can be visualized by plotting the state cartesian coordinates at arbitrary timesteps.

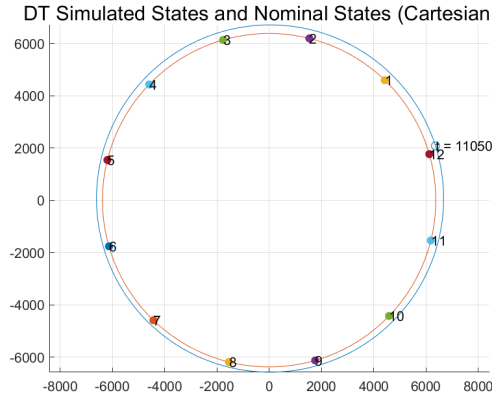


Figure 1: Vehicle position visualization.

2.3.2 Linearized DT simulation

State Simulation

The linearized DT simulation models the dynamics of the perturbations, and combines with the analytical solution of the "nominal" state, and so the full state is given as:

$$x_k = x_{nom}(k) + \delta x(k)$$

The nominal trajectory is considered as a pure 2D circular orbit, with no inclination, and so the position of the vehicle is fully described by the nominal radius r and the angular displacement

relative to the earth centered inertial frame described in the project description. $x_{nom}(k)$ in polar coordinates is calculated as:

$$x_{nom,r}(k) = r, x_{nom,\theta}(k) = \omega t + x_{nom,\theta}$$

$$x_{nom,polar}(k) = \begin{bmatrix} r(k) \\ \dot{r}(k) \\ \theta(k) \\ \dot{\theta}(k) \end{bmatrix} = \begin{bmatrix} r \\ r\omega \\ \omega t + x_{nom,\theta} \\ \omega \end{bmatrix}$$

$x_{nom}(k)$ in cartesian coordinates is then calculated as:

$$x_{nom}(k) = \begin{bmatrix} r \cos(\theta) \\ -r\dot{\theta} \sin(\theta) \\ r \sin(\theta) \\ r\dot{\theta} \cos(\theta) \end{bmatrix}$$

$$\delta x_k = F_k \delta x_{k-1} + G_k \delta u_k + w_k$$

Of course, for part 1, this simplifies to: $\delta x_k = F_k \delta x_{k-1}$

Measurement Simulation

Similarly to the states, the linearized model of the perturbation of $y(k)$ is modeled and combined with the analytical solution of the "nominal" state, and so the full measurement is given as:

$$y(k) = y_{nom}(k) + \delta y(k)$$

$y_{nom}(k)$ is found using the full nonlinear expressions for measurement described in the full nonlinear simulation section and in the earlier derivation section, however at $x_{nom}(k)$.

$\delta y(k)$ is calculated using the linearized expression, again described in parts a and b.

2.3.3 Full nonlinear simulation

Simulation Details

The state dynamics were simulated using the `ode45` function in Matlab. The time step ΔT was defined as 10s and the total simulation time must cover at least one full orbit period, but a simulation duration of 14,000s was selected, to cover multiple orbital periods.

Nonlinear State equations

The current radius at time t is calculated using:

$$r(t) = \sqrt{X^2(t) + Y^2(t)} = \sqrt{x_1^2(t) + x_3^2(t)}$$

The nonlinear state equations are then given as follows:

$$\begin{aligned} \dot{x}_1 &= \dot{X} = x_2 \\ \ddot{x}_2 &= \ddot{X} = -\mu \cdot \frac{x_1}{r^3} \\ \dot{x}_3 &= \dot{Y} = x_4 \\ \ddot{x}_4 &= \ddot{Y} = -\mu \cdot \frac{x_3}{r^3} \end{aligned}$$

with $\mu = 398600 km^3/s^2$ being the standard gravitational parameter.

Nonlinear measurement equations

The size of the measurement vector $y(t)$ changes with time t as it depends on which stations can currently see the spacecraft and therefore provide a valid measurement. Its dimension is $p \cdot N \times 1$ with N being the number of stations that can observe the spacecraft at time t .

The measurement vector $y^i(t)$ for each station i at time t is defined as follows:

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

This assumes, there is no measurement error and therefore the measurement error vector $\tilde{v}^i(t)$ is zero at each time t and for each station i . Each element of this measurement vector is calculated as follows:

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

Here $[X(t), Y(t), \dot{X}(t), \dot{Y}(t)]$ are given by the current state vector of the spacecraft. But $[X_s^i(t), Y_s^i(t), \dot{X}_s^i(t), \dot{Y}_s^i(t)]$ are the measurement station i 's state vector at time t . Its state equations for the positions are given by the problem statement as follows:

$$\begin{aligned} X_s^i(t) &= R_E \cdot \cos(\omega_E t + \theta^i(0)) \\ Y_s^i(t) &= R_E \cdot \sin(\omega_E t + \theta^i(0)) \end{aligned}$$

with $R_E = 6378 \text{ km}$ (Earth's radius), $\omega_E = \frac{2\pi}{86400} \text{ rad/s}$ (Earth's rotation speed) and the initial angle $\theta^i(0)$ of measurement station i at time $t = 0$ given as:

$$\theta^i(0) = (i - 1) \cdot \frac{\pi}{6}$$

For the velocity components of the measurement station's state vector, we take the time derivative of the position equations:

$$\begin{aligned} \dot{X}_s^i(t) &= \frac{d}{dt} X_s^i(t) = -\omega_E \cdot R_E \cdot \sin(\omega_E t + \theta^i(0)) \\ \dot{Y}_s^i(t) &= \frac{d}{dt} Y_s^i(t) = \omega_E \cdot R_E \cdot \cos(\omega_E t + \theta^i(0)) \end{aligned}$$

Spacecraft visibility to measurement station

Due to the relative motion of the spacecraft and each station, the number of stations providing a valid observation vector $y^i(t)$ varies with time and each station i only provides a valid observation when $\phi^i(t)$ is in the given range:

$$\begin{aligned} \phi^i(t) &\in \left[-\frac{\pi}{2} + \theta^i(t), \frac{\pi}{2} + \theta^i(t)\right] \\ \text{where } \theta^i(t) &= \tan^{-1} \left(\frac{Y^i(t)}{X^i(t)} \right) \end{aligned}$$

2.3.4 Results and Plots

The DT and nonlinear models are simulated with the same initial state perturbation, initial total state, and number of timesteps as provided for the sanity checks, and inspecting of exact values shows an exact match.

2.3.5 DT Linearized Simulation

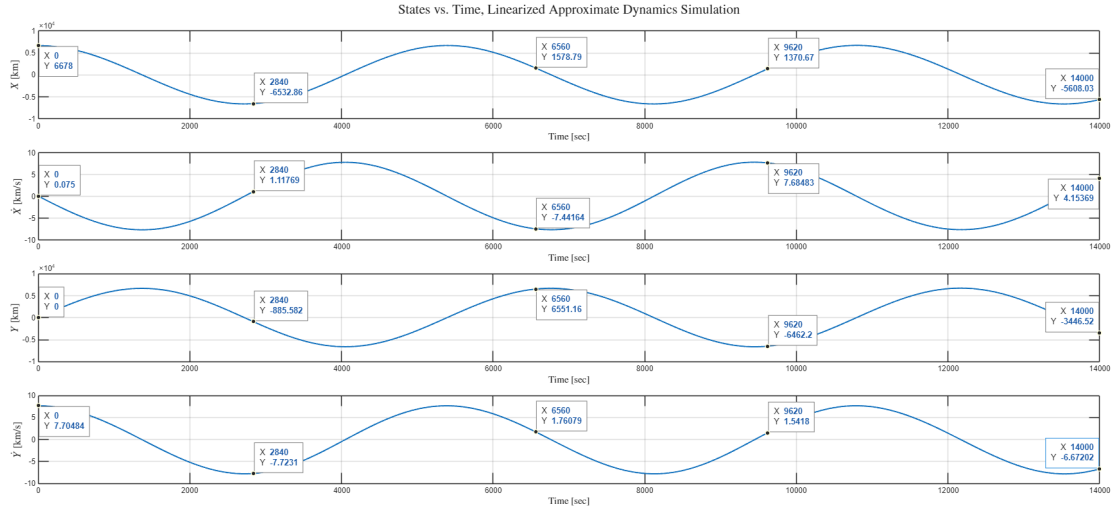


Figure 2: DT Simulation States and Nominal States

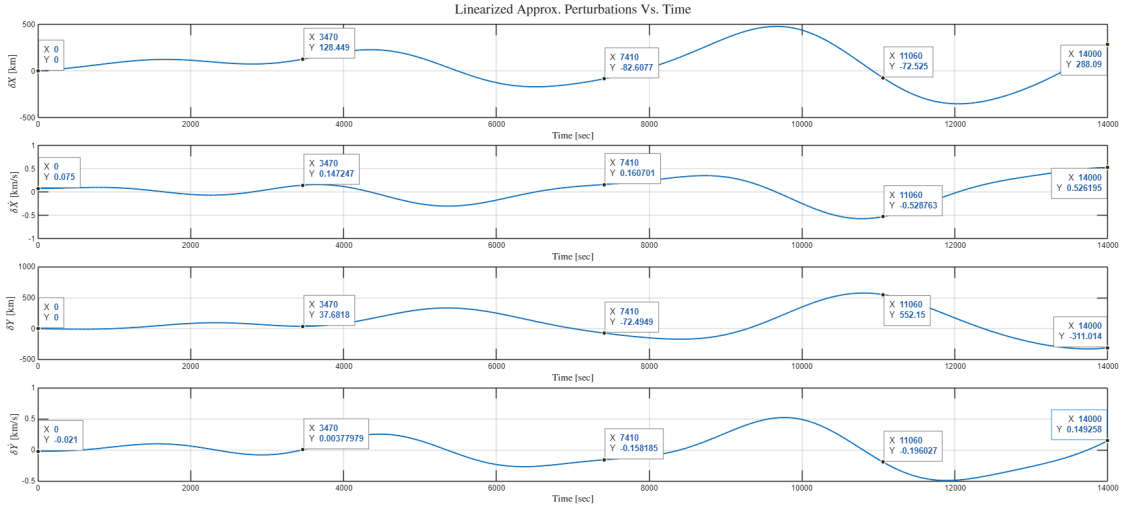


Figure 3: DT Linearized Simulated State Perturbations

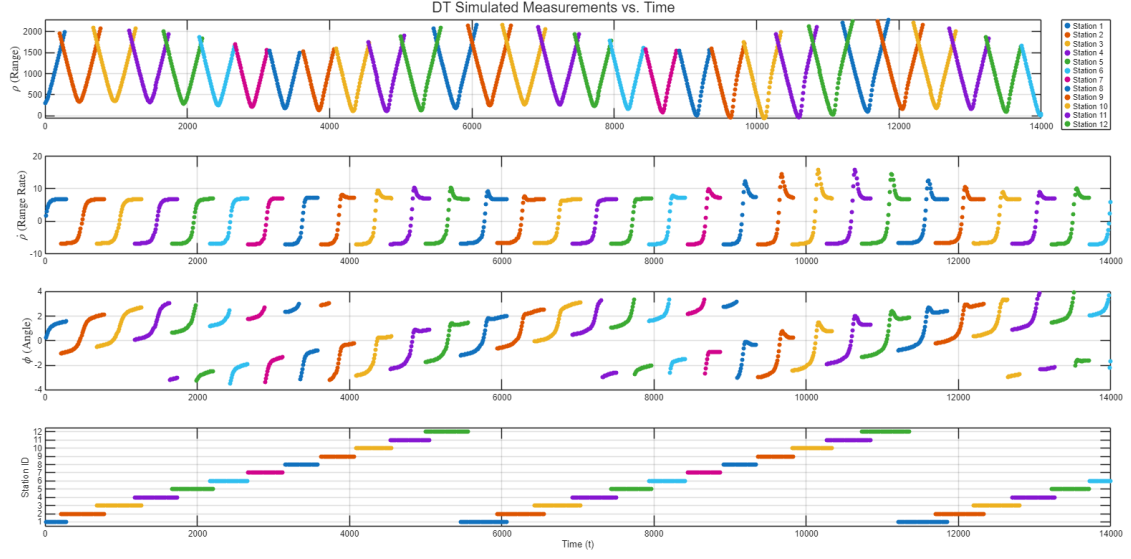


Figure 4: DT Linearized Simulated Measurements

2.3.6 Nonlinear Simulation

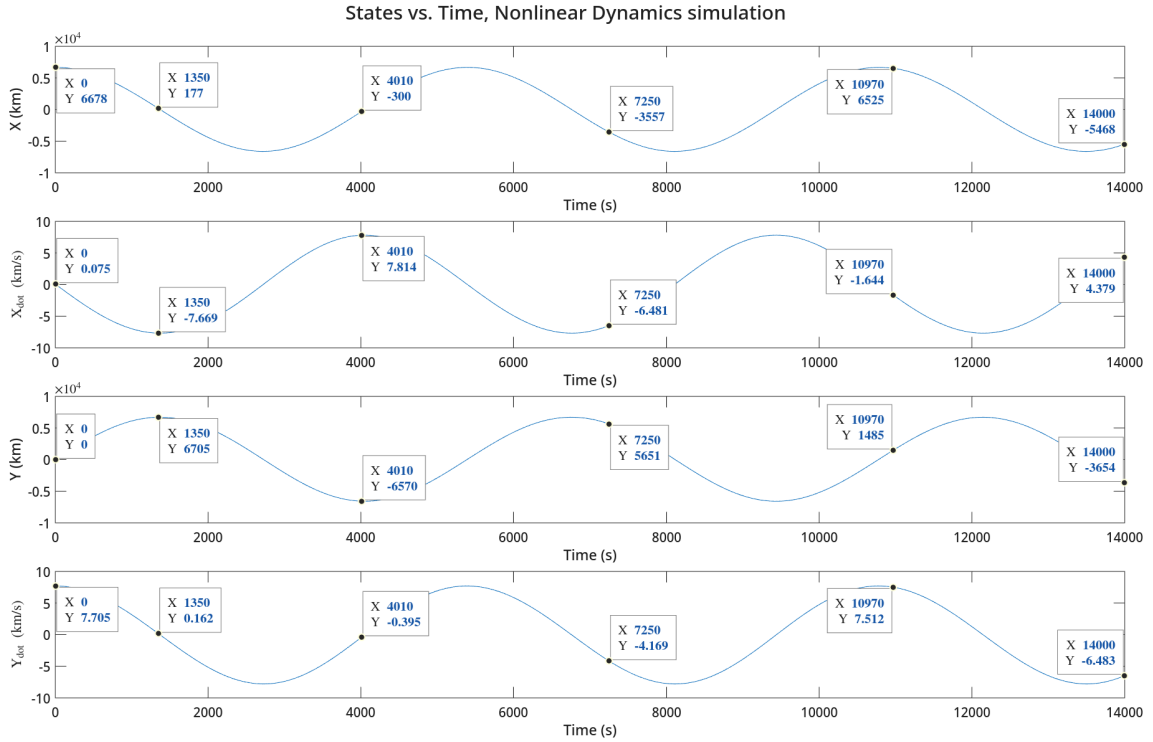


Figure 5: States vs. Time, Nonlinear Dynamics Simulation

2.3.7 Nonlinear Simulated Measurements vs. Time

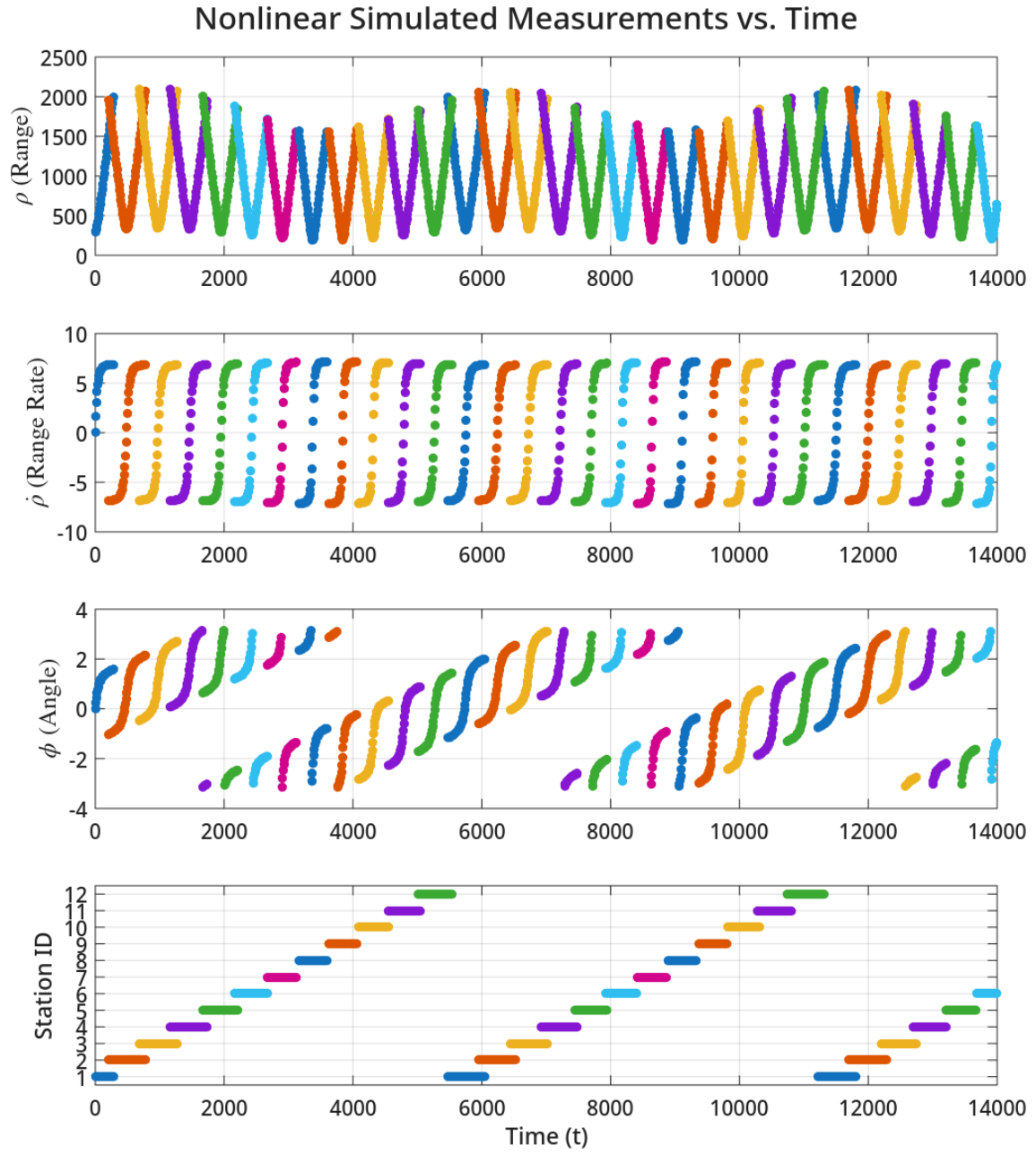


Figure 6: Nonlinear Simulated Measurements vs. Time

2.3.8 Comparison

The DT linearized simulation states can be simultaneously plotted, and the error of the DT linearized simulation versus the nonlinear simulation can be calculated. At the scale of the full states, it's difficult to see any error. It can be seen that error increases with time in every part of the state.

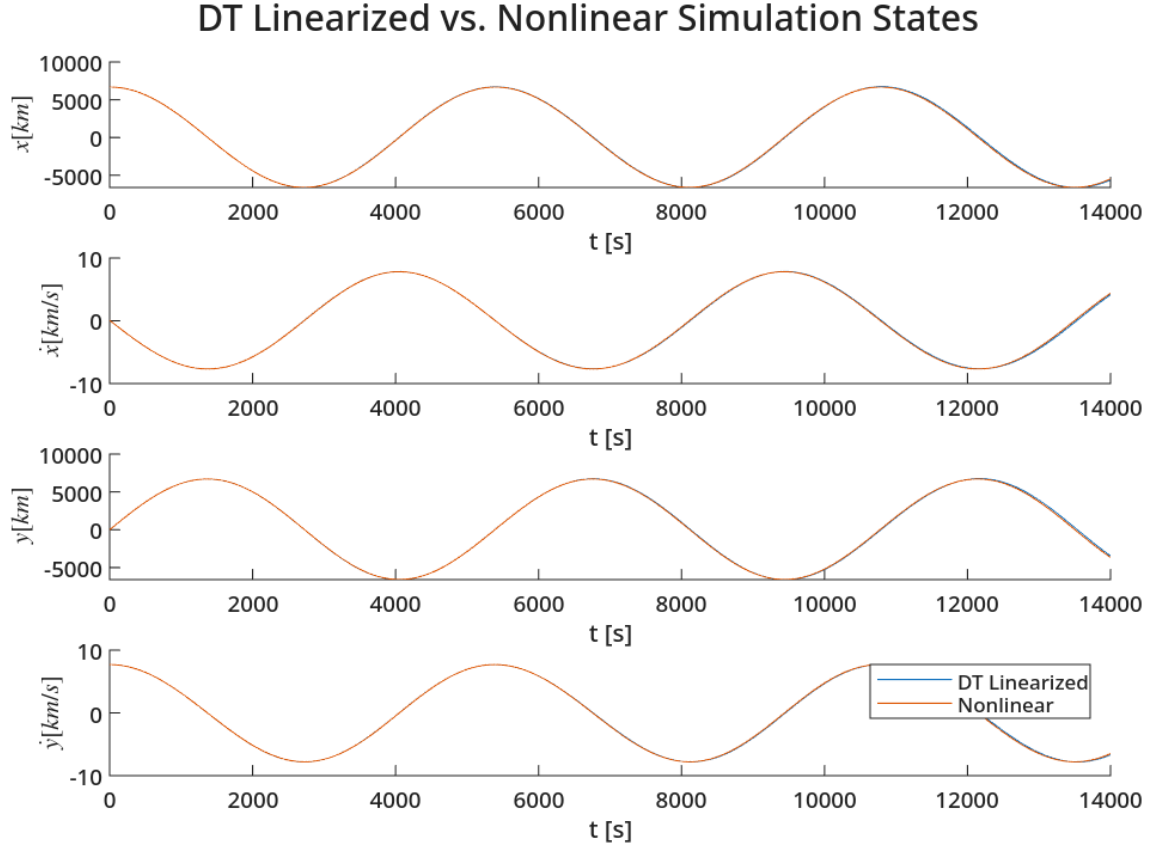


Figure 7: DT Linearized vs. Nonlinear Simulation States

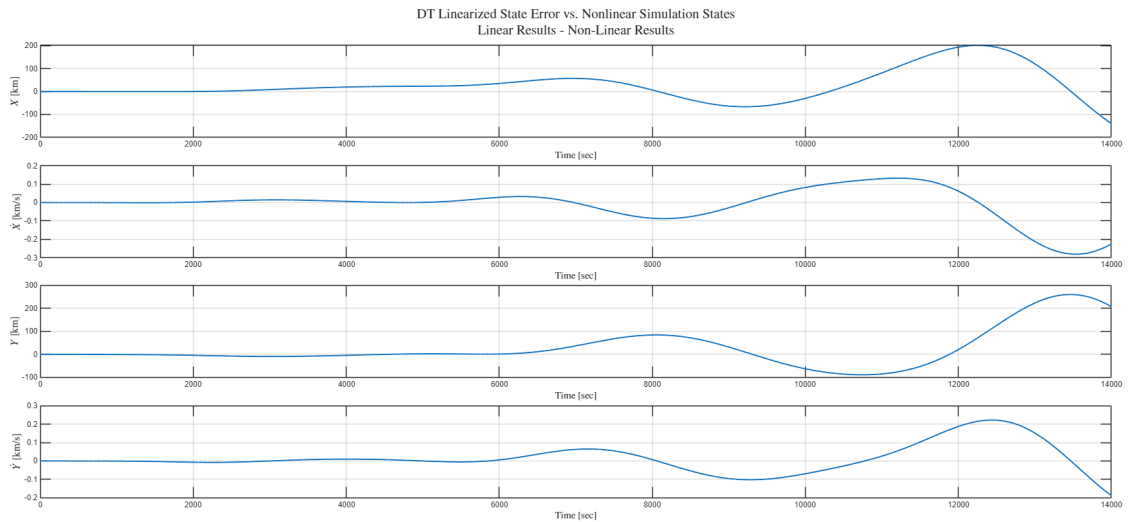


Figure 8: DT Linearized State Error vs. Nonlinear Simulation States

Measurement Comparison

Due to the complexity of comparing all measurements, this comparison is omitted, but it can be seen visually that the DT measurements deviate from the nonlinear measurements with increased time, particularly in the range rate measurements.

3 Part II - Stochastic Nonlinear Filtering

3.1 Stochastic Nonlinear Filter Validation via Monte Carlo Analysis

To validate the statistical consistency and performance of the Linearized Kalman Filter (LKF) and the Extended Kalman Filter (EKF), a **Monte Carlo Truth Model Test (TMT)** simulation was conducted. This involved running the filter $N = 1,000$ times over a trajectory of $K = 1,400$ time steps (14,000s, more than a full orbit, providing multiple station visibility cycles and sufficient data points for χ^2 -testing). N was chosen as 1,000 to ensure sufficient accuracy, as according to the **Law of Large Numbers**, the standard error of a normal distribution, as N grows, shrinks proportionally to $\frac{1}{\sqrt{N}}$.

3.1.1 Ground Truth Simulation

The true state trajectory, $\mathbf{x}_{\text{true}}(k+1)$, is generated using the full nonlinear dynamics (using Matlab's `ode45`) and full nonlinear measurements, incorporating true process noise $\mathbf{w}_k$ and measurement noise $\mathbf{v}_k$ according to the provided process noise covariance matrix $\mathbf{Q}_{\text{true}}$ and measurement noise covariance matrix $\mathbf{R}_{\text{true}}$.

Discrete-Time Noise Modeling

To get a discrete-time process noise sample from the continuous-time process noise covariance $\mathbf{Q}_{\text{true}}$, we first pick a sample ξ_k and then multiply that with $\tilde{\mathbf{\Omega}}_k$ to get DT process noise $\mathbf{w}_k$.

$$\begin{aligned}\xi_k &\sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{\text{true}}) \\ \tilde{\mathbf{\Omega}}(\mathbf{t}) &= \mathbf{\Gamma}(\mathbf{t})\Delta T \\ \mathbf{w}_k &= \tilde{\mathbf{\Omega}}_k \xi_k = \tilde{\mathbf{\Omega}}(\mathbf{k}\Delta\mathbf{T})\xi_k\end{aligned}$$

where $\mathbf{\Gamma}$ relates process noise to the state dynamics, and ΔT is the sampling time (10 sec).

The process noise $\mathbf{w}_k$ and measurement noise $\mathbf{v}_k$ are additive zero-mean, white, and Gaussian (AWGN):

$$\begin{aligned}\mathbf{w}_k &\sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \\ \mathbf{v}_k &\sim \mathcal{N}(\mathbf{0}, \mathbf{R})\end{aligned}$$

Initial State Randomization

To ensure the statistical validity of the TMT, the true initial state $\mathbf{x}_{\text{true}}(0)$ is randomly instantiated using the filter's initial estimate $\hat{\mathbf{x}}^+(0)$ and its state estimate covariance $\mathbf{P}^+(0)$:

$$\mathbf{x}_{\text{true}}(0) \sim \mathcal{N}(\hat{\mathbf{x}}^+(0), \mathbf{P}^+(0))$$

3.1.2 Filter Consistency Testing

The consistency of the filter's estimated covariance matrices ($\mathbf{P}$ and $\mathbf{S}$) with the actual errors and innovations is evaluated using the Normalized Estimation Error Squared (NEES) and Normalized Innovation Squared (NIS) metrics. A significance level of $\alpha = 0.05$ was chosen for the χ^2 tests. With $\alpha = 0.05$, we obtain a 95% confidence interval for the test statistic, which is a standard compromise between being too strict and too permissive. Therefore, for our correctly tuned filter, we still expect about 5% of the NEES/NIS points to fall outside the bounds purely by chance.

Normalized Estimation Error Squared (NEES)

The NEES statistic $\epsilon_x(k)$ quantifies the normalized state estimation error $\tilde{\mathbf{x}}(k) = \mathbf{x}_{\text{true}}(k) - \hat{\mathbf{x}}(k)$.

- **Statistic:** The NEES statistic at time k is calculated as:

$$\epsilon_x(k) = \tilde{\mathbf{x}}(k)^T \mathbf{P}^+(k)^{-1} \tilde{\mathbf{x}}(k)$$

where $\mathbf{P}^+(k)$ is the filter's updated state estimation covariance.

- **LKF Adaptation:** While the LKF estimates the **perturbation state** $\delta\mathbf{x} = \mathbf{x} - \mathbf{x}_{\text{nom}}$, it still returns the full state and thus the error can be computed the same way instead of using the perturbation:

$$\tilde{\mathbf{x}}(k) = (\mathbf{x}_{\text{true}}(k) - \mathbf{x}_{\text{nom}}(k)) - (\hat{\mathbf{x}}^+(k) - \mathbf{x}_{\text{nom}}(k)) = \mathbf{x}_{\text{true}}(k) - \hat{\mathbf{x}}^+(k)$$

- **EKF Adaptation:** The EKF estimates the **total state** $\hat{\mathbf{x}}$, so the error is the total state error anyway:

$$\tilde{\mathbf{x}}(k) = \mathbf{x}_{\text{true}}(k) - \hat{\mathbf{x}}^+(k)$$

- **Test:** The average NEES (ANEES) across N runs is compared to the χ^2 confidence interval with $N \cdot n$ degrees of freedom, where $n = 4$ is the state dimension.

$$\chi_{Nn, \alpha/2}^2 \leq \sum_{i=1}^N \epsilon_x^i(k) \leq \chi_{Nn, 1-\alpha/2}^2$$

Normalized Innovation Squared (NIS)

The NIS statistic $\epsilon_y(k)$ quantifies the normalized measurement innovation (residual) $\tilde{\mathbf{y}}(k) = \mathbf{y}_{\text{true}}(k) - \hat{\mathbf{y}}(k)$.

- **Statistic:** The NIS statistic at time k is calculated as:

$$\epsilon_y(k) = \tilde{\mathbf{y}}(k)^T \mathbf{S}(k)^{-1} \tilde{\mathbf{y}}(k)$$

where $\mathbf{S}(k)$ is the innovation covariance matrix.

- **LKF Adaptation:** The LKF innovation $\tilde{\mathbf{y}}(k)$ uses the difference between the true measurement and the linearized predicted measurement:

$$\tilde{\mathbf{y}}(k) = \mathbf{y}_{\text{true}}(k) - (\mathbf{h}(\hat{\mathbf{x}}^-(k)) - \mathbf{H}(\mathbf{x}_{\text{nom}}(k))\delta\hat{\mathbf{x}}^-(k))$$

- **EKF Adaptation:** The EKF innovation uses the difference between the true measurement and the predicted measurement from the full nonlinear measurement function $\mathbf{h}$:

$$\tilde{\mathbf{y}}(k) = \mathbf{y}_{\text{true}}(k) - \mathbf{h}(\hat{\mathbf{x}}^-(k))$$

- **Test:** The average NIS (ANIS) across N runs should be compared to the χ^2 confidence interval with $N \cdot p$ degrees of freedom, where p is the **time-varying** measurement dimension (i.e., $3 \times$ number of visible stations), but for simplicity, a constant $p = 3$ is considered).

$$\chi_{Np, \alpha/2}^2 \leq \sum_{i=1}^N \epsilon_y^i(k) \leq \chi_{Np, 1-\alpha/2}^2$$

3.1.3 Filter Tuning Methodology

The filter needs to be tuned by iterating on the "guessed" process noise covariance $\mathbf{Q}_{\text{KF}}$ (which may not equal $\mathbf{Q}_{\text{true}}$) until both the ANEES and ANIS plots largely fall within their respective $1 - \alpha$ confidence bounds. A consistent filter (ANEES/ANIS within bounds) indicates that the filter's covariance matrices are correctly reflecting the magnitude of the actual state estimation errors and measurement innovations.

3.2 Linearized Kalman Filter (LKF)

The LKF performs its estimation in a loop, for each timestep, according to the specified sample time.

3.2.1 Initialization

As an initial guess, the Linearized Kalman Filter is initialized with the following values:

$$\delta\hat{x}_0^+ = [0, 0, 0, 0]^T$$

$$P_0^+ = \text{diag}(\sigma_{x0}^2, \sigma_{v0}^2, \sigma_{x0}^2, \sigma_{v0}^2) = \text{diag}(1 \cdot 10^{-6}, 1 \cdot 10^{-5}, 1 \cdot 10^{-6}, 1 \cdot 10^{-6})$$

Having an initial estimate of $\hat{x} = x_{\text{nom}}$ is just our best guess, as we don't have any additional information on what the initial perturbation could look like. Setting P_0^+ as above, tells the filter to trust the position estimate slightly more than the velocity estimate (higher certainty) and that the initial estimate generally is quite good. Setting it as a diagonal matrix comes from the fact that we don't expect errors in these state variables to be correlated.

3.2.2 Time update/prediction step

$$\begin{aligned}\delta\hat{x}_{k+1}^- &= \tilde{F}_k \delta\hat{x}_k^+ + \tilde{G}_k \delta\hat{u}_k \\ P_{k+1}^- &= \tilde{F}_k P_k^+ \tilde{F}_k^T + \tilde{\Omega}_k Q_k \tilde{\Omega}_k^T\end{aligned}$$

The calculation of $\tilde{F}_k$ is described in Part I. In this orbit determination problem, the input to the plant is not considered, so $\tilde{G}_k \delta\hat{u}_k = 0$. $\tilde{\Omega}_k$ is calculated using a similar Euler's method approximation to $\tilde{F}_k$.

$$\tilde{\Omega}_k|_{\text{nom}[k]} \approx \Delta T \cdot \Gamma(t)|_{(t=t_k)}$$

3.2.3 Index Convention

Because the $k=0$ value of the state is an initial guess, the above is rearranged to obtain the estimate based on the previous timestep, where applicable. $\delta\hat{x}_{k+1}^-$ is replaced with $\delta\hat{x}_k^-$, and $\delta\hat{x}_k^-$ is replaced with $\delta\hat{x}_{k-1}^-$, so for example:

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

This has not been necessarily reflected in this report, but is reflected in the implementation.

3.2.4 Measurement update/correction step

When a measurement is available, the measurements info is treated as y_k . Because the LKF estimates perturbations, this is converted into an equivalent perturbation measurement based on the difference with the expected measurement from the nominal trajectory at t_k :

$$\delta y_{k+1} = y_{k+1} - y_{k+1}^*$$

To handle measurements from multiple ground stations, each nominal measurement must be found using the nominal state and the corresponding station, and they must be stacked, in a process very similar to the DT simulation. Throughout this report, no distinction is necessarily made that a "Stacked" version of the matrix is being used, and the matrix simply treated as having a size that changes with time.

$$\tilde{H}_{k+1,i} = \left. \frac{\partial h}{\partial x_i} \right|_{x=\hat{x}_{k+1}^-} \quad H_k = \begin{bmatrix} \tilde{H}_1 \\ \vdots \\ \tilde{H}_n \end{bmatrix} \quad y_{k+1,i}^* = h(x_{k+1,i}^*) \quad H_k = \begin{bmatrix} y_{k+1,1}^* \\ \vdots \\ y_{k+1,n}^* \end{bmatrix}$$

The equivalent R_{k+1} must be found by combining the R_{k+1} values in block diagonal form.

$$R = \begin{bmatrix} R & 0 \\ 0 & R \end{bmatrix}$$

In practice, though, there is only a maximum of two ground station measurements at any given time, due to their spacing. However, when calculating the perturbation, care must be taken to subsequently wrap the angle component! If the wrapped measurement perturbation is $\delta y'_{k+1}$:

$$\phi'_k = \begin{cases} \phi_k - 2\pi & \phi_k > \pi \\ \phi_k + 2\pi & \phi_k < -\pi \\ \phi_k & \text{otherwise} \end{cases}$$

Now this can be used to find the innovation vector for the measured perturbation vs the predicted perturbation.

$$e_{y,k} = \delta y_{k+1} - \tilde{H}_{k+1} \delta \hat{x}_{k+1}^-$$

The Kalman gain can be found with:

$$K_{k+1} = P_{k+1}^- \tilde{H}_{k+1}^T [\tilde{H}_{k+1} P_{k+1}^- \tilde{H}_{k+1}^T + R_{k+1}]^{-1}$$

The measurement covariance (which will also be used to evaluate the LKF) is found separately, and also used to get the Kalman gain:

$$S_k = \tilde{H}_{k+1} P_{k+1}^- \tilde{H}_{k+1}^T + R_{k+1} \rightarrow K_{k+1} = P_{k+1}^- \tilde{H}_{k+1}^T [S_k]^{-1}$$

We can now find the estimate given the previous best estimate and latest measurement, again using the innovation vector:

$$\delta \hat{x}_{k+1}^+ = \delta \hat{x}_{k+1}^- + K_{k+1} (\delta y_{k+1} - \tilde{H}_{k+1} \delta \hat{x}_{k+1}^-) \rightarrow \delta \hat{x}_{k+1}^+ = \delta \hat{x}_{k+1}^- + K_{k+1} (e_{y,k})$$

Now the Kalman gain, measurement matrix, and covariance of the prediction estimate can be used to calculate the updated estimate covariance.

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

No Measurements

If no measurements are available, the best estimate and covariance for that timestep is the best estimate and covariance based on previous predictions and measurements:

$$\hat{x}_{k+1}^+ = \hat{x}_{k+1}^- \quad P_{k+1}^+ = P_{k+1}^-$$

3.2.5 Output

The measurement covariance, innovation vector, and estimate covariance can all be used as-is, but the full state estimate must be calculated from the state of the nominal trajectory at the particular timestep.

$$\delta x(t) + x_{nom}(t) = x(t)$$

3.2.6 Results

a. Plots for a "typical" simulation instance

The "typical" simulation instance is initialized with an arbitrary value beyond the variances used for the monte-carlo, to show how large perturbations cause problems with the estimate. The initial covariance chosen is the same as in the truth model testing.

$$\delta x_0 = [0.0001, 0.005, 0.0001, -0.002]^T$$

The plot below shows the simulated noisy measurements used to test and tune the linearized Kalman Filter.

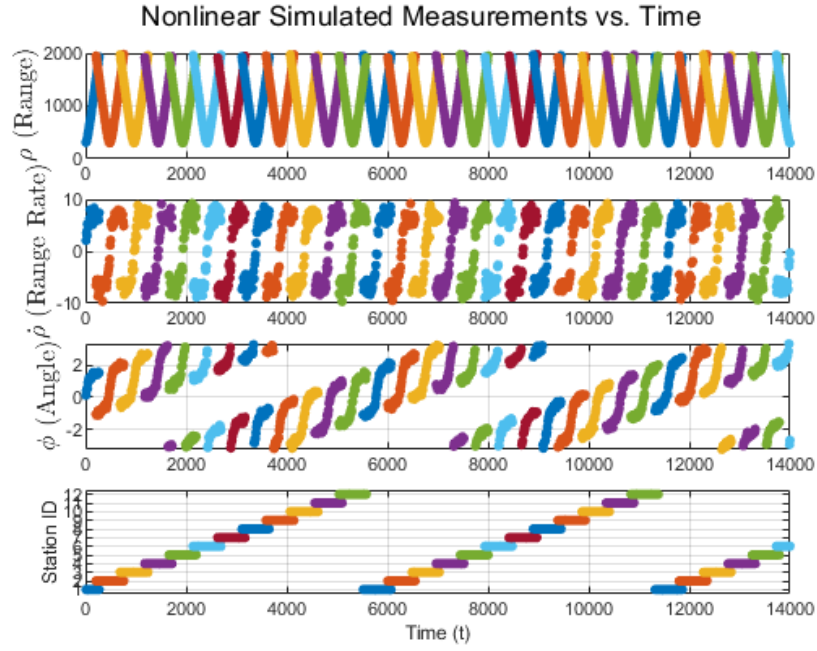


Figure 9: Typical Simulated Measurements

The plot below shows the state estimate from the linearized Kalman filter (LKF) compared with a typical simulation state that correspond to and result in the above simulated measurements.

Because this is a small perturbation, the initial estimate follows closely with the truth model state. As time goes on, the state drifts from the nominal trajectory, and the quality of the estimate decays rapidly. The error goes far outside the expected uncertainty in the measurement. On this scale, the uncertainty bounds are not visible, but the difference between the estimate and the truth model is:

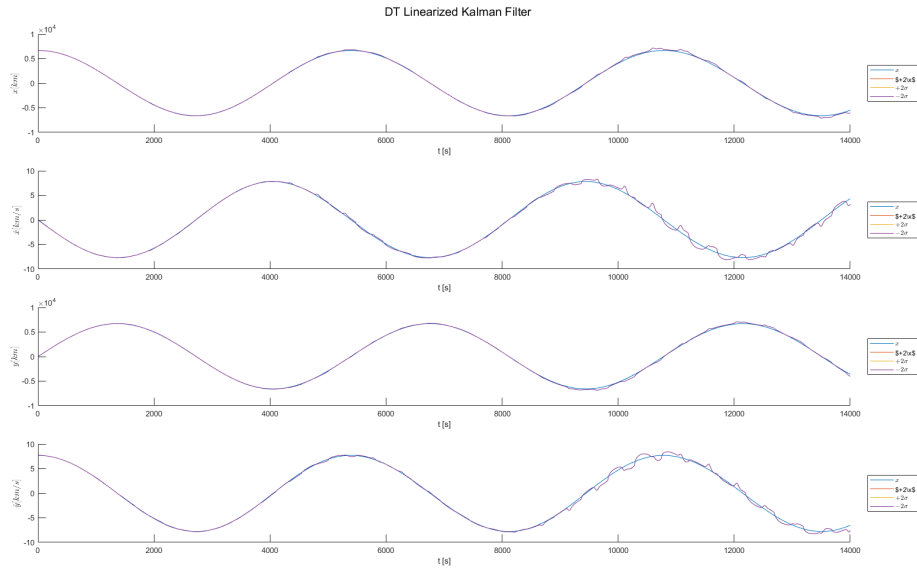


Figure 10: Discrete LKF State Estimate Compared with Simulated Measurement

Finally, the below plot shows the error state estimate from the LKF and the example truth state. Again, error increases to the point where it is not possible to see the estimate error compared to the uncertainty.

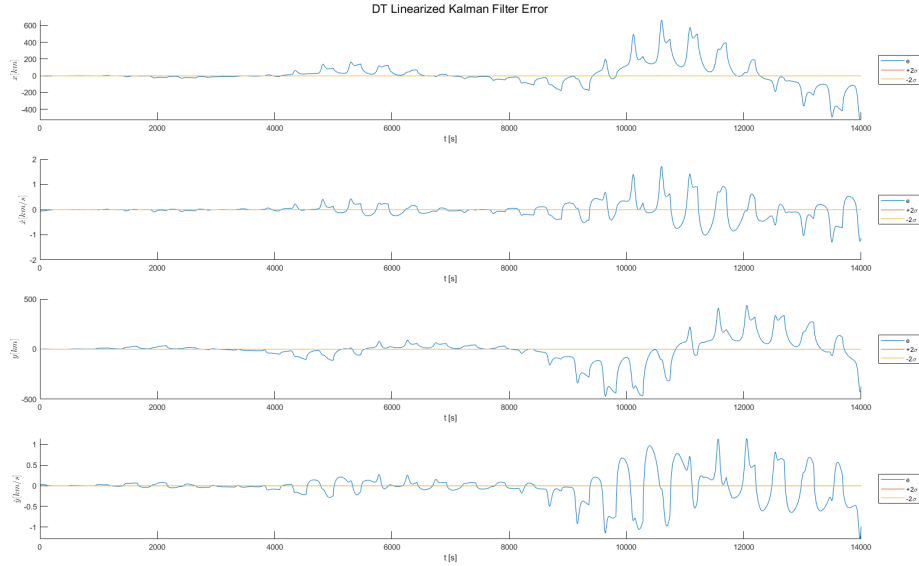


Figure 11: Linearized Kalman Filter Example Run Error

b. NEES test statistic points

These are the initial results for the NEES statistics of the Linearized Kalman Filter. As one can clearly see, the values grow massively out of bounds, which indicates that a lot of tuning needs to be done.

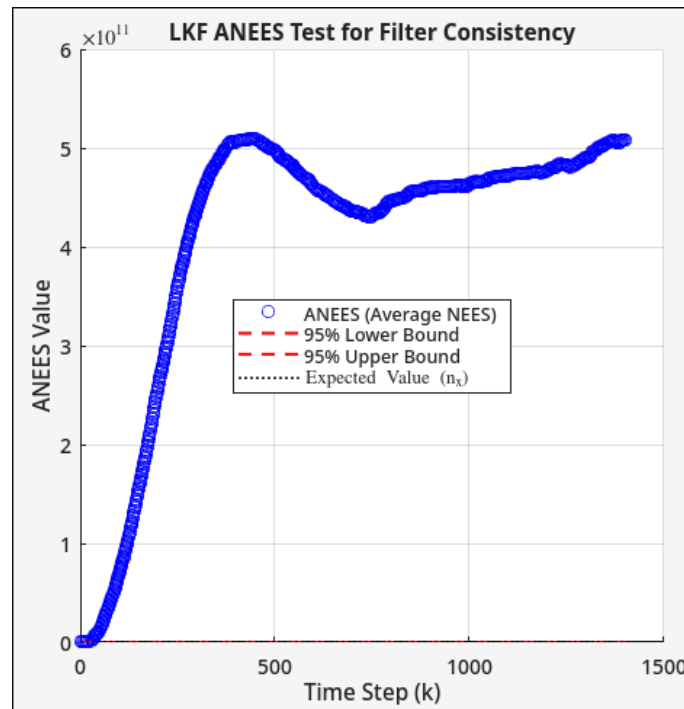


Figure 12: Initial result of NEES plot for LKF

It turns out there was actually more involved than just tuning the variances and parameters, and after refining our code (mostly inconsistencies in indices) and then tuning the parameters, LKF produced the following NEES plot:

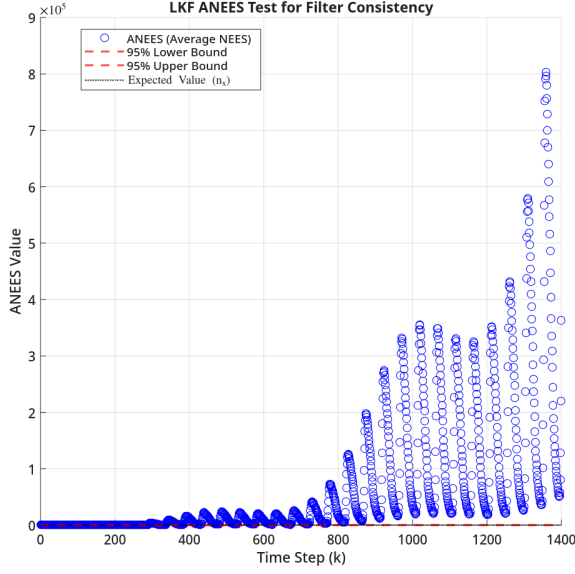


Figure 13: Final ANEES plot for LKF (full simulation)

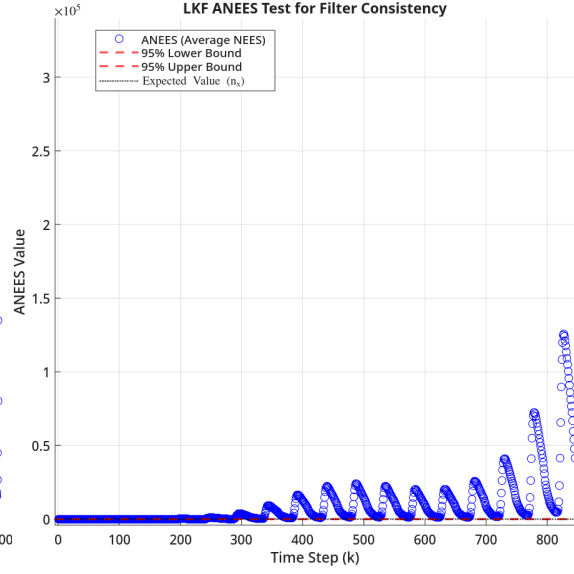


Figure 14: Final ANEES plot for LKF (zoomed into first 8500s)

As one can see in Figures 13 and 14, initially the error stays at or close to the expected bounds, but after about 3000s into the simulation, the values start to slowly oscillate and these errors start to accumulate until they increase exponentially after about 7000s. This behavior is to be expected as the LKF linearizes about a fixed nominal circular orbit, whereas the true trajectory with process noise changes over time. This accumulated error over time causes the linearization to break down.

c. NIS test statistic points

These are the initial results for the NIS statistics of the Linearized Kalman Filter. Same as with the NEES statistics, also here one can see that massive tuning is needed.

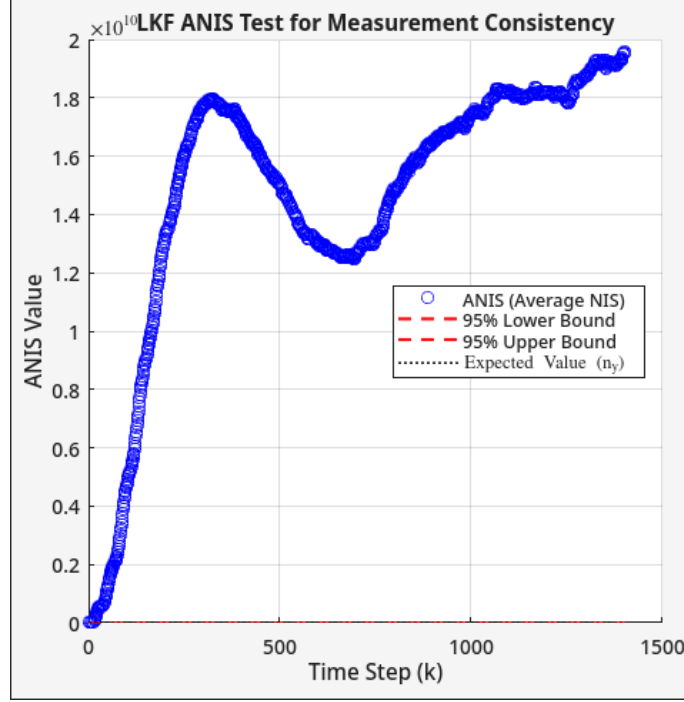


Figure 15: Initial result of NIS plot for LKF

After massively updating the code and arriving at the desired values for Q, P_0 , LKF produced the following NIS plot:

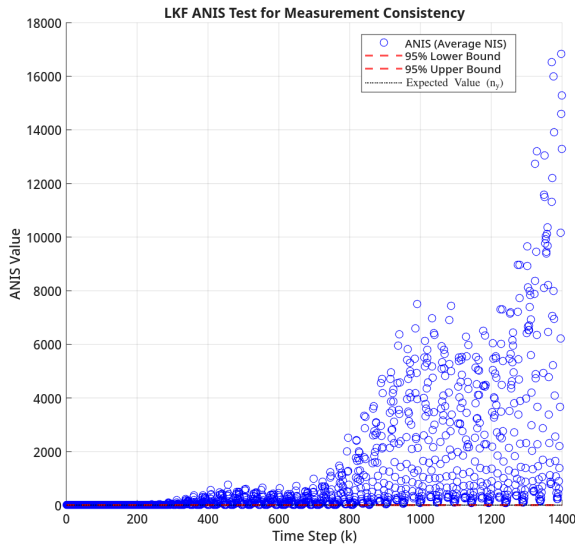


Figure 16: Final ANIS plot for LKF (full simulation)

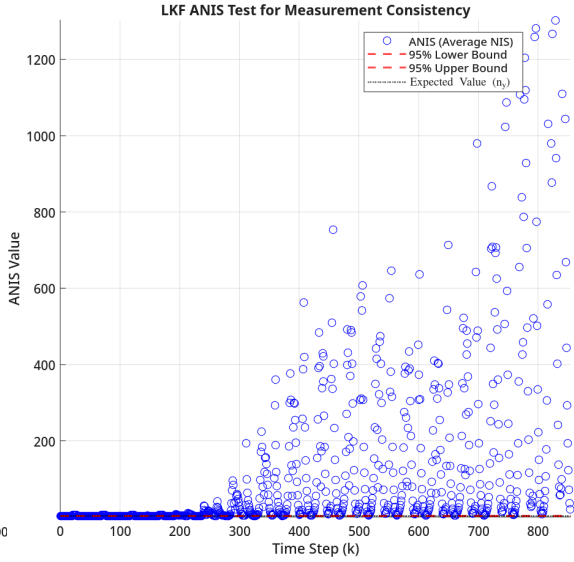


Figure 17: Final ANIS plot for LKF (zoomed into first 8500s)

Figures 16 and 17 show, that much like for NEES, also NIS initially stays at or close the expected bounds (with some outliers right from the beginning), but after about 2500s the NIS value starts to massively increase.

3.3 Extended Kalman Filter (EKF)

3.3.1 Initialization and EKF setup

For the Extended Kalman Filter (EKF), we estimate the full nonlinear spacecraft state

$$x_k = [X_k, \dot{X}_k, Y_k, \dot{Y}_k]^T,$$

directly, rather than a perturbation about a nominal trajectory. The discrete-time nonlinear dynamics and measurement models can be written as

$$x_{k+1} = f(x_k, u_k, w_k), \quad y_{k+1} = h(x_{k+1}, v_{k+1}),$$

where u_k is the (zero) control input, w_k is the process noise associated with unknown accelerations in the X and Y directions, and v_{k+1} is the measurement noise in range, range rate, and elevation angle from the ground stations.

The nominal circular orbit used throughout the project has radius

$$r_0 = 6678 \text{ km},$$

so that the nominal initial state is

$$x_{\text{nom},0} = \begin{bmatrix} r_0 \\ 0 \\ 0 \\ r_0 \sqrt{\frac{\mu}{r_0^3}} \end{bmatrix} = \begin{bmatrix} 6678 \\ 0 \\ 0 \\ 7.7258 \end{bmatrix} \text{ (km, km/s)}.$$

For the EKF we deliberately start from a slightly biased initial estimate with position and velocity offsets

$$\sigma_{x0} = 1 \cdot 10^{-6} \text{ km}, \quad \sigma_{v0} = 1 \cdot 10^{-5} \text{ km/s},$$

so that the initial state estimate is

$$\hat{x}_0^+ = x_{\text{nom},0} + \begin{bmatrix} \sigma_{x0} \\ \sigma_{v0} \\ \sigma_{x0} \\ \sigma_{v0} \end{bmatrix}$$

The corresponding initial covariance is chosen diagonal,

$$P_0^+ = \text{diag}(\sigma_{x0}^2, \sigma_{v0}^2, \sigma_{x0}^2, \sigma_{v0}^2) = \text{diag}(1 \cdot 10^{-6}, 1 \cdot 10^{-5}, 1 \cdot 10^{-6}, 1 \cdot 10^{-5}),$$

which encodes larger initial uncertainty in velocity than in position, but does not assume any cross-correlation between the states.

The truth-model simulation uses the provided continuous-time acceleration noise covariance Q_{true} and single-station measurement covariance R_{true} . For the EKF we introduce a separate guessed acceleration covariance

$$Q_{\text{KF}} = 10^{-10} I_2,$$

and tune its magnitude using NEES/NIS tests (see Part 5(b,c)). The EKF measurement covariance uses the same value as the truth model, i.e. $R_{\text{KF}} = R_{\text{true}}$. When multiple stations are visible at a given time, the filter builds

$$R_k = \text{blkdiag}(R_{\text{true}}, \dots, R_{\text{true}}),$$

with one 3×3 block per visible station.

At every time step, the required Jacobians are evaluated at the most recent EKF estimate: the dynamics Jacobian A_k is evaluated at $\hat{x}_k^+$ and the stacked measurement Jacobian $\tilde{H}_{k+1}$ is evaluated at $\hat{x}_{k+1}^-$.

3.3.2 Time update (prediction)

Starting from the previous estimate $(\hat{x}_k^+, P_k^+)$, the EKF prediction step proceeds as follows.

State prediction. The nonlinear orbital dynamics are numerically integrated over the sampling interval Δt_k with zero input,

$$\hat{x}_{k+1}^- = \Phi(\hat{x}_k^+, u_k, \Delta t_k),$$

using the same nonlinear dynamics routine as in the truth model. Because $u_k = 0$ for this problem, only gravity and process noise drive the motion.

Covariance prediction. To propagate the covariance, we linearize the dynamics about $\hat{x}_k^+$,

$$A_k = \left. \frac{\partial f}{\partial x} \right|_{x=\hat{x}_k^+},$$

and approximate the discrete-time state transition and noise-influence matrices as

$$\tilde{F}_k \approx I + \Delta t_k A_k, \quad \tilde{\Omega}_k \approx \Delta t_k \Gamma,$$

where

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

maps the white acceleration disturbances in the X and Y directions into the state. The predicted covariance is then

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

3.3.3 Measurement update

At the next measurement time, the filter uses whichever ground stations report measurements. Because the propagated nonlinear state will not match the true state exactly, the set of stations that are actually in view in the noisy truth simulation may differ slightly from what would be predicted from the EKF state alone. In the implementation, we therefore always stack the measurements from the station IDs that actually report at time $k + 1$, and construct the corresponding stacked measurement vector and Jacobian using those same station IDs.

Each visible station contributes a range, range-rate, and elevation angle measurement. These are stacked into a single measurement vector, and R_{k+1} is built by repeating R_{true} along the block diagonal.

Predicted measurement. Given $\hat{x}_{k+1}^-$ and the list of visible stations at time $k + 1$, the EKF computes the predicted measurement

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

and the stacked measurement Jacobian

$$\tilde{H}_{k+1} = \left. \frac{\partial h}{\partial x} \right|_{x=\hat{x}_{k+1}^-}.$$

Innovation and gain. The stacked innovation vector and its covariance are

$$\tilde{v}_{k+1} = y_{k+1} - \hat{y}_{k+1}^-, \quad S_{k+1} = \tilde{H}_{k+1} P_{k+1}^- \tilde{H}_{k+1}^T + R_{k+1}.$$

Because the elevation angle is wrapped to the interval $[-\pi, \pi]$ in the truth model, the corresponding entries of $\tilde{v}_{k+1}$ are also wrapped to $[-\pi, \pi]$ before computing S_{k+1} and the Kalman gain, to avoid large jumps when the angle crosses $\pm\pi$.

The Kalman gain is then

$$K_{k+1} = P_{k+1}^- \tilde{H}_{k+1}^T S_{k+1}^{-1}.$$

Posterior update. Finally, we update the state and covariance at $k + 1$,

$$\hat{x}_{k+1}^+ = \hat{x}_{k+1}^- + K_{k+1} \tilde{v}_{k+1},$$

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

This completes one EKF cycle, which is repeated for all $K = 1400$ steps with sampling time $\Delta t = 10$ s.

3.3.4 Results

a. Typical single-run results

Before performing the full NEES and NIS consistency tests, we examine a single representative EKF run driven by a nonlinear truth simulation with process noise Q_{true} and measurement noise R_{true} . Figures 18–20 summarize the behaviour of the filter for this run.

Figure 18 shows the true noisy state trajectory (black) and the EKF posterior mean (blue) for all four states, along with the corresponding $\pm 2\sigma$ bounds derived from the diagonal of P_k^+ (red dashed curves). The EKF starts from a noticeably biased initial condition, but within the first few hundred seconds the estimated orbit locks onto the true trajectory and the covariance collapses to realistic values. Over the remainder of the trajectory, the estimated states track the truth almost exactly, and the true state stays well inside the $\pm 2\sigma$ envelopes.

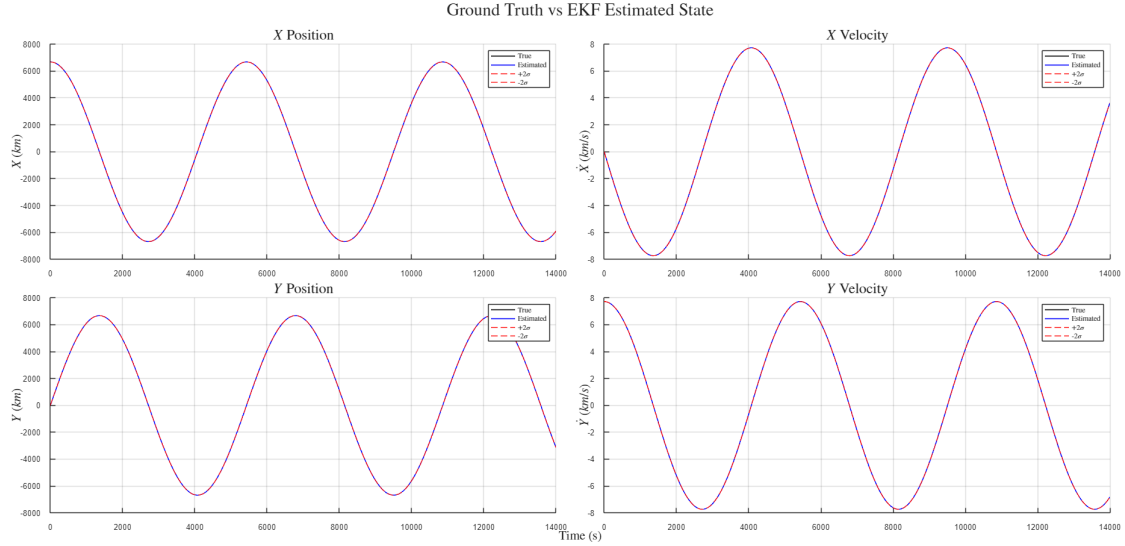


Figure 18: Ground truth vs. EKF estimated states with $\pm 2\sigma$ covariance bounds for all four state components.

To better assess the estimation accuracy, Figure 19 shows the state estimation error $e_x(k) = \hat{x}_k^+ - x_{\text{true}}(k)$ together with the corresponding $\pm 2\sigma$ bounds. The position errors remain on the order of one meter, while the velocity errors are on the order of 10^{-3} km/s, and the errors are generally well contained within the $\pm 2\sigma$ envelopes. There is no obvious long-term bias or drift, which is consistent with a well-tuned filter and a reasonable choice of Q_{KF} .

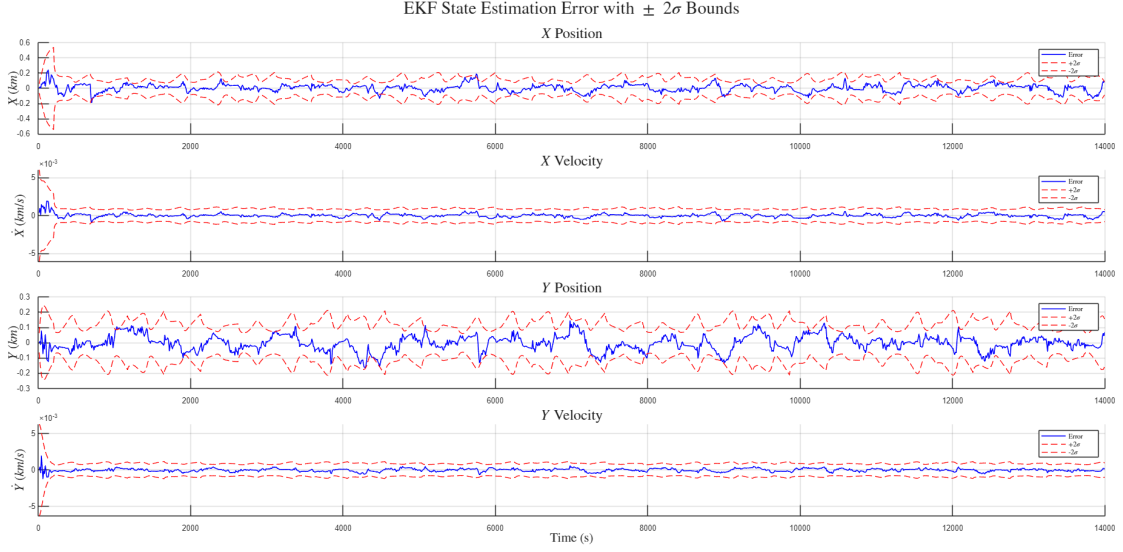


Figure 19: Error in EKF estimated states with $\pm 2\sigma$ covariance bounds.

Figure 20 compares the noisy simulated measurements from the nonlinear truth model (blue points) with the EKF predicted measurements $h(\hat{x}_k^-)$ (red points) for range, range rate, and elevation angle. The EKF predictions closely track the noisy measurements during every station pass, including gaps when no stations are visible, indicating that the filter is correctly fusing information from the switching measurement set. The predicted measurements appears to properly filter the noise from the provided measurements.

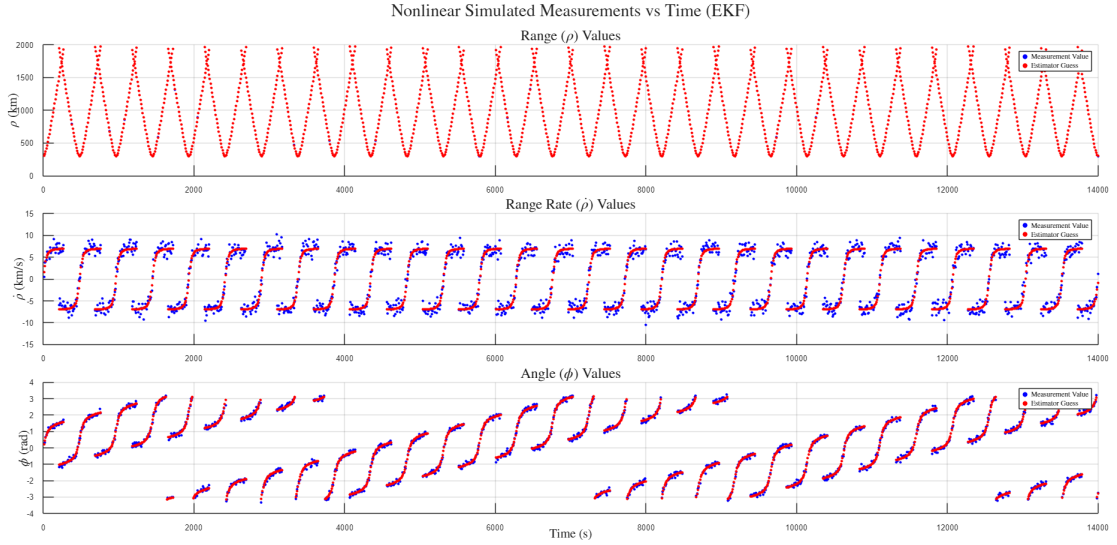


Figure 20: Nonlinear simulated measurements vs. time (blue) and EKF predicted measurements (red) for range, range rate, and elevation angle.

b. NEES test statistic points

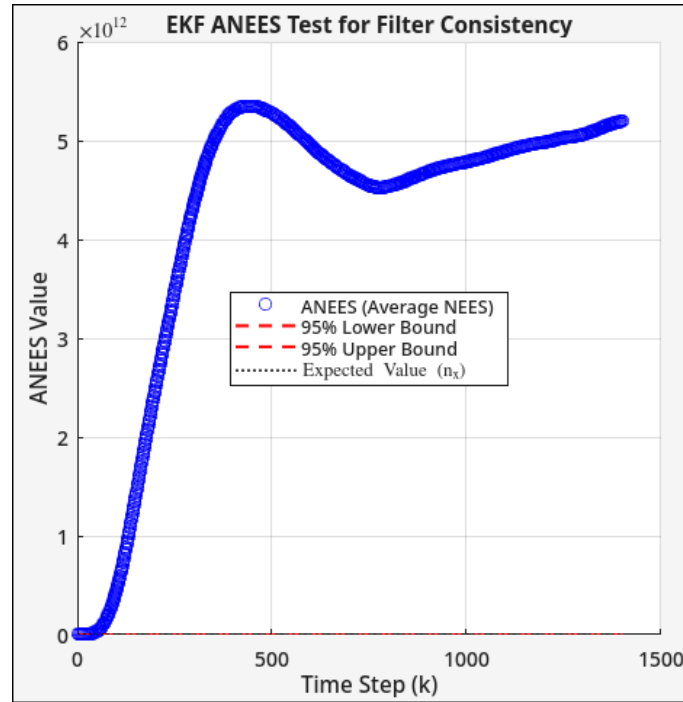


Figure 21: Initial result of NEES plot for EKF

These are the initial results for the NEES statistics of the Extended Kalman Filter. As one can clearly see, the values grow massively out of bounds, which indicates that a lot of tuning needs to be done.

After tuning and tweaking the code, the final NEES plot for the EKF looks like this:

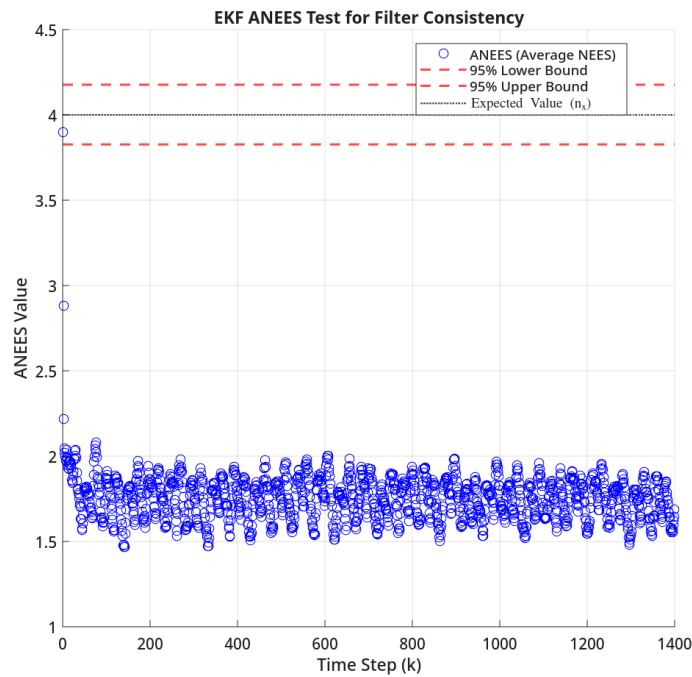


Figure 22: Final NEES plot for EKF

Figure 22 clearly shows that while the spread of the error is largely correct, we still arrived at a

much lower average (about 1.5) than expected ($4=n$). However, compared to LKF, the error stays largely the same during the whole simulation of 14,000s and doesn't increase exponentially.

c. NIS test statistic points

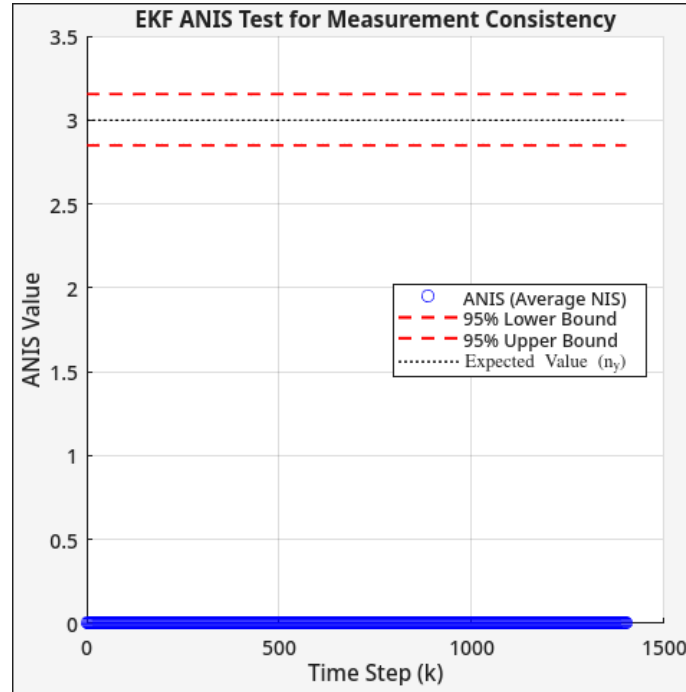


Figure 23: Initial result of NIS plot for EKF

These are the initial results for the NIS statistics of the Extended Kalman Filter. While the results for NIS are very different than with the LKF, additional work is needed to find out why ANIS seems to stay at or around zero.

After finding the error in our code and tuning the system, the final NIS plot for the EKF looks very promising:

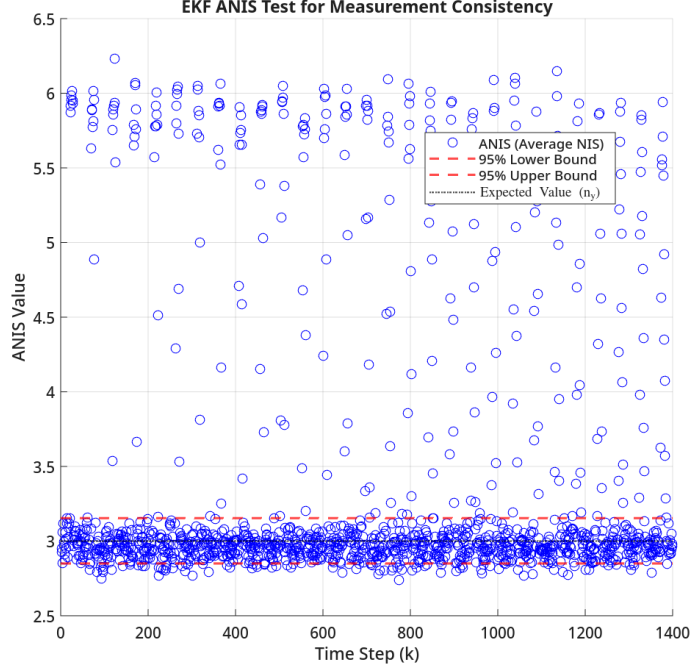


Figure 24: Final NIS plot for EKF

Figure 24 clearly shows that while there are some outliers, we largely stay within the bounds, showing **high measurement consistency** for the whole simulation of 14000s and doesn't increase exponentially or fluctuate.

3.4 State trajectory estimation and comparison

For Part 6 we applied both the linearized Kalman filter (LKF) and the Extended Kalman Filter (EKF) to the provided observation log `orbitdeterm_finalproj_KFdata.mat`. Both filters use the same sampling time $\Delta T = 10$ s, the same initial covariance

$$P_0 = \text{diag}(\sigma_{x0}^2, \sigma_{v0}^2, \sigma_{x0}^2, \sigma_{v0}^2) = \text{diag}(1 \cdot 10^{-6}, 1 \cdot 10^{-5}, 1 \cdot 10^{-6}, 1 \cdot 10^{-6}),$$

and the same process and measurement noise settings. The process noise covariance used in all runs is

$$Q_{KF} = 1 \cdot 10^{-10} \cdot I_2 = \begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{-10} \end{bmatrix},$$

The measurement noise covariance is taken directly from the data file, $R_{KF} = R_{\text{true}}$.

Figure 25 shows the LKF estimated states with $\pm 2\sigma$ bounds computed from the filter covariance. The position components X and Y follow the expected sinusoidal orbit and the uncertainty stays small compared to the orbital radius. However, in the velocity plots ($\dot{X}$ and $\dot{Y}$) the LKF develops noticeable small-scale oscillations toward the end of the time window. These are expected and seem to be a result of the linear model struggling with the nonlinear dynamics.

Figure 26 shows the same plots for the EKF with the same Q_{KF} and R_{KF} . Here the position and velocity trajectories are smooth over the entire 14,000 s window, and the $\pm 2\sigma$ envelopes remain narrow. Because the EKF repeatedly relinearizes about its current state estimate instead of a fixed nominal orbit, it handles the nonlinear two-body dynamics better and does not see the deviation as the time step increases like the LKF does.

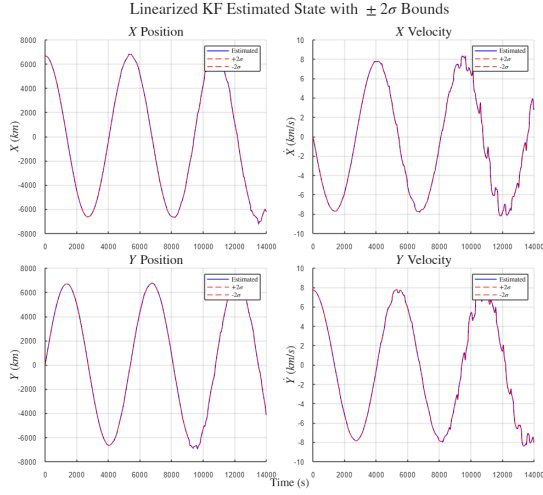


Figure 25: Linearized KF estimated states with $\pm 2\sigma$ bounds.

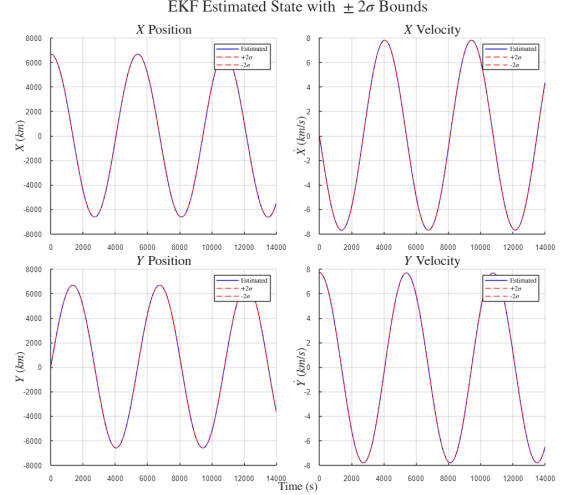


Figure 26: EKF estimated states with $\pm 2\sigma$ bounds.

To compare the filters directly, Figure 27 overlays the LKF and EKF state estimates. The position estimates from the two filters lie on top of each other which confirms that the estimated parameters are correct. Since we do not have the true state for this data log, we cannot compute an actual error, but it would be expected for a physically reasonable orbit to not have these rapid jumps in velocity. This would strongly suggest that the EKF is doing a better job of tracking the underlying motion with the same noise assumptions.

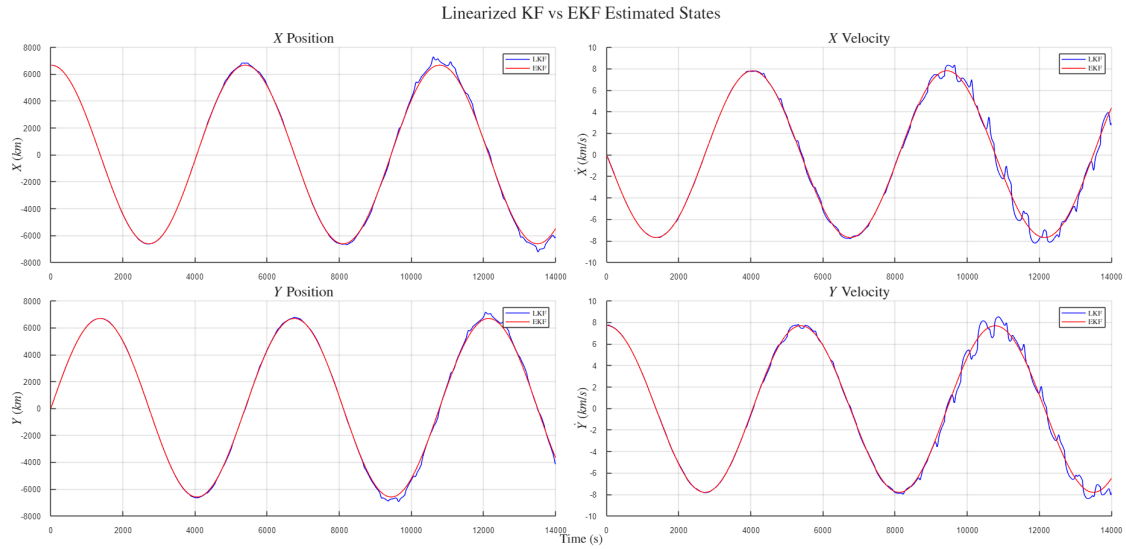


Figure 27: Direct comparison of Linearized KF (blue) and EKF (red) estimated states.

Finally, Figure 28 compares the logged measurements to the EKF predicted measurements for range, range rate, and elevation angle. The red EKF predictions lie almost exactly on top of the blue measurement points for every station pass, including the gaps when no station is visible. This suggests that the EKF state estimate is consistent with the actual data stream and that the chosen Q_{KF} and R_{KF} values are reasonable for this problem.

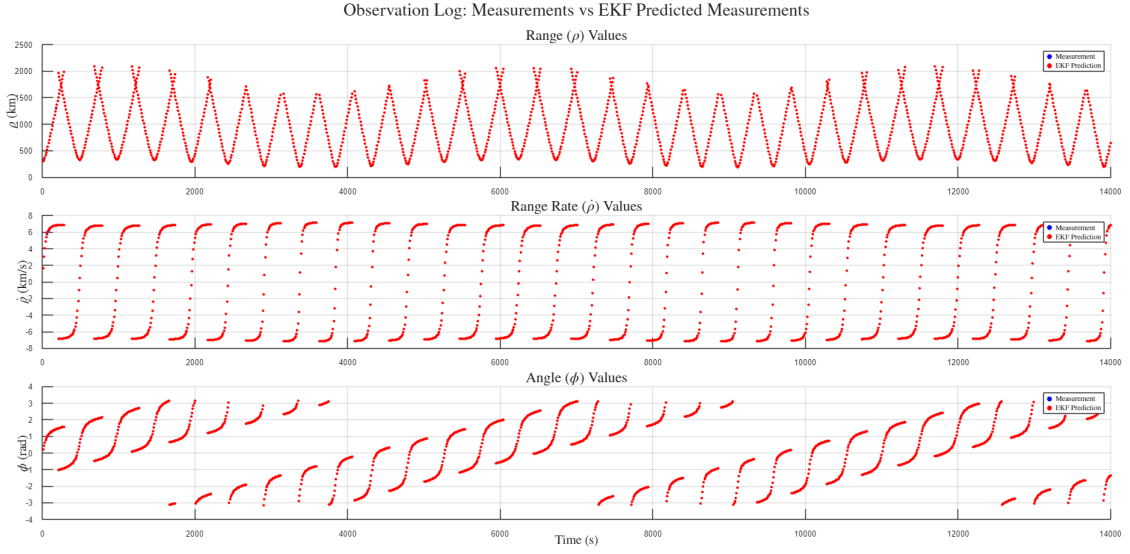


Figure 28: Observation log measurements (blue) versus EKF predicted measurements (red).

Overall, both filters give usable state estimates on the observation log, but the EKF does a better job. With the same initial conditions and noise settings it produces smoother, more realistic states while matching the measurements. The LKF starts to show artificial oscillatory motion because it is tied to a fixed linearization about the nominal orbit, whereas the EKF continually updates its linearization around the current estimate and better captures the nonlinear orbital dynamics.

4 Estimation Haiku - Advanced Question 13

*Noisy data in
Kalman learns and updates it
Corrected path out*

List of Figures

1	Vehicle position visualization.	7
2	DT Simulation States and Nominal States	10
3	DT Linearized Simulated State Perturbations	10
4	DT Linearized Simulated Measurements	11
5	States vs. Time, Nonlinear Dynamics Simulation	11
6	Nonlinear Simulated Measurements vs. Time	12
7	DT Linearized vs. Nonlinear Simulation States	13
8	DT Linearized State Error vs. Nonlinear Simulation States	13
9	Typical Simulated Measurements	18
10	Discrete LKF State Estimate Compared with Simulated Measurement	18
11	Linearized Kalman Filter Example Run Error	19
12	Initial result of NEES plot for LKF	19
13	Final ANEES plot for LKF (full simulation)	20
14	Final ANEES plot for LKF (zoomed into first 8500s)	20
15	Initial result of NIS plot for LKF	21
16	Final ANIS plot for LKF (full simulation)	21
17	Final ANIS plot for LKF (zoomed into first 8500s)	21
18	Ground truth vs. EKF estimated states with $\pm 2\sigma$ covariance bounds for all four state components.	24
19	Error in EKF estimated states with $\pm 2\sigma$ covariance bounds.	25
20	Nonlinear simulated measurements vs. time (blue) and EKF predicted measurements (red) for range, range rate, and elevation angle.	25
21	Initial result of NEES plot for EKF	26
22	Final NEES plot for EKF	26
23	Initial result of NIS plot for EKF	27
24	Final NIS plot for EKF	28
25	Linearized KF estimated states with $\pm 2\sigma$ bounds.	29
26	EKF estimated states with $\pm 2\sigma$ bounds.	29
27	Direct comparison of Linearized KF (blue) and EKF (red) estimated states.	29
28	Observation log measurements (blue) versus EKF predicted measurements (red).	30

5 Appendix

5.1 Appendix A: Equation Derivation

For Earth-Centered Coordinates:

$$\mathbf{x}(t) = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \Rightarrow \dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{y} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} f_1(x) \\ f_2(x) \\ f_3(x) \\ f_4(x) \end{bmatrix} = \underbrace{\begin{bmatrix} x_2 \\ -\frac{\mu x_1}{r^3} + u_1 + \ddot{w}_1 \\ x_4 \\ -\frac{\mu x_3}{r^3} + u_2 + \ddot{w}_2 \end{bmatrix}}_{f(x)} \quad \text{where } r = \sqrt{x_1^2 + x_3^2}$$

Find Dynamics Jacobian $\bar{A}(x)$:

$$A(x) = \frac{\partial f(x)}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \frac{\partial f_1}{\partial x_4} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \frac{\partial f_2}{\partial x_4} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \frac{\partial f_3}{\partial x_4} \\ \frac{\partial f_4}{\partial x_1} & \frac{\partial f_4}{\partial x_2} & \frac{\partial f_4}{\partial x_3} & \frac{\partial f_4}{\partial x_4} \end{bmatrix}$$

Easy ones:

$$f_1(x) = x_2 \Rightarrow \frac{\partial f_1}{\partial x} = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$$

$$f_3(x) = x_4 \Rightarrow \frac{\partial f_3}{\partial x} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$

$f_2(x)$ Partial Derivatives:

$$f_2(x) = -\frac{\mu x_1}{r^3} + u_1 + \ddot{w}_1 \quad \text{where } r = \sqrt{x_1^2 + x_3^2}$$

$$\begin{aligned} \frac{\partial f_2}{\partial x_1} &= \frac{\partial}{\partial x_1} \left(-\mu x_1 (x_1^2 + x_3^2)^{-3/2} + \ddot{w}_1 \right) \\ &= -\mu \left[(x_1^2 + x_3^2)^{-3/2} + x_1 \cdot (-3/2) \cdot (x_1^2 + x_3^2)^{-5/2} \cdot 2x_1 \right] \quad (\text{Product Rule}) \\ &= -\mu \left[(x_1^2 + x_3^2)^{-3/2} - 3x_1^2 (x_1^2 + x_3^2)^{-5/2} \right] \\ &= -\mu (x_1^2 + x_3^2)^{-5/2} [(x_1^2 + x_3^2) - 3x_1^2] \\ &= -\mu (x_1^2 + x_3^2)^{-5/2} (x_3^2 - 2x_1^2) \\ \frac{\partial f_2}{\partial x_1} &= \frac{\mu (2x_1^2 - x_3^2)}{r^5} \\ \frac{\partial f_2}{\partial x_2} &= 0 \quad (\text{trivial}) \end{aligned}$$

$$\begin{aligned} \frac{\partial f_2}{\partial x_3} &= \frac{\partial}{\partial x_3} \left(-\mu x_1 (x_1^2 + x_3^2)^{-3/2} + \ddot{w}_1 \right) \\ &= -\mu x_1 \cdot (-3/2) \cdot (x_1^2 + x_3^2)^{-5/2} \cdot 2x_3 \\ &= 3\mu x_1 x_3 (x_1^2 + x_3^2)^{-5/2} \\ \frac{\partial f_2}{\partial x_3} &= \frac{3\mu x_1 x_3}{r^5} \end{aligned}$$

$$\frac{\partial f_2}{\partial x_1} = \frac{\mu (2x_1^2 - x_3^2)}{r^5}$$

$$\frac{\partial f_2}{\partial x_2} = 0 \quad (\text{trivial})$$

$$\frac{\partial f_2}{\partial x_4} = 0 \quad (\text{trivial})$$

$$\frac{\partial f_2}{\partial x} = \begin{bmatrix} \frac{\mu (2x_1^2 - x_3^2)}{r^5} & 0 & \frac{3\mu x_1 x_3}{r^5} & 0 \end{bmatrix}$$

$f_4(x)$ Partial Derivatives:

As $f_2 = f_4$ where $x_1 \leftrightarrow x_3$, we can swap x_1 and x_3 from the result of $\partial f_2 / \partial x$ to solve for $\partial f_4 / \partial x$.

$$\frac{\partial f_4}{\partial x} = \begin{bmatrix} \frac{3\mu x_1 x_3}{r^5} & 0 & \frac{\mu (2x_3^2 - x_1^2)}{r^5} & 0 \end{bmatrix}$$

$$\therefore \bar{A}(x) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{\mu (2x_1^2 - x_3^2)}{r^5} & 0 & \frac{3\mu x_1 x_3}{r^5} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{3\mu x_1 x_3}{r^5} & 0 & \frac{\mu (2x_3^2 - x_1^2)}{r^5} & 0 \end{bmatrix}$$

Find Input Jacobian $\bar{B}(x)$:

$$B(x) = \frac{\partial f}{\partial u} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} \\ \frac{\partial f_3}{\partial u_1} & \frac{\partial f_3}{\partial u_2} \\ \frac{\partial f_4}{\partial u_1} & \frac{\partial f_4}{\partial u_2} \end{bmatrix} \quad \begin{aligned} f_1(x) &= x_2 \\ f_2(x) &= -\frac{\mu x_1}{r^3} + u_1 + \tilde{w}_1 \\ f_3(x) &= x_4 \\ f_4(x) &= -\frac{\mu x_2}{r^3} + u_2 + \tilde{w}_2 \end{aligned} \quad \text{where } r = \sqrt{x_1^2 + x_2^2}$$

$$\bar{B}(x) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

Find Disturbance Jacobian $\bar{\Gamma}(x)$:

$$\Gamma(x) = \frac{\partial f}{\partial \tilde{w}} = \begin{bmatrix} \frac{\partial f_1}{\partial \tilde{w}_1} & \frac{\partial f_1}{\partial \tilde{w}_2} \\ \frac{\partial f_2}{\partial \tilde{w}_1} & \frac{\partial f_2}{\partial \tilde{w}_2} \\ \frac{\partial f_3}{\partial \tilde{w}_1} & \frac{\partial f_3}{\partial \tilde{w}_2} \\ \frac{\partial f_4}{\partial \tilde{w}_1} & \frac{\partial f_4}{\partial \tilde{w}_2} \end{bmatrix} \quad \begin{aligned} f_1(x) &= x_2 \\ f_2(x) &= -\frac{\mu x_1}{r^3} + u_1 + \tilde{w}_1 \\ f_3(x) &= x_4 \\ f_4(x) &= -\frac{\mu x_2}{r^3} + u_2 + \tilde{w}_2 \end{aligned} \quad \text{where } r = \sqrt{x_1^2 + x_2^2}$$

$$\bar{\Gamma}(x) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

Find Measurement Jacobian $\bar{H}(x) = \bar{C}(x)$:

$$y^i(k) = \begin{bmatrix} \rho^i(k) \\ \hat{\theta}^i(k) \\ \phi^i(k) \end{bmatrix} + \tilde{v}^i(k)$$

where: $\rho^i(k) = \sqrt{[x(k) - x_s^i(k)]^2 + [y(k) - y_s^i(k)]^2}$

$$\hat{\theta}^i(k) = \frac{[x(k) - x_s^i(k)] \cdot [\dot{x}(k) - \dot{x}_s^i(k)] + [y(k) - y_s^i(k)] \cdot [\dot{y}(k) - \dot{y}_s^i(k)]}{\rho^i(k)}$$

$$\phi^i(k) = \tan^{-1} \left(\frac{y(k) - y_s^i(k)}{x(k) - x_s^i(k)} \right)$$

$$\bar{H}(x) = \frac{\partial H}{\partial x} = \begin{bmatrix} \frac{\partial \rho^i}{\partial x_1} & \frac{\partial \rho^i}{\partial x_2} & \frac{\partial \rho^i}{\partial x_3} & \frac{\partial \rho^i}{\partial x_4} \\ \frac{\partial \hat{\theta}^i}{\partial x_1} & \frac{\partial \hat{\theta}^i}{\partial x_2} & \frac{\partial \hat{\theta}^i}{\partial x_3} & \frac{\partial \hat{\theta}^i}{\partial x_4} \\ \frac{\partial \phi^i}{\partial x_1} & \frac{\partial \phi^i}{\partial x_2} & \frac{\partial \phi^i}{\partial x_3} & \frac{\partial \phi^i}{\partial x_4} \end{bmatrix}$$

Find Range Jacobian $\frac{\partial \rho^i}{\partial x}$: where $\rho^i = \sqrt{[x_1(k) - x_s^i(k)]^2 + [x_3 - y_s^i(k)]^2}$

$$\begin{aligned} \frac{\partial \rho^i}{\partial x_1} &= \frac{1}{2} \left([x_1 - x_s^i]^2 + [x_3 - y_s^i]^2 \right)^{-1/2} \cdot \frac{\partial}{\partial x_1} \left([x_1(k) - x_s^i(k)]^2 + [x_3 - y_s^i(k)]^2 \right) \\ &= \frac{2[x_1(k) - x_s^i(k)]}{2 \sqrt{[x_1(k) - x_s^i(k)]^2 + [x_3 - y_s^i(k)]^2}} \\ \frac{\partial \rho^i}{\partial x_1} &= \frac{x_1(k) - x_s^i(k)}{\sqrt{[x_1(k) - x_s^i(k)]^2 + [x_3 - y_s^i(k)]^2}} \end{aligned}$$

As ρ^i is symmetrical about $x_1 \leftrightarrow x_3$:
 $x_s^i \leftrightarrow y_s^i$

$$\frac{\partial \rho^i}{\partial x_3} = \frac{x_3(k) - y_s^i(k)}{\sqrt{[x_1(k) - x_s^i(k)]^2 + [x_3 - y_s^i(k)]^2}}$$

$$\frac{\partial \rho^i}{\partial x_2} = \frac{\partial \rho^i}{\partial x_4} = 0 \quad (\text{inval as } \rho^i = f(x_1, x_3) \notin x_2, x_4)$$

$$\frac{\partial \rho^i}{\partial x} = \begin{bmatrix} \frac{x_1 - x_s^i}{\rho^i}, 0, \frac{x_3 - y_s^i}{\rho^i}, 0 \end{bmatrix}$$

Find Range Rate Jacobian $\frac{\partial \dot{\rho}_i}{\partial \mathbf{x}}$:

$$\dot{\rho}_i(t) = \frac{[x(t) - x_s^i(t)] \cdot [\dot{x}(t) - \dot{x}_s^i(t)] + [y(t) - y_s^i(t)] \cdot [\dot{y}(t) - \dot{y}_s^i(t)]}{\sqrt{[x(t) - x_s^i(t)]^2 + [y(t) - y_s^i(t)]^2}} = \frac{[x_1 - x_s^i(t)] \cdot [\dot{x}_1 - \dot{x}_s^i(t)] + [x_2 - x_s^i(t)] \cdot [\dot{x}_2 - \dot{x}_s^i(t)] + [x_3 - y_s^i(t)] \cdot [\dot{x}_3 - \dot{y}_s^i(t)]}{\sqrt{[x_1 - x_s^i(t)]^2 + [x_2 - y_s^i(t)]^2}} = \frac{a}{b}$$

designate to make cleaner

Find Partial Derivatives of numerator ($\frac{\partial b}{\partial x_i}$):

$$\bullet x_1) \quad \frac{\partial}{\partial x_1} ([x(t) - x_s^i(t)] \cdot [\dot{x}(t) - \dot{x}_s^i(t)] + [y(t) - y_s^i(t)] \cdot [\dot{y}(t) - \dot{y}_s^i(t)]) = \dot{x}(t) - \dot{x}_s^i(t)$$

$$\bullet x_2) \quad \frac{\partial}{\partial x_2} ([x(t) - x_s^i(t)] \cdot [\dot{x}(t) - \dot{x}_s^i(t)] + [y(t) - y_s^i(t)] \cdot [\dot{y}(t) - \dot{y}_s^i(t)]) = x(t) - x_s^i(t)$$

$$\bullet x_3) \quad \frac{\partial}{\partial x_3} ([x(t) - x_s^i(t)] \cdot [\dot{x}(t) - \dot{x}_s^i(t)] + [y(t) - y_s^i(t)] \cdot [\dot{y}(t) - \dot{y}_s^i(t)]) = \dot{y}(t) - \dot{y}_s^i(t)$$

$$\bullet x_4) \quad \frac{\partial}{\partial x_4} ([x(t) - x_s^i(t)] \cdot [\dot{x}(t) - \dot{x}_s^i(t)] + [y(t) - y_s^i(t)] \cdot [\dot{y}(t) - \dot{y}_s^i(t)]) = y(t) - y_s^i(t)$$

Find Partial Derivatives of denominator ($\frac{\partial b}{\partial x_i}$) (found previously):

$$\frac{\partial b}{\partial x_1} = \frac{\partial \rho_i}{\partial x_1} = \frac{[x_1 - x_s^i]}{\rho_i} \quad \frac{\partial b}{\partial x_2} = \frac{\partial \rho_i}{\partial x_2} = 0 \quad \frac{\partial b}{\partial x_3} = \frac{\partial \rho_i}{\partial x_3} = \frac{x_3 - y_s^i}{\rho_i} \quad \frac{\partial b}{\partial x_4} = \frac{\partial \rho_i}{\partial x_4} = 0$$

Find $\frac{\partial \dot{\rho}_i}{\partial x_1}$:

$$\frac{\partial \dot{\rho}_i(t)}{\partial x_1} = \frac{b \cdot \frac{\partial a}{\partial x_1} - a \cdot \frac{\partial b}{\partial x_1}}{b^2} = \frac{\rho_i \cdot [\dot{x}(t) - \dot{x}_s^i(t)] - a \cdot \frac{[x_1 - x_s^i]}{\rho_i}}{\rho_i^2} = \frac{[\dot{x}(t) - \dot{x}_s^i(t)]}{\rho_i} - \frac{a \cdot [x(t) - x_s^i(t)]}{\rho_i^3}$$

$$\frac{\partial \dot{\rho}_i(t)}{\partial x_1} = \frac{[x(t) - x_s^i(t)]}{\rho_i} - \frac{a \cdot [x(t) - x_s^i(t)]}{\rho_i^3}$$

Find $\frac{\partial \dot{\rho}_i}{\partial x_2}$:

$$\frac{\partial \dot{\rho}_i(t)}{\partial x_2} = \frac{b \cdot \frac{\partial a}{\partial x_2} - a \cdot \frac{\partial b}{\partial x_2}}{b^2} = \frac{\rho_i \cdot [x(t) - x_s^i(t)] - a \cdot [0]}{\rho_i^2}$$

$$\frac{\partial \dot{\rho}_i(t)}{\partial x_2} = \frac{x(t) - x_s^i(t)}{\rho_i}$$

Find $\frac{\partial \dot{\rho}_i}{\partial x_3}$:

$$\frac{\partial \dot{\rho}_i(t)}{\partial x_3} = \frac{b \cdot \frac{\partial a}{\partial x_3} - a \cdot \frac{\partial b}{\partial x_3}}{b^2} = \frac{\rho_i \cdot [x(t) - \dot{y}_s^i(t)] - a \cdot \frac{x_3 - y_s^i}{\rho_i}}{\rho_i^2} = \frac{[x(t) - \dot{y}_s^i(t)]}{\rho_i} - \frac{a \cdot [x_3 - y_s^i]}{\rho_i^3}$$

$$\frac{\partial \dot{\rho}_i(t)}{\partial x_3} = \frac{[x(t) - \dot{y}_s^i(t)]}{\rho_i} - \frac{a \cdot [x_3 - y_s^i]}{\rho_i^3}$$

Find $\frac{\partial \dot{\rho}_i}{\partial x_4}$:

$$\frac{\partial \dot{\rho}_i(t)}{\partial x_4} = \frac{b \cdot \frac{\partial a}{\partial x_4} - a \cdot \frac{\partial b}{\partial x_4}}{b^2} = \frac{\rho_i \cdot [y(t) - y_s^i(t)] - a \cdot (0)}{\rho_i^2}$$

$$\frac{\partial \dot{\rho}_i(t)}{\partial x_4} = \frac{y(t) - y_s^i(t)}{\rho_i}$$

$$\frac{\partial \dot{\rho}_i}{\partial \mathbf{x}} = \left[\frac{[x(t) - \dot{x}_s^i(t)]}{\rho_i} - \frac{a \cdot [x(t) - x_s^i(t)]}{\rho_i^3}, \frac{x(t) - x_s^i(t)}{\rho_i}, \frac{[x(t) - \dot{y}_s^i(t)]}{\rho_i} - \frac{a \cdot [x_3 - y_s^i]}{\rho_i^3}, \frac{y(t) - y_s^i(t)}{\rho_i} \right]$$

$$\text{where } a = [x_1 - x_s^i(t)] \cdot [x_2 - \dot{x}_s^i(t)] + [x_3 - y_s^i(t)] \cdot [x_4 - \dot{y}_s^i(t)]$$

Find Elevation Angle Jacobian $\frac{d\phi_i}{dx}$:

$$\phi_i = \tan^{-1} \left(\frac{x_3(t) - y_3^i(t)}{x_1(t) - x_5^i(t)} \right)$$

As $\phi_i = f(x_1, x_3)$ and $\notin \{x_2, x_4\}$:

$$\frac{d\phi_i}{dx_2} = 0$$

$$\frac{d\phi_i}{dx_4} = 0$$

Find $\frac{d\phi_i}{dx_1}$:

$$\begin{aligned} \frac{d\phi_i}{dx_1} &= \frac{1}{1 + \left(\frac{x_3(t) - y_3^i(t)}{x_1(t) - x_5^i(t)} \right)^2} \cdot \frac{d}{dx_1} \left(\frac{x_3(t) - y_3^i(t)}{x_1(t) - x_5^i(t)} \right) \\ &= \frac{1}{1 + \left(\frac{x_3(t) - y_3^i(t)}{x_1(t) - x_5^i(t)} \right)^2} \cdot \left[\frac{\frac{d}{dt}(x_3(t) - y_3^i(t)) \cdot (x_1(t) - x_5^i(t)) - (x_3(t) - y_3^i(t)) \cdot \frac{d}{dt}(x_1(t) - x_5^i(t))}{(x_1(t) - x_5^i(t))^2} \right] \\ &= \frac{1}{1 + \left(\frac{x_3(t) - y_3^i(t)}{x_1(t) - x_5^i(t)} \right)^2} \cdot \frac{- (x_3(t) - y_3^i(t))}{(x_1(t) - x_5^i(t))^2} \\ &= \frac{- (x_3(t) - y_3^i(t))}{(x_1(t) - x_5^i(t))^2 + (x_3(t) - y_3^i(t))^2} \\ &= \frac{- (x_3(t) - y_3^i(t))}{\rho_i^2} \end{aligned}$$

Find $\frac{d\phi_i}{dx_3}$:

$$\begin{aligned} \frac{d\phi_i}{dx_3} &= \frac{1}{1 + \left(\frac{x_3(t) - y_3^i(t)}{x_1(t) - x_5^i(t)} \right)^2} \cdot \frac{d}{dx_3} \left(\frac{x_3(t) - y_3^i(t)}{x_1(t) - x_5^i(t)} \right) \\ &= \frac{1}{1 + \left(\frac{x_3(t) - y_3^i(t)}{x_1(t) - x_5^i(t)} \right)^2} \cdot \frac{1}{x_1(t) - x_5^i(t)} \\ &= \frac{1}{1 + \left(\frac{x_3(t) - y_3^i(t)}{x_1(t) - x_5^i(t)} \right)^2} \cdot \frac{x_1(t) - x_5^i(t)}{(x_1(t) - x_5^i(t))^2} \\ &= \frac{x_1(t) - x_5^i(t)}{(x_1(t) - x_5^i(t))^2 + (x_3(t) - y_3^i(t))^2} \\ &= \frac{x_1(t) - x_5^i(t)}{\rho_i^2} \end{aligned}$$

$$\frac{d\phi_i}{dx} = \begin{bmatrix} \frac{- (x_3(t) - y_3^i(t))}{\rho_i^2} & 0 & \frac{x_1(t) - x_5^i(t)}{\rho_i^2} & 0 \end{bmatrix}$$

$$C(x) = \bar{H}(x) = \frac{dH}{dx} = \begin{bmatrix} \frac{\partial p_1}{\partial x_1} & \frac{\partial p_1}{\partial x_2} & \frac{\partial p_1}{\partial x_3} & \frac{\partial p_1}{\partial x_4} \\ \frac{\partial p_2}{\partial x_1} & \frac{\partial p_2}{\partial x_2} & \frac{\partial p_2}{\partial x_3} & \frac{\partial p_2}{\partial x_4} \\ \frac{\partial p_3}{\partial x_1} & \frac{\partial p_3}{\partial x_2} & \frac{\partial p_3}{\partial x_3} & \frac{\partial p_3}{\partial x_4} \\ \frac{\partial p_4}{\partial x_1} & \frac{\partial p_4}{\partial x_2} & \frac{\partial p_4}{\partial x_3} & \frac{\partial p_4}{\partial x_4} \end{bmatrix}$$

$$H(x) = \begin{bmatrix} \frac{\Delta x_i}{\rho_i} & 0 & \frac{\Delta y_i}{\rho_i} & 0 \\ \frac{\Delta \dot{x}_i}{\rho_i} - \frac{a \Delta x_i}{\rho_i^3} & \frac{\Delta x_i}{\rho_i} & \frac{\Delta \dot{y}_i}{\rho_i} - \frac{a \Delta y_i}{\rho_i^3} & \frac{\Delta y_i}{\rho_i} \\ -\frac{\Delta y_i}{\rho_i^3} & 0 & \frac{\Delta x_i}{\rho_i^3} & 0 \end{bmatrix}$$

where:

$$\begin{aligned} \Delta x_i &= x - x_s^i(t) = x_i - x_s^i(t) \\ \Delta \dot{x}_i &= \dot{x} - \dot{x}_s^i(t) = \dot{x}_i - \dot{x}_s^i(t) \\ \Delta y_i &= y - y_s^i(t) = x_i - y_s^i(t) \\ \Delta \dot{y}_i &= \dot{y} - \dot{y}_s^i(t) = \dot{x}_i - \dot{y}_s^i(t) \\ a &= \Delta x_i \Delta \dot{x}_i + \Delta x_i \Delta \dot{y}_i \\ \rho &= \sqrt{\Delta x_i^2 + \Delta y_i^2} \end{aligned}$$

Continuous Time:

$$\dot{\bar{x}} = \bar{A}|_{x_{nom}} \cdot \bar{x}(t) + \bar{B}|_{x_{nom}} \cdot \bar{u}(t) + \bar{\Gamma}|_{x_{nom}} \cdot \bar{\omega}(t)$$

$$\bar{y} = \bar{C}|_{x_{nom}} \cdot \bar{x}(t) + \bar{D}|_{x_{nom}} \cdot \bar{u}(t) + \bar{v}(t)$$

where:

$$\bar{A}(x) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{\mu(2x_1^2 - x_3^2)}{r^5} & 0 & \frac{3\mu x_1 x_3}{r^5} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{3\mu x_1 x_3}{r^5} & 0 & \frac{\mu(2x_3^2 - x_1^2)}{r^5} & 0 \end{bmatrix}$$

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

$$\bar{B}(x) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \quad \bar{\Gamma}(x) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \quad \bar{D}(x) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\bar{C}(x) = \bar{H}(x) = \begin{bmatrix} \frac{\Delta x_i}{\rho_i} & 0 & \frac{\Delta y_i}{\rho_i} & 0 \\ \frac{\Delta \dot{x}_i}{\rho_i} - \frac{a \Delta x_i}{\rho_i^3} & \frac{\Delta x_i}{\rho_i} & \frac{\Delta \dot{y}_i}{\rho_i} - \frac{a \Delta y_i}{\rho_i^3} & \frac{\Delta y_i}{\rho_i} \\ -\frac{\Delta y_i}{\rho_i^3} & 0 & \frac{\Delta x_i}{\rho_i^3} & 0 \end{bmatrix}$$

where:

$$\begin{aligned} \Delta x_i &= x - x_s^i(t) = x_i - x_s^i(t) \\ \Delta \dot{x}_i &= \dot{x} - \dot{x}_s^i(t) = \dot{x}_i - \dot{x}_s^i(t) \\ \Delta y_i &= y - y_s^i(t) = x_i - y_s^i(t) \\ \Delta \dot{y}_i &= \dot{y} - \dot{y}_s^i(t) = \dot{x}_i - \dot{y}_s^i(t) \\ a &= \Delta x_i \Delta \dot{x}_i + \Delta x_i \Delta \dot{y}_i \\ \rho &= \sqrt{\Delta x_i^2 + \Delta y_i^2} \end{aligned}$$

Discrete Time (Euler Approach)

Given the following CT system:

$$\delta \dot{x}(t) = A_k \delta x(t) + B_k \delta u(t) + \Gamma_k \tilde{\omega}(t)$$

We can approximate the derivative with:

$$\begin{aligned} \frac{\delta x_{k+1} - \delta x_k}{\Delta T} &\approx \delta \dot{x}(t) = A_k \delta x_k + B_k \delta u_k + \Gamma_k \tilde{\omega}_k \\ \Downarrow \\ \delta x_{k+1} &= \underbrace{\delta x_k + \Delta T \cdot A_k \cdot \delta x_k}_{\substack{(\mathbf{I} + \Delta T \cdot \mathbf{A}_k) \cdot \delta x_k \\ \tilde{\mathbf{F}}_k}} + \underbrace{\Delta T \cdot B_k \cdot \delta u_k}_{\tilde{\mathbf{G}}_k} + \underbrace{\Delta T \cdot \Gamma_k \cdot \tilde{\omega}_k}_{\tilde{\mathbf{d}}_k} \end{aligned}$$

Therefore:

$$\delta x_{k+1} = \tilde{\mathbf{F}}_k \cdot \delta x_k + \tilde{\mathbf{G}}_k \cdot \delta u_k + \tilde{\mathbf{d}}_k \omega_k$$

$$y_k = \hat{\mathbf{H}}_k \cdot \delta x_k$$

where:

$$\tilde{\mathbf{F}}_k = \left(\mathbf{I} + \Delta T \cdot \begin{bmatrix} 0 & 1 \\ \frac{\mu(2x_1^2 - x_2^2)}{r^5} & 0 \\ 0 & \frac{3\mu x_1 x_2}{r^5} \\ \frac{3\mu x_1 x_2}{r^5} & 0 \\ 0 & \frac{\mu(2x_2^2 - x_1^2)}{r^5} \end{bmatrix} \right)$$

where $r = \sqrt{x_1^2 + x_2^2}$

$$\tilde{\mathbf{G}}_k = \Delta T \cdot \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad \tilde{\mathbf{d}}_k = \Delta T \cdot \Gamma_k$$

$$\bar{\mathbf{H}}(x) = \begin{bmatrix} \frac{\Delta x_i}{\rho_i} & 0 & \frac{\Delta y_i}{\rho_i} & 0 \\ \frac{\Delta \dot{x}_i}{\rho_i} - \frac{a \Delta x_i}{\rho_i^3} & \frac{\Delta x_i}{\rho_i} & \frac{\Delta \dot{y}_i}{\rho_i} - \frac{a \Delta y_i}{\rho_i^3} & \frac{\Delta y_i}{\rho_i} \\ -\frac{\Delta y_i}{\rho_i^2} & 0 & \frac{\Delta x_i}{\rho_i^2} & 0 \end{bmatrix}$$

where:

$$\begin{aligned} \Delta x_i &= x - x_b^i(t) = x_1 - x_b^i(t) \\ \Delta \dot{x}_i &= \dot{x} - \dot{x}_b^i(t) = \dot{x}_1 - \dot{x}_b^i(t) \\ \Delta y_i &= y - y_b^i(t) = x_2 - y_b^i(t) \\ \Delta \dot{y}_i &= \dot{y} - \dot{y}_b^i(t) = \dot{x}_2 - \dot{y}_b^i(t) \\ a &= \Delta y_i \Delta \dot{y}_i + \Delta x_i \Delta \dot{x}_i \\ \rho &= \sqrt{\Delta x^2 + \Delta y^2} \end{aligned}$$

5.2 Appendix B: MATLAB Code

File: A_jacobian.m

```

1 function A = A_jacobian(x, mu)
2 %A_JACOBIAN Continuous-time dynamics Jacobian A(x) = df/dx.
3 %Inputs:
4 % x - 4x1 state [X; Xdot; Y; Ydot]
5 % mu - gravitational parameter
6
7 %Output:
8 % A - 4x4 Jacobian matrix evaluated at x
9
10 %Pull out and calculate commonly used variables
11 x1 = x(1);
12 x3 = x(3);
13 r = sqrt(x1^2 + x3^2);
14 r5 = r^5;
15
16 %Calculate Matrix Inputs
17 A_11 = mu * (2 * x1^2 - x3^2) / r5;
18 A_13 = 3 * mu * x1 * x3 / r5;
19 A_31 = 3 * mu * x1 * x3 / r5;
20 A_33 = mu * (2 * x3^2 - x1^2) / r5;
21
22 %Form Output Matrix
23 A = [0    1    0    0;
24      A_11 0    A_13 0;
25      0    0    0    1;
26      A_31 0    A_33 0];
27 end

```

Listing 1: Code for A_jacobian.m

File: DTLinearizedKF.m

```

1 function [P_plus, x_hat_plus, P_minus, x_hat_minus, e_y, S] = DTLinearizedKF(P_0, mu_0, N, Dt, Q, R, ←
    ydata,params,t_0)
2     mu = params.mu;
3     P_plus = P_0;
4     x_hat_plus = mu_0';
5     dx_hat_plus = mu_0' - getXNom(t_0)';
6     P_minus = [];
7     x_hat_minus = [];
8     dx_hat_minus = [];
9     t = zeros(N + 1);
10    t(1) = t_0;
11    n = size(mu_0,2);
12
13    ydata = cellfun(@(ydata_k) reshape(ydata_k,4,[]), ydata, 'UniformOutput',false);
14    y = cellfun(@(ydata_k) ydata_k(1:3,:), ydata, 'UniformOutput',false);
15    visible_stations = cellfun(@(ydata_k) ydata_k(4,:),ydata, 'UniformOutput',false);
16    % Clean visible_stations by removing NaN entries
17    visible_stations = cellfun(@(x) x(~isnan(x)), visible_stations, 'UniformOutput', false);
18
19    e_y = cell(1, N); % Innovation vector dy_k at each step
20    S = cell(1, N); % Innovation covariance matrix S_k at each step
21    for k = 1:N
22        i = k + 1;
23        i_prev = i - 1;
24        t_k_prev = t(i-1);
25        t_k = t_k_prev + Dt;
26        t(i) = t_k;
27        %% Eulerized DT Jacobians
28        x_nom_k_prev = getXNom(t_k_prev)';
29        [~, ~, ~, ~, Gamma_k, F_k, G_k, ~, ~] = getLinearizedMatrices(x_nom_k_prev,t_k,Dt,1,mu);
30        Omega_k = Dt*Gamma_k;
31        %getDTLinearizedMatrices()
32        %% Prediction Step

```

```

33 %
34 % <include>kalmanFilterPrediction.m</include>
35 %
36 P_plus_k_prev = P_plus(:, :, i_prev);
37 dx_hat_plus_k_prev = dx_hat_plus(:, i_prev);
38 u = zeros(2, 1); % TODO: Do we need u data?
39 [dx_hat_minus_k, P_minus_k] = DTLinearizedKFPrediction(...
40     dx_hat_plus_k_prev, P_plus_k_prev, ...
41     F_k, G_k, Omega_k, Q, u);
42 P_minus(:, :, k) = P_minus_k;
43 dx_hat_minus(:, k) = dx_hat_minus_k;
44 x_nom_k = getXNom(t_k)';
45 x_hat_minus_k = dx_hat_minus_k + x_nom_k;
46 %% Correction Step
47 H_k = [];
48 visible_stations_k = visible_stations{k};
49 visible_stations_nom_k = [];
50 y_nom_k = [];
51 dy_k = [];
52 y_k = y{k};
53 y_k = reshape(y_k, 3, []);
54 N_visible_stations_k = size(visible_stations_k, 2);
55 for i_meas = 1:N_visible_stations_k
56     y_k_i = y_k(:, i_meas);
57     stationID = visible_stations_k(i_meas);
58     if (stationID > 0)
59         [~, ~, ~, ~, ~, ~, H_k_i, ~] = getLinearizedMatrices(x_nom_k, t_k, Dt, stationID, mu);
60         [y_nom_k_i, ~] = GetStationMeasurement(x_nom_k, t_k, stationID);
61
62     else
63         % TODO: Check if this is correct handling for when no station
64         % sees the spacecraft
65         H_k_i = [];
66         y_nom_k_i = zeros(3, 1);
67         y_k_i = zeros(3, 1);
68     end
69     % If we have a measurement, assume it's visible.
70
71     dy_k_i = y_k_i - y_nom_k_i;
72     %% Correct angle perturbation for quadrant
73     phi_k_i = y_k_i(3);
74     phi_k_norm_i = y_nom_k_i(3);
75     dphi_k_i = phi_k_i - phi_k_norm_i;
76     % Bound the perturbation to [-pi/2
77     if dphi_k_i > pi
78         dphi_k_i = dphi_k_i - 2*pi;
79     end
80     if dphi_k_i < -pi
81         dphi_k_i = dphi_k_i + 2*pi;
82     end
83     dy_k_i(3) = dphi_k_i;
84     dy_k = [dy_k; dy_k_i];
85     y_nom_k = [y_nom_k; y_nom_k_i];
86     H_k = [H_k; H_k_i];
87 end
88 [p_k, ~] = size(H_k);
89 % TODO: track measurements by station...
90 measurementNotAvailable = (p_k == 0);
91 if measurementNotAvailable % measurements not available
92     dx_hat_plus_k = dx_hat_minus_k;
93     P_plus_k = P_minus_k;
94     e_y_k = {};
95     S_k = {};
96 else % measurements available
97     R_diag = mat2cell(repmat(R, 1, N_visible_stations_k), 3, 3*ones(1, N_visible_stations_k));
98     R_k = blkdiag(R_diag{:});
99
100 [e_y_k, dx_hat_plus_k, P_plus_k, S_k, ~] = DTLinearizedKFCorrection( ...
101     dx_hat_minus_k, P_minus_k, dy_k, ...

```

```

102     H_k,R_k);
103 end
104 e_y{k} = e_y_k; % Store current innovation vector
105 S{k} = S_k; % Store current innovation covariance matrix
106 P_plus(:, :, i) = P_plus_k;
107 dx_hat_plus(:, i) = dx_hat_plus_k;
108 x_hat_plus_k = dx_hat_plus_k + x_nom_k;
109 x_hat_plus(:, i) = x_hat_plus_k;
110
111 end
112 x_hat_plus = x_hat_plus';
113 x_hat_minus = x_hat_minus';
114 end

```

Listing 2: Code for DTLinearizedKF.m

File: DTLinearizedKFCorrection.m

```

1 function [e_y_k, x_hat_plus_k, P_plus_k, S_k, y_hat_minus_k] = DTLinearizedKFCorrection(...
2     x_hat_minus_k, P_minus_k, y_k, ...
3     H_k, R)
4 y_hat_minus_k = H_k*x_hat_minus_k;
5 e_y_k = y_k - y_hat_minus_k;
6
7 N_visible_stations_k = numel(y_k)/3;
8 for i_phi_k = 3*(1:N_visible_stations_k)
9     e_phi_k = e_y_k(i_phi_k);
10    % Bound the perturbation to [-pi/2
11    if e_phi_k > pi
12        e_phi_k = e_phi_k - 2*pi;
13    end
14    if e_phi_k < -pi
15        e_phi_k = e_phi_k + 2*pi;
16    end
17    e_y_k(i_phi_k) = e_phi_k;
18 end
19 S_k = H_k*P_minus_k*H_k' + R;
20 K_k = (P_minus_k*H_k')/S_k;
21 x_hat_plus_k = x_hat_minus_k + K_k*(e_y_k);
22 [n, ~] = size(x_hat_minus_k);
23 P_plus_k = (eye(n) - K_k*H_k)*P_minus_k*(eye(n) - K_k*H_k)' + K_k*R*K_k';
24
25 end

```

Listing 3: Code for DTLinearizedKFCorrection.m

File: DTLinearizedKFPrediction.m

```

1 function [x_hat_minus_k, P_minus_k] = DTLinearizedKFPrediction(...
2     x_hat_plus_k_prev, P_plus_k_prev, ...
3     F_k, G_k, Omega_k, Q_k, u_k)
4
5 x_hat_minus_k = F_k*x_hat_plus_k_prev + G_k*u_k;
6 P_minus_k = F_k*P_plus_k_prev*F_k' + Omega_k*Q_k*Omega_k';
7 end

```

Listing 4: Code for DTLinearizedKFPrediction.m

File: DTMeasurementSimulationStep.m

```

1 function [y_k] = DTMeasurementSimulationStep(x_k_prev, y_k_prev, ...
2     H, R)
3 if y_k == []
4 end

```

Listing 5: Code for DTMeasurementSimulationStep.m

File: DTSimulation.m

```

1 function outputs = DTSimulation(Q,R,dx_0,T,Dt,params)
2     %% Config
3     mu = params.mu;
4     N_stations = 12;
5     % N_stations = 2;
6     N = T/Dt;
7     t = 0:Dt:T;
8     N_x = numel(t);
9     % N_y = N_x - 1;
10    N_y = N_x;
11    n = numel(dx_0);
12    x_nom_0 = getXNom(t(1))';
13    x_0 = dx_0 + x_nom_0;
14    outputs.dx = zeros(N_x,n);
15    outputs.x = zeros(N_x,n);
16    outputs.x_nom = zeros(N_x,n);
17    outputs.y = cell(N_y,1);
18    outputs.dy = cell(N_y,1);
19    outputs.H = cell(N_y,1);
20    outputs.t = t;
21    outputs.dx(1,:) = dx_0;
22    outputs.x(1,:) = x_0;
23    outputs.x_nom(1,:) = x_nom_0;
24    outputs.visible_stations = cell(N_y,1);
25    %% State Simulation
26    for k = 1:N
27        %% Get Parameters
28        i = k + 1; % x(k = 0) == x(i = 1), x(k = 1) == x(i = 2), ...
29        % i_x = k + 1; % x(k = 0) == x(i = 0), x(k = 0
30        % i_y = k; % y(k = 1) == y(i = 1), ..
31        dx_k_prev = outputs.dx(i-1,:)' ;
32        t_k = t(i);
33        x_nom_k_prev = outputs.x_nom(i-1,:)' ;
34        Q_k = Q; % TODO: Update
35        R_k = R; % TODO: Update
36        %% State Simulation Step
37        [~, ~, ~, ~, ~, F_k, ~, ~, ~] = getLinearizedMatrices(x_nom_k_prev,t_k,Dt,1,mu);
38        % Process Noise
39        if all(Q_k == 0)
40            w_k = zeros(size(dx_k_prev));
41        else
42            S_w_k = chol(Q_k,'lower'); % cholesky decomposition of Q_k
43            q_w = randn(n,1); % random number
44            w_k = S_w_k*q_w; % process noise
45        end
46        dx_k = F_k*dx_k_prev + w_k; % propagate state x(k-1) -> x(k)
47        x_nom_k = getXNom(t_k)';
48        x_k = dx_k + x_nom_k;
49        %% Measurement Simulation Step
50        H_k = [];
51        visible_stations_k = [];
52        visible_stations_nom_k = [];
53        y_nom_k = [];
54        for stationID = 1:N_stations
55            [~, ~, ~, ~, ~, ~, ~, H_k_i, ~] = getLinearizedMatrices(x_nom_k_prev,t_k,Dt,stationID,mu);
56            [y_nom_k_i, ~] = GetStationMeasurement(x_nom_k, t_k, stationID);
57            [~, is_visible] = GetStationMeasurement(x_k, t_k, stationID);
58            if is_visible
59                y_nom_k = [y_nom_k; y_nom_k_i];
60                visible_stations_k = [visible_stations_k stationID];
61                H_k = [H_k; H_k_i];
62            end
63        end
64        [p_k,~ ] = size(H_k);
65        measurementNotAvailable = (p_k ~= numel(y_nom_k)) || (p_k == 0);
66        if measurementNotAvailable % measurements not available

```

```

67     y_k = [];
68     else % measurements available
69         mu_dy_k = H_k*dx_k;
70         if all(Q_k == 0)
71             v_k = zeros(size(mu_dy_k));
72         else
73             [p,~] = size(H_k);
74             S_v = chol(R_k,'lower'); % cholesky decomposition of R_k
75             q_v = randn(p,1); % random number
76             v_k = S_v*q_v; % measurement noise
77         end
78         dy_k = mu_dy_k + v_k;
79         y_k = y_nom_k + dy_k;
80     end
81     %% Store Updated Parameters
82     outputs.dx(i,:) = dx_k';
83     outputs.x_nom(i,:) = x_nom_k';
84     outputs.x(i,:) = x_k';
85     outputs.y_nom{i} = y_nom_k';
86     outputs.dy{i} = dy_k';
87     outputs.y{i} = y_k';
88     outputs.H{i} = H_k;
89     outputs.visible_stations{i} = visible_stations_k;
90     end
91 end

```

Listing 6: Code for DTSimulation.m

File: EKF_Orbit.m

```

1 function [xhat_hist, P_hist, innov_cell, S_cell, visible_ids_cell, yhat_cell] = EKF_Orbit(ydata, tvec, ←
    x0_hat, P0, Q_matrix, R_matrix, mu, R_E, omega_E)
2 %Runs an Extended Kalman Filter for a single measurement sequence
3 %
4 %Inputs
5 % ydata - 1xN cell array of measurements. ydata{i} = m_i x 1 measurement
6 %         associated with the measurement at time tvec(i). m_i = 3 * (# visible stations)
7 % tvec - 1xN double time vector associated with the time of each measurement [seconds]
8 % x0_hat - 4x1 initial state estimate
9 % P0 - 4x4 initial covariance
10 % Qtrue - 2x2 process noise covariance matrix applied to acceleration disturbances
11 % Rtrue - 3x3 measurement noise covariance matrix for each station
12 % mu - Earth gravitational parameter [km^3/s^2]
13 % R_E - Earth radius [km]
14 % omega_E - Earth rotation rate [rad/s]
15 %
16 %Outputs
17 % xhat_hist - 4xN double matrix of posterior state estimates
18 % P_hist - 4x4xN double matrix of posterior covariance matrices
19 % innov_cell - 1xN cell array of innovation vectors v_k = y_k - yhat_k
20 % S_cell - 1xN cell array of innovation covariance matrices S_k
21 % visible_ids_cell - 1xN cell array of visible station indices at each time step
22 % yhat_cell - 1xN cell array of predicted measurement vectors h(xhat_minus)
23
24 %Get measurement length
25 N = length(tvec);
26 numStates = 4;
27
28 %Preallocate output variables
29 xhat_hist = zeros(numStates,N);
30 P_hist = zeros(numStates,numStates,N);
31 innov_cell = cell(1,N);
32 S_cell = cell(1,N);
33 visible_ids_cell = cell(1,N);
34 yhat_cell = cell(1,N);
35
36 %Initialize state estimate and covariance for time tvec(1)

```



```

37 xhat_plus = x0_hat;
38 P_plus = P0;
39
40 %Add to output arrays
41 xhat_hist(:,1) = xhat_plus;
42 P_hist(:,1) = P_plus;
43 innov_cell{1} = [];
44 S_cell{1} = [];
45 visible_ids_cell{1} = [];
46 yhat_cell{1} = [];
47
48 %Define Gamma (constant)
49 gamma = [0 0;
50          1 0;
51          0 0;
52          0 1];
53
54 %Assume input zero at all times
55 u_k = [0; 0];
56
57 %Extended Kalman Filter loop
58 for k = 1:(N-1)
59     %Time step
60     dT = tvec(k+1) - tvec(k);
61
62     % Now propagate using nonlinear dynamics
63     xhat_minus = propagate_orbit(xhat_plus, u_k, dT, mu);
64
65     %Time update / prediction
66     A_tilde_k = A_jacobian(xhat_plus, mu);
67     F_tilde_k = eye(numStates) + dT*A_tilde_k;
68     Omega_tilde_k = dT * gamma;
69
70     P_minus = F_tilde_k*P_plus*F_tilde_k' + Omega_tilde_k * Q_matrix * Omega_tilde_k';
71
72     %Measurement data at time t_{k+1}
73     measurement = ydata{k+1};
74
75     %If no measurements, prediction only
76     if isempty(measurement)
77         xhat_plus = xhat_minus;
78         P_plus = P_minus;
79         xhat_hist(:,k+1) = xhat_plus;
80         P_hist(:,k+1) = P_plus;
81         innov_cell{k+1} = [];
82         S_cell{k+1} = [];
83         visible_ids_cell{k+1} = [];
84         yhat_cell{k+1} = [];
85         continue
86     end
87
88     %Extract station IDs and number of visible stations
89     station_ids = measurement(4,:);
90     n_vis = numel(station_ids);
91
92     %Build measurement vector y_meas by stacking [rho; rho_dot; angle] columns
93     y_meas = reshape(measurement(1:3,:),3*n_vis,1);
94
95     %Measurement update / correction step
96     t_k1 = tvec(k+1);
97     [yhat_minus, H_tilde_k, ~] = H_jacobian(xhat_minus, t_k1, station_ids, R_E, omega_E);
98
99     %Force predicted measurement to column
100    yhat_minus = yhat_minus(:);
101
102    %Store predicted measurement and station IDs
103    yhat_cell{k+1} = yhat_minus;
104    visible_ids_cell{k+1} = station_ids;
105

```

```

106 %Form measurement noise covariance for n_vis stations
107 R_k = R_matrix;
108 for j = 2:n_vis
109     R_k = blkdiag(R_k,R_matrix);
110 end
111
112 if(k == 21 || k == 22)
113     abc = 1;
114 end
115
116 %Innovation
117 e_tilde_k = y_meas - yhat_minus;
118
119 %Wrap innovations to [-pi, pi]
120 for j = 1:n_vis
121     angle_idx = 3*j; % Every 3rd element is an angle
122     e_tilde_k(angle_idx) = wrapToPi(e_tilde_k(angle_idx));
123 end
124
125 %Innovation covariance
126 S_k = H_tilde_k*P_minus*H_tilde_k' + R_k;
127
128 %Kalman gain
129 K_tilde_k = P_minus*H_tilde_k'/S_k;
130
131 %State update
132 xhat_plus = xhat_minus + K_tilde_k*e_tilde_k;
133
134 %Covariance update
135 P_plus = (eye(numStates) - K_tilde_k*H_tilde_k)*P_minus;
136
137 %Store results
138 xhat_hist(:,k+1) = xhat_plus;
139 P_hist(:,:,k+1) = P_plus;
140 innov_cell{k+1} = e_tilde_k;
141 S_cell{k+1} = S_k;
142 end
143
144 end

```

Listing 7: Code for EKF_Orbit.m

File: Final_Project_Main.m

```

1 %% Part 1 Problem A, B
2 Final_Project_Part_1_ab
3 %% Part 1 Problem C
4 Final_Project_Part_1_c
5 % TODO: Simulation Comparison
6 %% Part II Problem 4 a
7 Final_Project_Part_2_Problem_4_part_a
8 %% Part II Problem 5 a
9 % STANDALONE
10 %Final_Project_Part_2_Problem_5_part_a
11 %% Part II Problem 4/5 b,c
12 % STANDALONE
13
14 %RunTMT_rmh
15 %% Part II Problem 6
16 Final_Project_Part_2_Problem_6

```

Listing 8: Code for Final_Project_Main.m

File: Final_Project_Part_1_ab.m

```

1 %clear; clc; close all;

```

```

2
3 %Load in constants
4 mu = 398600; %Gravitational Constant [km^3/s^2]
5 Re = 6378; %Earth radius [km]
6 omegaE = (2*pi) / 86400; %Earth rotational rate [rad/s]
7
8 %Nominal Orbital Parameters
9 r0 = Re + 300; %300 km orbit radius [km]
10 X0 = r0; %Initial X Position [km]
11 Y0 = 0; %Initial Y Position [km]
12 X0_dot = 0; %Initial X Velocity [km/s]
13 Y0_dot = r0 * sqrt(mu / r0^3); %Initial Y Velocity [km/s]
14
15 x_nom = [X0; X0_dot; Y0; Y0_dot]; %Nominal state (used as linearization point)
16
17 %Define Disturbances (process noise standard deviations)
18 sigma_w = [0; 0]; %[w_tilde_1; w_tilde_2]
19
20 %Nominal Input (No Forcing)
21 u_nom = [0; 0];
22
23 %Simulation Constants
24 delta_t = 10; %[seconds]
25 time = 0;
26 stationID = 1;
27
28 %Get CT and DT system matrices at linearization point
29 [A, B, C, D, Gamma, F, G, H, M] = getLinearizedMatrices(x_nom, time, delta_t, stationID, mu);

```

Listing 9: Code for Final_Project_Part_1_ab.m

File: Final_Project_Part_1_c.m

```

1 %% Simulation Config
2 % Dummy Variables % TODO: Remove
3 n = numel(x_nom);
4 p = 2; % TODO: Vary this.
5 Q = zeros(n);
6 R = zeros(p);
7 r = Re + 300;
8 T = 2*pi*sqrt(r^3/mu);
9 Dt = 10;
10 T = 1400*Dt;
11
12 params.mu = mu;
13 dX0 = 0.001*x_nom; % TODO: Good IC
14 dX0 = [0,0.075,0,-0.021];
15 %% DT Simulation
16 dt_LTI_outputs = DTSimulation(Q,R,dX0',T,delta_t,params);
17 %% Plot
18 %% DT Simulated States and Nominal States vs. Time
19 plotSimulationStates(dt_LTI_outputs.t,dt_LTI_outputs.x)
20 plotSimulationStates(dt_LTI_outputs.t,dt_LTI_outputs.x_nom)
21 sgtitle("DT Simulated States and Nominal States vs. Time")
22
23 %% Sanity Check on Visibility
24 figure; hold on;
25 plotSimulationStatesCartesian(dt_LTI_outputs.t,dt_LTI_outputs.x)
26 plotSimulationStatesCartesian(dt_LTI_outputs.t,dt_LTI_outputs.x_nom)
27 plot(Re*sin(0:0.01:(2*pi)),Re*cos(0:0.01:(2*pi)))
28 ifoo = 251;
29 for stationID = 1:12
30     [X_1_tfoo, Y_1_tfoo, ~, ~, ~] = GetGroundStationState(stationID, dt_LTI_outputs.t(ifoo));
31     scatter(X_1_tfoo,Y_1_tfoo,'filled','HandleVisibility',"off")
32     text(X_1_tfoo,Y_1_tfoo,num2str(stationID))
33 end
34 x_tfoo = dt_LTI_outputs.x(ifoo,:);

```

```

35 scatter(x_tfoo(1),x_tfoo(3))
36 sgtitle("DT Simulated States and Nominal States (Cartesian)")
37
38 %% DT Simulated State Perturbations vs. Time"
39 figure; hold on;
40 plotSimulationStates(dt_LTI_outputs.t,dt_LTI_outputs.dx)
41 sgtitle("DT Simulated State Perturbations vs. Time")
42
43 %% DT Simulated Measurements vs. Time
44 figure; hold on;
45 plotStationMeasurements(dt_LTI_outputs)
46 sgtitle("DT Simulated Measurements vs. Time")
47
48
49 %% Define variables and ICs
50 r_0 = sqrt(6678^2);
51 mu = 398600; % Gravitational parameter for Earth in km^3/s^2
52 x_IC = [6678; 0; 0; r_0 * sqrt(mu/r_0^3)];
53 x_0_perturb = [0; 0.075; 0; -0.021];
54 x_0 = x_IC + x_0_perturb;
55 Dt = 10;
56 T = 1400*10; % 1400 steps, each 10s
57
58 %% Simulate states
59 outputs = NonlinearSimulation(x_0, T, Dt);
60
61 %% Plot states over time
62 t = 0:Dt:T;
63 data_tip_indices = [1, 136, 402, 726, 1098, 1401];
64
65 figure;
66 sgtitle('States vs. Time, Nonlinear Dynamics simulation');
67 subplot(4,1,1);
68 p = plot(t, outputs.x(:,1));
69 xlabel('Time (s)');
70 ylabel('X (km)');
71 ylim([-1e4, 1e4]);
72
73 % Create the Data Tips
74 p.DataTipTemplate.DataTipRows(2).Format = '%.0f';
75 for i = 1:length(data_tip_indices)
76     datatip(p, 'DataIndex', data_tip_indices(i));
77 end
78
79 subplot(4,1,2);
80 p = plot(t, outputs.x(:,2));
81 xlabel('Time (s)');
82 ylabel('X_{dot} (km/s)');
83
84 % Create the Data Tips
85 p.DataTipTemplate.DataTipRows(2).Format = '%.3f';
86 for i = 1:length(data_tip_indices)
87     datatip(p, 'DataIndex', data_tip_indices(i));
88 end
89
90 subplot(4,1,3);
91 p = plot(t, outputs.x(:,3));
92 xlabel('Time (s)');
93 ylabel('Y (km)');
94 ylim([-1e4, 1e4]);
95
96 % Create the Data Tips
97 p.DataTipTemplate.DataTipRows(2).Format = '%.0f';
98 for i = 1:length(data_tip_indices)
99     datatip(p, 'DataIndex', data_tip_indices(i));
100 end
101
102 subplot(4,1,4);
103 p = plot(t, outputs.x(:,4));

```

```

104 xlabel('Time (s)');
105 ylabel('Y_{dot} (km/s)');
106
107 % Create the Data Tips
108 p.DataTipTemplate.DataTipRows(2).Format = '%.3f';
109 for i = 1:length(data_tip_indices)
110     datatip(p, 'DataIndex', data_tip_indices(i));
111 end
112
113 %% Plot Nonlinear Measurements vs. Time
114 outputs.t = t;
115 figure; hold on;
116 plotStationMeasurements(outputs)
117 sgtitle("Nonlinear Simulated Measurements vs. Time")
118
119 %% Sanity Check on Visibility
120 figure; hold on;
121 plotSimulationStatesCartesian(outputs.t,outputs.x)
122 plot(Re*sin(0:0.01:(2*pi)),Re*cos(0:0.01:(2*pi)))
123 ifoo = 1106;
124 for stationID = 1:12
125     [X_1_tfoo, Y_1_tfoo, ~, ~, ~] = GetGroundStationState(stationID, outputs.t(ifoo));
126     scatter(X_1_tfoo,Y_1_tfoo,'filled',"HandleVisibility","off")
127     text(X_1_tfoo,Y_1_tfoo,num2str(stationID))
128 end
129 x_tfoo = outputs.x(ifoo,:);
130 scatter(x_tfoo(1),x_tfoo(3))
131 text(x_tfoo(1),x_tfoo(3),['t = ' num2str(outputs.t(ifoo))])
132 sgtitle("DT Simulated States and Nominal States (Cartesian)")
133
134 %% Simulation Comparison
135
136 figure; hold on;
137 sgtitle("DT Linearized vs. Nonlinear Simulation States")
138 plotSimulationStates(dt_LTI_outputs.t,dt_LTI_outputs.x)
139 plotSimulationStates(t,outputs.x);
140 legend("DT Linearized","Nonlinear")
141 figure; hold on;
142 sgtitle("DT Linearized States error vs. Nonlinear Simulation States")
143 outputs.error = dt_LTI_outputs.x - outputs.x;
144 plotSimulationStates(outputs.t,outputs.error);

```

Listing 10: Code for Final_Project_Part_1_c.m

File: Final_Project_Part_2_Problem_4_part_a.m

```

1 %% TODO: A "typical" sim run.
2
3 mu_0 = getXNom(0);
4 N = 1400;
5 n = 4;
6 pos_variance = 1e-6;
7 vel_variance = 1e-5;
8 P_0 = diag([pos_variance, vel_variance, pos_variance, vel_variance]);
9 Q_KF = 1e-10*eye(2); % TODO: Use a proper guess
10 %Q = Q - 1e-12*ones(2) + 1e-12*eye(2);
11 params.mu = mu;
12 T = 1400*Dt;
13
14 load('orbitdeterm_finalproj_KFdata.mat')
15 % Euler Approximation for DT Process Noise Covariance
16 G = [0 0;
17     1 0;
18     0 0;
19     0 1];
20 % Q_DT = G * Qtrue * G' * Dt;
21 % epsilon = 1e-12;

```

```

22 % Rtrue = Rtrue/1e6
23 % % Apply jitter to process noise covariance
24 % Q_DT = Q_DT + epsilon * eye(n);
25 t_0 = 0;
26 S_w = chol(Qtrue, 'lower');
27 S_v = chol(Rtrue, 'lower');
28 dX0 = [0.0001,0.005,0.0001,-0.002];
29 x_0 = mu_0 + dX0;
30 outputs = NonlinearSimulationNoisy(x_0,t_0,1400*Dt,Dt,S_w,S_v);
31
32 ydata_simulated = convertOutput2Ydata(outputs);
33 %ydata_simulated = cell(1,N);
34 %Rtrue = Rtrue/1e6
35 %% Plot Nonlinear Measurements vs. Time
36 t = Dt:Dt:T;
37 outputs.t = t;
38 figure; hold on;
39 plotStationMeasurements(outputs)
40 sgtitle("Nonlinear Simulated Measurements vs. Time")
41
42 %% Run LKF
43 [P_plus, x_hat_plus, P_minus, x_hat_minus] = <-
    DTLinearizedKF(P_0,mu_0,N,Dt,Q_KF,Rtrue,ydata_simulated,params,t_0);
44
45 [sigma_x_hat_plus, ~] = getSigmas(n, N, P_plus, P_minus);
46 e_x_hat_plus = x_hat_plus(2:end,:) - outputs.x(2:end,:);
47
48
49 %% Plot
50
51 figure; hold on;
52 plotSimulationStatesCartesian(outputs.t,outputs.x)
53 plotSimulationStatesCartesian(t(2:end),x_hat_plus(2:end,:));
54 t = 0:Dt:T;
55 figure; hold on;
56 sgtitle("DT Linearized Kalman Filter");
57 plotSimulationStates(t(2:end),outputs.x(2:end,:));
58 plotSimulationStates(t(2:end),x_hat_plus(2:end,:));
59 plotSimulationStates(t(2:end),x_hat_plus(2:end,:)+sigma_x_hat_plus);
60 plotSimulationStates(t(2:end),x_hat_plus(2:end,:)-sigma_x_hat_plus);
61 for subplot_i = 1:n
62     subplot(n,1,subplot_i); hold on;
63     legend("$x$", "$+2\sigma$", "$+2\sigma$", "$-2\sigma$", "Location", "eastoutside", "Interpreter", 'latex');
64 end
65
66 figure; hold on;
67 sgtitle("DT Linearized Kalman Filter Error");
68 plotSimulationStates(t(2:end),e_x_hat_plus);
69 plotSimulationStates(t(2:end),2*sigma_x_hat_plus);
70 plotSimulationStates(t(2:end),-2*sigma_x_hat_plus);
71 for subplot_i = 1:n
72     subplot(n,1,subplot_i); hold on;
73     legend("e", "+2\sigma", "-2\sigma", "Location", "eastoutside");
74 end
75
76 %% Helper Functions
77 % TODO: Move to common?
78
79 function [sigma_x_hat_plus, sigma_x_hat_minus] = getSigmas(n, N, P_plus, P_minus)
80 sigma_2_x_hat_plus = zeros(n,N);
81 sigma_2_x_hat_minus = zeros(n,N);
82 for k = 1:N
83     sigma_2_x_hat_plus(:,k) = diag(P_plus(:,k));
84     sigma_2_x_hat_minus(:,k) = diag(P_minus(:,k));
85 end
86 sigma_x_hat_plus = sqrt(sigma_2_x_hat_plus)';
87 sigma_x_hat_minus = sqrt(sigma_2_x_hat_minus)';

```

88 || end

Listing 11: Code for Final_Project_Part_2_Problem_4_part_a.m

File: Final_Project_Part_2_Problem_5_part_a.m

```
1 %Clear workspace
2 clear; clc; close all;
3
4 %Load in Matlab data file
5 load('orbitdeterm_finalproj_KFdata.mat');
6
7 %Define orbital constants
8 mu      = 398600;      %km^3/s^2 (Earth gravitational parameter)
9 R_e     = 6378;       %km (Earth radius)
10 omega_E = (2*pi)/86400; %rad/s (Earth rotational rate)
11
12 dT      = 10;         %Sampling interval (seconds)
13 K       = 1400;       %Number of time steps
14 tvec    = 0:dT:K*dT;  %Data simulation time vector (length K+1)
15
16 %Define nominal initial state
17 r0      = 6678;       %300 km altitude orbit
18 x_nom0  = [r0; 0; 0; r0*sqrt(mu/r0^3)]; %[X_nom, X_dot_nom, Y_nom, Y_dot_nom]
19
20 %Define initial desired perturbation
21 desiredPerturbation = [0.0; 0.0; 0.0; 0.0]; %[delta_X, delta_X_dot, delta_Y, delta_Y_dot]
22 x_initial = x_nom0 + desiredPerturbation; %[X, X_dot, Y, Y_dot]
23
24 %Eulerized DT process-noise covariance
25 Gamma    = [0 0;
26             1 0;
27             0 0;
28             0 1];
29
30 %Apply first-order noise
31 Q_DT     = Gamma * Qtrue * Gamma' * dT;
32
33 %Apply jitter to process noise to ensure positive-definite matrix
34 Q_DT     = Q_DT + 1e-12 * eye(4);
35
36 %Find Cholesky factors for process / measurement noise
37 S_w      = chol(Qtrue, 'lower'); %Process noise (4x4)
38 % S_w     = chol(Qtrue, 'lower'); %Process noise (4x4)
39
40 S_v      = chol(Rtrue, 'lower'); %Single-station measurement noise (3x3)
41
42 %Simulate noisy nonlinear "truth" and convert to y-data format
43 sim_outputs = NonlinearSimulationNoisy(x_initial,0,K*dT,dT,S_w,S_v);
44 sim_yData   = convertOutput2Ydata(sim_outputs);
45
46 %Form yData cell array aligned with tvec (index 1 corresponds to t = 0)
47 yData      = cell(1,K+1);
48 yData{1}   = []; %No measurement at t = 0
49 yData(2:end) = sim_yData(:); %Measurements from t = dT to K*dT
50
51 %EKF initial conditions and noise guesses
52 x0_hat     = x_nom0;
53 sigma_x0   = 10000; %Position STD in km
54 sigma_v0   = 10; %Velocity STD in km/s
55 P_0        = diag([sigma_x0^2, sigma_v0^2, sigma_x0^2, sigma_v0^2]);
56 Q_KF       = 5*Qtrue; %Process-noise guess for EKF
57
58 %Run Extended Kalman Filter
59 [xhat_hist,P_hist,innov_cell,S_cell,visible_ids_cell,yhat_cell] = ...
60     EKF_Orbit(yData,tvec,x0_hat,P_0,Q_KF,Rtrue,mu,R_e,omega_E);
61
```

```

62 %Pre-compute true states and 1-sigma standard deviations
63 N = numel(tvec);
64 x_true = sim_outputs.x.'; %4xN
65 sigma_hist = zeros(4,N);
66 for k = 1:N
67     sigma_hist(:,k) = sqrt(diag(P_hist(:, :, k)));
68 end
69 state_labels = {'X (km)', 'X_dot (km/s)', 'Y (km)', 'Y_dot (km/s)'};
70
71 %% Plot 1: Ground truth vs EKF estimated state (with 2 bounds)
72 figure('Color','w');
73 tiles = tiledlayout('flow','TileSpacing','compact','Padding','compact');
74 title(tiles,'Ground Truth vs EKF Estimated State');
75
76 %X position
77 nexttile; hold on;
78 plot(tvec,x_true(1,:), 'k', 'LineWidth', 1.2);
79 plot(tvec,xhat_hist(1,:), 'b', 'LineWidth', 1.2);
80 plot(tvec,xhat_hist(1,:) + 2*sigma_hist(1,:), 'r--', 'LineWidth', 1);
81 plot(tvec,xhat_hist(1,:) - 2*sigma_hist(1,:), 'r--', 'LineWidth', 1);
82 ylabel(state_labels{1});
83 legend('True','Estimated','+2\sigma','-2\sigma','Location','best');
84 grid on;
85
86 %X velocity
87 nexttile; hold on;
88 plot(tvec,x_true(2,:), 'k', 'LineWidth', 1.2);
89 plot(tvec,xhat_hist(2,:), 'b', 'LineWidth', 1.2);
90 plot(tvec,xhat_hist(2,:) + 2*sigma_hist(2,:), 'r--', 'LineWidth', 1);
91 plot(tvec,xhat_hist(2,:) - 2*sigma_hist(2,:), 'r--', 'LineWidth', 1);
92 ylabel(state_labels{2});
93 legend('True','Estimated','+2\sigma','-2\sigma','Location','best');
94 grid on;
95
96 %Y position
97 nexttile; hold on;
98 plot(tvec,x_true(3,:), 'k', 'LineWidth', 1.2);
99 plot(tvec,xhat_hist(3,:), 'b', 'LineWidth', 1.2);
100 plot(tvec,xhat_hist(3,:) + 2*sigma_hist(3,:), 'r--', 'LineWidth', 1);
101 plot(tvec,xhat_hist(3,:) - 2*sigma_hist(3,:), 'r--', 'LineWidth', 1);
102 ylabel(state_labels{3});
103 legend('True','Estimated','+2\sigma','-2\sigma','Location','best');
104 grid on;
105
106 %Y velocity
107 nexttile; hold on;
108 plot(tvec,x_true(4,:), 'k', 'LineWidth', 1.2);
109 plot(tvec,xhat_hist(4,:), 'b', 'LineWidth', 1.2);
110 plot(tvec,xhat_hist(4,:) + 2*sigma_hist(4,:), 'r--', 'LineWidth', 1);
111 plot(tvec,xhat_hist(4,:) - 2*sigma_hist(4,:), 'r--', 'LineWidth', 1);
112 ylabel(state_labels{4});
113 xlabel('Time (s)');
114 legend('True','Estimated','+2\sigma','-2\sigma','Location','best');
115 grid on;
116
117 %Set up for additional comparison plots (matching Part 4 style)
118 n = 4;
119 Dt_loc = tvec(2) - tvec(1);
120 T = tvec(end);
121
122 %Use times k = 1..K (skip k = 0 to match LKF style)
123 t_plot = tvec(2:end); %1xK
124 x_true_plot = sim_outputs.x(2:end,:); %Kx4
125 xhat_plot = xhat_hist(:,2:end)'; %Kx4
126
127 %Posterior 1-sigma from P_hist for k = 1..K
128 K_steps = numel(t_plot);
129 sigma_x_hat_ekf = zeros(K_steps,n);
130 for k = 1:K_steps

```



```

131     sigma_x_hat_ekf(k,:) = sqrt(diag(P_hist(:,:,k+1)))';
132 end
133
134 %Estimation error (estimate - truth)
135 e_x_hat_ekf = xhat_plot - x_true_plot; %Kx4
136
137 %% Plot 2: EKF state estimation error with 2 bounds
138 figure('Color','w');
139 tiles4 = tiledlayout(4,1,'TileSpacing','compact','Padding','compact');
140 title(tiles4,'EKF State Estimation Error with  $\pm 2\sigma$  Bounds');
141
142 for i = 1:n
143     nexttile; hold on;
144     plot(t_plot,e_x_hat_ekf(:,i),'b','LineWidth',1.2);
145     plot(t_plot, 2*sigma_x_hat_ekf(:,i),'r--','LineWidth',1);
146     plot(t_plot,-2*sigma_x_hat_ekf(:,i),'r--','LineWidth',1);
147     ylabel(['e_{x_}',num2str(i),'']);
148     if i == n
149         xlabel('Time (s)');
150     end
151
152     % previousID = [];
153     % for j = 1:numel(innov_cell)
154     %     iterationValue = visible_ids_cell{j};
155     %
156     %     if((isempty(previousID) && ~isempty(iterationValue)) || any(previousID ~= iterationValue))
157     %         xline(t_plot(j));
158     %         previousID = iterationValue;
159     %     end
160     % end
161     %
162     % legend('Error','+2\sigma','-2\sigma', 'Location','best');
163     % grid on;
164
165
166
167 end
168
169
170 %% Plot 3: Nonlinear simulated measurements vs time (EKF case)
171 figure('Color','w');
172 tiles2 = tiledlayout(3,1,'TileSpacing','compact','Padding','compact');
173 title(tiles2,'Nonlinear Simulated Measurements vs Time (EKF)');
174
175 t_meas = [];
176 rho_meas = [];
177 rhod_meas = [];
178 phi_meas = [];
179
180 for k = 1:numel(yData)
181     yk = yData{k};
182     if isempty(yk)
183         continue;
184     end
185     n_vis = size(yk,2);
186     t_meas = [t_meas, repmat(tvec(k),1,n_vis)];
187     rho_meas = [rho_meas, yk(1,:)];
188     rhod_meas = [rhod_meas, yk(2,:)];
189     phi_meas = [phi_meas, yk(3,:)];
190 end
191
192 %Plot H_matrix Y Output values
193 range_vec = [];
194 range_rate_vec = [];
195 angle_vec = [];
196 time_vec = [];
197
198 times = [0 t_plot];
199

```

```

200 for i = 1:numel(yhat_cell)
201     disp(i);
202
203     iterationValue = yhat_cell{i};
204
205     range = []; range_rate = []; angle = [];
206
207     %Pull Out Value
208     if(~isempty(iterationValue))
209         iterationValue = reshape(iterationValue, 3, []);
210
211         range = iterationValue(1, :);
212         range_rate = iterationValue(2, :);
213         angle = iterationValue(3, :);
214     end
215
216     range_vec = [range_vec range(:)'];
217     range_rate_vec = [range_rate_vec range_rate(:)'];
218     angle_vec = [angle_vec angle(:)'];
219     time_vec = [time_vec times(i) * ones(1, numel(range))];
220 end
221
222
223 nexttile; hold on;
224 scatter(t_meas,rho_meas,10,'filled', 'MarkerFaceColor', [0, 0, 1]);
225 scatter(time_vec, range_vec, 10, 'filled', 'MarkerFaceColor',[1, 0, 0]);
226 ylabel('\rho (km)');
227 grid on;
228
229 nexttile; hold on;
230 scatter(t_meas,rhod_meas,10,'filled', 'MarkerFaceColor', [0, 0, 1]);
231 scatter(time_vec, range_rate_vec, 10, 'filled', 'MarkerFaceColor',[1, 0, 0]);
232
233 ylabel('\dot{\rho} (km/s)');
234 grid on;
235
236 nexttile; hold on;
237 scatter(t_meas,phi_meas,10,'filled', 'MarkerFaceColor', [0, 0, 1]);
238 scatter(time_vec, angle_vec, 10, 'filled', 'MarkerFaceColor',[1, 0, 0]);
239
240 ylabel('\phi (rad)');
241 xlabel('Time (s)');
242 grid on;
243
244 %% Plot 4: True vs EKF estimated trajectory in Cartesian coordinates
245 figure('Color','w');
246 hold on;
247 plotSimulationStatesCartesian(t_plot,x_true_plot);
248 plotSimulationStatesCartesian(t_plot,xhat_plot);
249 title('EKF: True vs Estimated States (Cartesian)');
250 grid on;

```

Listing 12: Code for Final_Project_Part_2_Problem_5_part_a.m

File: Final_Project_Part_2_Problem_6.m

```

1
2 load('orbitdeterm_finalproj_KFdata.mat')
3
4 %% Plot Nonlinear Measurements vs. Time
5 outputs = convertYdata2Output(ydata,Dt);
6 % sanitize NaNs
7 outputs.visible_stations{1} = [];
8 outputs.y{1} = [];
9 figure; hold on;
10
11 plotStationMeasurements(outputs)

```

```

12 sgtitle("Given Measurements vs. Time")
13
14 %% LKF
15 mu_0 = getXNom(0);
16 N = 1400;
17 pos_variance = 1e-6;
18 vel_variance = 1e-5;
19 P_0 = diag([pos_variance, vel_variance, pos_variance, vel_variance]);
20 Q_KF = 1e-8*eye(2); % TODO: Use a proper guess
21 % Euler Approximation for DT Process Noise Covariance
22 G = [0 0;
23      1 0;
24      0 0;
25      0 1];
26 t_0 = 0;
27 params.mu = mu;
28 [P_plus, x_hat_plus, P_minus, x_hat_minus] = DTLinearizedKF(P_0,mu_0,N,Dt,Q_KF,Rtrue,ydata,params,t_0);
29 x_hat_plus = x_hat_plus';
30 [sigma_x_hat_plus, ~] = getSigmas(n, N, P_plus, P_minus);
31
32 %% Plot
33 t=0:Dt:N*Dt;
34 figure; hold on;
35 sgtitle("DT Linearized Kalman Filter");
36 plotSimulationStates(t(2:end),x_hat_plus(:,2:end));
37 plotSimulationStates(t(2:end),x_hat_plus(:,2:end)+sigma_x_hat_plus);
38 plotSimulationStates(t(2:end),x_hat_plus(:,2:end)-sigma_x_hat_plus);
39 for subplot_i = 1:n
40     subplot(n,1,subplot_i); hold on;
41     legend("$x$", "$+2\sigma$", "$+2\sigma$", "$-2\sigma$", "Location", "eastoutside", "Interpreter", 'latex');
42 end
43
44 t=0:Dt:N*Dt;
45 figure; hold on;
46 sgtitle("DT Linearized Kalman Filter");
47 plotSimulationStates(t(2:end),2*sigma_x_hat_plus);
48 plotSimulationStates(t(2:end),-2*sigma_x_hat_plus);
49 for subplot_i = 1:n
50     subplot(n,1,subplot_i); hold on;
51     legend("$+2\sigma$", "$-2\sigma$", "Location", "eastoutside", "Interpreter", 'latex');
52 end
53 subplot(4,1,1); xlabel("$x$")
54 subplot(4,1,2); xlabel("$\dot{x}$")
55 subplot(4,1,3); xlabel("$y$")
56 subplot(4,1,4); xlabel("$\dot{y}$")
57
58 %% EKF
59 r_0 = sqrt(6678^2);
60 mu = 398600; % Gravitational parameter for Earth in km^3/s^2
61 R_E = 6378; %Earth radius [km]
62 omega_E = 2*pi/86400; %Earth rotation rate [rad/s]
63 x_0 = [6678; 0; 0; r_0 * sqrt(mu/r_0^3)];
64 params = struct('mu', mu, 'R_E', R_E, 'omega_E', omega_E);
65
66 [x_hat_plus_ekf, P_plus_ekf, innov_ekf, S_ekf] = EKF_Orbit(ydata, t, mu_0, P_0, Q_KF, Rtrue, params.mu, ←
67     params.R_E, params.omega_E);
68 [sigma_x_hat_plus_ekf, ~] = getSigmas(n, N, P_plus_ekf, P_plus_ekf);
69 x_hat_plus_ekf = x_hat_plus_ekf';
70 %% Plot
71 figure; hold on;
72 sgtitle("EKF State Estimate");
73 plotSimulationStates(t(2:end),x_hat_plus_ekf(2:end,:));
74 plotSimulationStates(t(2:end),x_hat_plus_ekf(2:end,:)+sigma_x_hat_plus_ekf);
75 plotSimulationStates(t(2:end),x_hat_plus_ekf(2:end,:)-sigma_x_hat_plus_ekf);
76 for subplot_i = 1:n
77     subplot(n,1,subplot_i); hold on;
78     legend("$x$", "$+2\sigma$", "$+2\sigma$", "$-2\sigma$", "Location", "eastoutside", "Interpreter", 'latex');
79 end

```

```

80
81 %% Plot
82 figure; hold on;
83 sgttitle("EKF State Estimate Certainties");
84 plotSimulationStates(t(2:end),2*sigma_x_hat_plus_ekf);
85 plotSimulationStates(t(2:end),-2*sigma_x_hat_plus_ekf);
86 for subplot_i = 1:n
87     subplot(n,1,subplot_i); hold on;
88     legend("$+2\sigma$", "$-2\sigma$", "Location", "eastoutside", "Interpreter", 'latex');
89 end
90 %% Helper Functions
91 % TODO: Move to common?
92
93 function [sigma_x_hat_plus, sigma_x_hat_minus] = getSigmas(n, N, P_plus, P_minus)
94 sigma_2_x_hat_plus = zeros(n,N);
95 sigma_2_x_hat_minus = zeros(n,N);
96 for k = 1:N
97     sigma_2_x_hat_plus(:,k) = diag(P_plus(:, :, k));
98     sigma_2_x_hat_minus(:,k) = diag(P_minus(:, :, k));
99 end
100 sigma_x_hat_plus = sqrt(sigma_2_x_hat_plus)';
101 sigma_x_hat_minus = sqrt(sigma_2_x_hat_minus)';
102 end

```

Listing 13: Code for Final_Project_Part_2_Problem_6.m

File: GetAllStationMeasurements.m

```

1 function [y_k, visible_stations_k, y_k_stacked] = GetAllStationMeasurements(x, t)
2 % TODO: Check if this should be global
3 N_stations = 12;
4
5 y_k = [];
6 visible_stations_k = [];
7 y_k_stacked = [];
8 for i = 1:N_stations
9     [y_i, is_visible] = GetStationMeasurement(x, t, i);
10    if is_visible
11        % Add measurement vector y~i(t) to stacked measurement vector
12        y_k = [y_k; y_i];
13        stationID = i;
14        visible_stations_k = [visible_stations_k stationID];
15        identified_measurements = [y_i; stationID];
16        y_k_stacked = [y_k_stacked identified_measurements];
17    end
18 end
19 end

```

Listing 14: Code for GetAllStationMeasurements.m

File: GetGroundStationState.m

```

1 % i is the station index (1 to 12)
2 function [X_i, Y_i, X_dot_i, Y_dot_i, theta_i] = GetGroundStationState(i, t)
3 % TODO: Check if these should be global
4 R_E = 6378; % Earth radius in km
5 omega_E = 2*pi/86400; % Earth rotation speed in rad/s
6 theta_i0 = (i - 1) * pi/6; % Initial location angle of ground station i
7
8 X_i = R_E * cos(omega_E * t + theta_i0);
9 Y_i = R_E * sin(omega_E * t + theta_i0);
10 X_dot_i = -omega_E * R_E * sin(omega_E * t + theta_i0);
11 Y_dot_i = omega_E * R_E * cos(omega_E * t + theta_i0);
12 theta_i = atan2(Y_i, X_i);
13 end

```

Listing 15: Code for GetGroundStationState.m

File: GetStationMeasurement.m

```

1 function [y_i, is_visible] = GetStationMeasurement(x, t, i)
2     [X_i, Y_i, X_dot_i, Y_dot_i, theta_i] = GetGroundStationState(i, t);
3
4     X = x(1);
5     X_dot = x(2);
6     Y = x(3);
7     Y_dot = x(4);
8     phi_i = atan2((Y - Y_i), (X - X_i));
9
10    phi_min = -pi/2 + theta_i;
11    phi_max = pi/2 + theta_i;
12
13    % Check if the LOS angle is within the station's "sky-side" view cone
14    is_visible = (phi_i >= phi_min) && (phi_i <= phi_max);
15    angle_diff_raw = phi_i - theta_i; % GEN_AI
16    true_angle_diff_rad = mod(angle_diff_raw + pi, 2*pi) - pi; % GEN_AI
17    angle_rad = abs(true_angle_diff_rad); % GEN_AI
18    is_visible = angle_rad < pi/2; % GEN_AI
19    rho_i = sqrt((X - X_i)^2 + (Y - Y_i)^2);
20    X_diff = X - X_i;
21    X_dot_diff = X_dot - X_dot_i;
22    Y_diff = Y - Y_i;
23    Y_dot_diff = Y_dot - Y_dot_i;
24    rho_dot_i = ((X_diff * X_dot_diff) + (Y_diff * Y_dot_diff))/rho_i;
25    y_i = [rho_i; rho_dot_i; phi_i];
26 end

```

Listing 16: Code for GetStationMeasurement.m

File: H_jacobian.m

```

1 function [y, H, visible_ids] = H_jacobian(x, t, visible_ids, R_E, omega_E)
2 %Compute measurements and Jacobian for all visible stations.
3 %Inputs:
4 % x      - 4x1 spacecraft state [X; Xdot; Y; Ydot]
5 % t      - current time [s]
6 % visible_ids- List of visible station IDs from actual time step measurements
7 % R_E     - Earth radius [km]
8 % omega_E - Earth rotation rate [rad/s]
9 %
10 %Outputs:
11 % y      - stacked measurement vector for visible stations (3*number_visible x 1)
12 % H      - stacked measurement Jacobian (3*number_visible x 4)
13 % visible_ids - station IDs used (returned as a column vector)
14
15 %Pull Out Current State
16 x1 = x(1); x2 = x(2); x3 = x(3); x4 = x(4);
17
18 %Ensure visible_ids is a column vector
19 visible_ids = visible_ids(:);
20
21 %Get Number of Measurements
22 number_visible = numel(visible_ids);
23
24 %Compute Current Position of Each Ground Tracking Station
25 X_i = zeros(1,12);
26 X_dot_i = zeros(1,12);
27 Y_i = zeros(1,12);
28 Y_dot_i = zeros(1,12);
29
30 for i = 1:12
31     theta_i_0 = (i - 1)*pi/6;
32     X_i(i) = R_E*cos(omega_E*t + theta_i_0);
33     Y_i(i) = R_E*sin(omega_E*t + theta_i_0);
34     X_dot_i(i) = -omega_E*R_E*sin(omega_E*t + theta_i_0);

```

```

35     Y_dot_i(i) = omega_E*R_E*cos(omega_E*t + theta_i_0);
36 end
37
38 %Preallocate output arrays
39 H = zeros(3*number_visible,4);
40 y = zeros(3*number_visible,1);
41
42 %Loop through specified visible station IDs
43 for k = 1:number_visible
44     i = visible_ids(k);
45
46     %Solve for Estimated Position Using Non-Linear CT Functions
47     delta_X = x1 - X_i(i);
48     delta_X_dot = x2 - X_dot_i(i);
49     delta_Y = x3 - Y_i(i);
50     delta_Y_dot = x4 - Y_dot_i(i);
51
52     phi_i = atan2(x3 - Y_i(i), x1 - X_i(i));
53     rho_i = sqrt(delta_X^2 + delta_Y^2);
54     rho_dot_i = (delta_X*delta_X_dot + delta_Y*delta_Y_dot)/rho_i;
55
56     y_i = [rho_i; rho_dot_i; phi_i];
57
58     %Calculate DT H Matrix Linearized at Estimation Point
59     H_i = zeros(3,4);
60     a = delta_X*delta_X_dot + delta_Y*delta_Y_dot;
61
62     H_i(1,1) = delta_X/rho_i;
63     H_i(1,3) = delta_Y/rho_i;
64
65     H_i(2,1) = (delta_X_dot/rho_i) - (a*delta_X/rho_i^3);
66     H_i(2,3) = (delta_Y_dot/rho_i) - (a*delta_Y/rho_i^3);
67     H_i(2,2) = delta_X/rho_i;
68     H_i(2,4) = delta_Y/rho_i;
69
70     H_i(3,1) = -delta_Y/(rho_i^2);
71     H_i(3,3) = delta_X/(rho_i^2);
72
73     %Add to Output Matrix
74     idx_start = 3*(k - 1) + 1;
75     y(idx_start:idx_start+2) = y_i;
76     H(idx_start:idx_start+2,:) = H_i;
77 end
78
79 end

```

Listing 17: Code for H_jacobian.m

File: NonlinearSimulation.m

```

1 function outputs = NonlinearSimulation(x_0,T,Dt)
2     t = 0:Dt:T;
3     % 1. Simulate the states
4     options = odeset('RelTol', 1e-12, 'AbsTol', 1e-12);
5     [~,x] = ode45(@NonlinearStateEquations, t, x_0, options);
6     outputs.x = x;
7
8     % 2. Calculate the Measurements at Each Time Step
9     N_steps = T/Dt+1;
10    y = cell(N_steps, 1); % Use a cell array for variable-length vectors
11    visible_stations = cell(N_steps,1);
12    y_stacked = cell(N_steps,1);
13    for k = 1:N_steps
14        current_t = t(k);
15        current_x = x(k, :);
16        [y_k,visible_stations{k},y_stacked{k}] = GetAllStationMeasurements(current_x, current_t);
17        y{k} = y_k';
18    end

```

```

19 outputs.y = y;
20 outputs.visible_stations = visible_stations;
21 outputs.y_stacked = y_stacked;
22 end

```

Listing 18: Code for NonlinearSimulation.m

File: NonlinearSimulationNoisy.m

```

1 function outputs = NonlinearSimulationNoisy(x_0,t_0,T,Dt,S_w,S_v)
2     t = t_0:Dt:T;
3     % 1. Simulate the states
4     options = odeset('RelTol', 1e-12, 'AbsTol', 1e-12);
5     [~,x] = ode45(@NonlinearStateEquations, t, x_0, options);
6     outputs.x = x;
7
8     [~, ~, ~, ~, Gamma_0, ~, ~, ~, ~] = getLinearizedMatrices(x_0, t_0, Dt, 0, 0);
9
10    if numel(t) == 2
11        outputs.x = x(end,:);
12        x = x(end,:);
13    end
14    M = size(S_w, 1);
15    q_k = randn(M, numel(t));
16    xi_k = S_w * q_k;
17    Omega_tilde_k = Gamma_0 * Dt;
18    w_k = Omega_tilde_k * xi_k;
19    x = x' + w_k;
20    x = x';
21    % 2. Calculate the Measurements at Each Time Step
22    N_steps = (T-t_0)/Dt;
23    y = cell(N_steps, 1); % Use a cell array for variable-length vectors
24    visible_stations = cell(N_steps,1);
25    y_stacked = cell(N_steps,1);
26
27    for k = 1:N_steps
28        current_t = t(k+1);
29        current_x = x(k+1, :)';
30        [y_k,visible_stations{k},y_stacked{k}] = GetAllStationMeasurements(current_x, current_t);
31        p = size(y_k,1);
32        q_k = randn(p, 1);
33        N_visible_stations_k = numel(visible_stations{k});
34        S_v_diag = mat2cell(repmat(S_v,1,N_visible_stations_k),3,3*ones(1,N_visible_stations_k));
35        S_v_blk = blkdiag(S_v_diag{:});
36        v_k = S_v_blk * q_k; % Measurement noise
37        y_k = y_k + v_k;
38
39        y_stacked{k} = reshape(y_stacked{k},4,[]) + [reshape(v_k,3,[]);zeros(1,N_visible_stations_k)];
40        y{k} = y_k';
41    end
42    outputs.y = y;
43    outputs.visible_stations = visible_stations;
44    outputs.y_stacked = y_stacked;
45 end

```

Listing 19: Code for NonlinearSimulationNoisy.m

File: NonlinearStateEquations.m

```

1 function dx = NonlinearStateEquations(~, x)
2     % TODO: Check if we want that globally
3     mu = 398600; % Standard gravitational parameter
4     r = sqrt(x(1)^2 + x(3)^2);
5     x1_dot = x(2);
6     x2_dot = -mu*x(1)/r^3;
7     x3_dot = x(4);

```

```

8 | x4_dot = -mu*x(3)/r^3;
9 | dx = [x1_dot; x2_dot; x3_dot; x4_dot];
10 | end

```

Listing 20: Code for NonlinearStateEquations.m

File: Part1Question3LinearDTSimulation.m

```

1 | %% Simulation Config
2 | % Dummy Variables % TODO: Remove
3 | n = numel(x_nom);
4 | p = 2; % TODO: Vary this.
5 | Q = zeros(n);
6 | R = zeros(p);
7 | r = Re + 300;
8 | T = 2*pi*sqrt(r^3/mu);
9 | Dt = 10;
10 | T = 1400*Dt;
11 |
12 | params.mu = mu;
13 | dx0 = 0.001*x_nom; % TODO: Good IC
14 | dx0 = [0,0.075,0,-0.021];
15 | %% Simulation
16 | dt_LTI_outputs = DTSimulation(Q,R,dx0',T,delta_t,params);
17 | %% Plot
18 | % DT Simulated States and Nominal States vs. Time
19 | plotSimulationStates(dt_LTI_outputs.t,dt_LTI_outputs.x)
20 | plotSimulationStates(dt_LTI_outputs.t,dt_LTI_outputs.x_nom)
21 | sgtitle("DT Simulated States and Nominal States vs. Time")
22 |
23 | %% Sanity Check on Visibility
24 | figure; hold on;
25 | plotSimulationStatesCartesian(dt_LTI_outputs.t,dt_LTI_outputs.x)
26 | plotSimulationStatesCartesian(dt_LTI_outputs.t,dt_LTI_outputs.x_nom)
27 | plot(Re*sin(0:0.01:(2*pi)),Re*cos(0:0.01:(2*pi)))
28 | ifoo = 251;
29 | for stationID = 1:12
30 |     [X_1_tfoo, Y_1_tfoo, ~, ~, ~] = GetGroundStationState(stationID, dt_LTI_outputs.t(ifoo));
31 |     scatter(X_1_tfoo,Y_1_tfoo,'filled','HandleVisibility',"off")
32 |     text(X_1_tfoo,Y_1_tfoo,num2str(stationID))
33 | end
34 | x_tfoo = dt_LTI_outputs.x(ifoo,:);
35 | scatter(x_tfoo(1),x_tfoo(3))
36 | sgtitle("DT Simulated States and Nominal States (Cartesian)")
37 |
38 | % DT Simulated State Perturbations vs. Time
39 | figure; hold on;
40 | plotSimulationStates(dt_LTI_outputs.t,dt_LTI_outputs.dx)
41 | sgtitle("DT Simulated State Perturbations vs. Time")
42 |
43 | % DT Simulated Measurements vs. Time
44 | figure; hold on;
45 | plotStationMeasurements(dt_LTI_outputs)
46 | sgtitle("DT Simulated Measurements vs. Time")

```

Listing 21: Code for Part1Question3LinearDTSimulation.m

File: PlotChiSquareTests.m

```

1 | function PlotChiSquareTests(FilterName, ANEES, ANIS, K, alpha, N, n, p)
2 |     % NEES Confidence Interval bounds
3 |     r1_x = chi2inv(alpha/2, N*n) / N;
4 |     r2_x = chi2inv(1 - alpha/2, N*n) / N;
5 |
6 |     % NIS Confidence Interval bounds
7 |     r1_y = chi2inv(alpha/2, N*p) / N;

```



```

8     r2_y = chi2inv(1 - alpha/2, N*p) / N;
9
10    % Expected value (theoretical mean) for a consistent filter
11    expected_nees = n;
12    expected_nis = p;
13
14    % Time vector for plotting
15    t_vec = 1:K;
16
17    figure();
18    subplot(1, 2, 1);
19    hold on; grid on;
20    scatter(t_vec, ANEES, 'b', 'DisplayName', 'ANEES (Average NEES)');
21    yline(r1_x, 'r--', 'LineWidth', 1.5, 'DisplayName', '95% Lower Bound');
22    yline(r2_x, 'r--', 'LineWidth', 1.5, 'DisplayName', '95% Upper Bound');
23    yline(expected_nees, 'k:', 'LineWidth', 1, 'DisplayName', 'Expected Value (n_x)');
24    xlabel('Time Step (k)');
25    ylabel('ANEES Value');
26    title([FilterName ' ANEES Test for Filter Consistency']);
27    legend('show', 'Location', 'best');
28    ylim([0.8*r1_x, 1.2*r2_x]);
29
30    subplot(1, 2, 2);
31    hold on; grid on;
32    scatter(t_vec, ANIS, 'b', 'DisplayName', 'ANIS (Average NIS)');
33    yline(r1_y, 'r--', 'LineWidth', 1.5, 'DisplayName', '95% Lower Bound');
34    yline(r2_y, 'r--', 'LineWidth', 1.5, 'DisplayName', '95% Upper Bound');
35    yline(expected_nis, 'k:', 'LineWidth', 1, 'DisplayName', 'Expected Value (n_y)');
36    xlabel('Time Step (k)');
37    ylabel('ANIS Value');
38    title([FilterName ' ANIS Test for Measurement Consistency']);
39    legend('show', 'Location', 'best');
40    ylim([0.8*r1_y, 1.2*r2_y]);
41 end

```

Listing 22: Code for PlotChiSquareTests.m

File: RunTMT_rmh.m

```

1 clear; clc; %close all;
2 rng(100);
3 r_0 = sqrt(6678^2);
4 mu = 398600; % Gravitational parameter for Earth in km^3/s^2
5 R_E = 6378; %Earth radius [km]
6 omega_E = 2*pi/86400; %Earth rotation rate [rad/s]
7 x_0 = [6678; 0; 0; r_0 * sqrt(mu/r_0^3)];
8
9 addpath('Extended Kalman Filter');
10
11 load('orbitdeterm_finalproj_KFdata.mat');
12 Dt = 10; % Time-step size 10s
13 K = 1400; % 1400 steps
14 N = 1000; % Number of Monte Carlo runs
15 alpha = 0.05; % For 95% Confidence Interval
16 n = 4; % Number of state vars
17 p = 3; % Size of measurement vector
18 Q_KF = 1e-10*eye(2); % TODO: Use a proper guess
19 R_KF = Rtrue;
20 [ANEES_LKF, ANIS_LKF, ANEES_EKF, ANIS_EKF] = TruthModelTesting_rmh(...
21     N, x_0, Dt, K, Qtrue, Rtrue, Q_KF, R_KF, ...
22     @DTLinearizedKF, @EKF_Orbit, struct('mu', mu, 'R_E', R_E, 'omega_E', omega_E));
23
24 %%
25 PlotChiSquareTests('LKF', ANEES_LKF, ANIS_LKF, K, alpha, N, n, p);
26 PlotChiSquareTests('EKF', ANEES_EKF, ANIS_EKF, K, alpha, N, n, p);

```

Listing 23: Code for RunTMT_rmh.m

File: TruthModelTesting_rmh.m

```

1 function [ANEES_LKF, ANIS_LKF, ANEES_EKF, ANIS_EKF] = TruthModelTesting_rmh(N, x_nom, Dt, K, Q_true, R_true, ←
    Q_KF_guess, R_KF_guess, LKF, EKF, params)
2 % Define variables
3 n = size(x_nom, 1);
4 p = 3; % Measurement vector dimension
5 KF_params.mu = params.mu;
6
7 % Arrays to store Ground Truth states and measurements
8 x_true = zeros(N, n, K+1);
9 y_true = cell(N, K+1);
10
11 % Arrays to store NEES and NIS results
12 NEES_LKF = zeros(N, K);
13 NIS_LKF = zeros(N, K);
14 NEES_EKF = zeros(N, K);
15 NIS_EKF = zeros(N, K);
16
17 % To calculate DT Process Noise Covariance from Q_true
18 [~, ~, ~, ~, Gamma_0, ~, ~, ~, ~] = getLinearizedMatrices(x_nom, 0, Dt, 0, params.mu);
19
20 disp(['Starting Monte Carlo simulation with N = ', num2str(N), ' runs...']);
21 for i = 1:N
22     % Choose dx_hat_plus_0 and P_0
23     dx_hat_plus_0 = zeros(1,n);
24     x_hat_plus_0 = x_nom + dx_hat_plus_0';
25
26
27     % TODO: Verify these are correct
28     pos_variance = 1e-6;
29     vel_variance = 1e-5;
30     ICs = diag([pos_variance, vel_variance, pos_variance, vel_variance]);
31
32     % Instantiate ground truth state x(0) randomly
33     dx_true_0 = mvnrnd(dx_hat_plus_0', ICs)';
34     x_true(i,:,1) = x_nom + dx_true_0;
35
36     % Get S_w, S_v from Cholesky decomposition
37     S_w = chol(Q_true, 'lower');
38     S_v = chol(R_true, 'lower');
39
40     % Loop over K time steps to simulate ground truth
41     % for k = 1:K
42     %     % Generate Ground Truth (Full Nonlinear Dynamics)
43     %
44     %     % ----- New process noise generation -----
45     %     M = size(Q_true, 1);
46     %     q_k = randn(M, 1);
47     %     xi_k = S_w * q_k;
48     %     Omega_tilde_k = Gamma_0 * Dt;
49     %     w_k = Omega_tilde_k * xi_k;
50     %     % -----
51     %
52     %     % Use Full Nonlinear Dynamics to propagate state
53     %     x_true_k = x_true(i,:,k);
54     %     nonlinearOutputs = NonlinearSimulation(x_true_k, Dt, Dt);
55     %     x_prop_k_plus_1 = nonlinearOutputs.x(end,:)';
56     %     x_true(i,:,k+1) = x_prop_k_plus_1 + w_k;
57     %
58     %     % Use noisy state to get measurements
59     %     [y_k_plus_1, visible_stations_k_plus_1, y_k_plus_1_stacked] = ←
    GetAllStationMeasurements(x_true(i,:,k+1), (k+1)*Dt);
60
61     %     % Add measurement noise to measurements
62     %     p_k_plus_1 = size(y_k_plus_1, 1);
63     %     q_k_plus_1 = randn(p_k_plus_1, 1);
64     %     N_visible_stations_k_plus_1 = numel(visible_stations_k_plus_1);

```

```

65 % S_v_diag = mat2cell(repmat(S_v, 1, N_visible_stations_k_plus_1), p, p*ones(1, ←
N_visible_stations_k_plus_1));
66 % S_v_blk = blkdiag(S_v_diag{:});
67 % v_k_plus_1 = S_v_blk * q_k_plus_1; % Measurement noise
68 % y_k_plus_1_stacked = reshape(y_k_plus_1_stacked, 4, []) + [reshape(v_k_plus_1, p, []) ; zeros(1, ←
N_visible_stations_k_plus_1)];
69 % y_true{i,k+1} = y_k_plus_1_stacked; % NOTE: measurements should be 4 x n
70 % end
71 t_0 = 0;
72
73 % ----- Alternative nonlinear simulation -----
74 t_f = 1400*Dt;
75 outputs_i = NonlinearSimulationNoisy(x_true(i,:,1),t_0,t_f,Dt,S_w,S_v);
76 ydata_simulated_i = convertOutput2Ydata(outputs_i);
77 ydata_simulated_i = [{NaN(4,1)};ydata_simulated_i];
78 [y_true{i,:}] = deal(ydata_simulated_i{:});
79 x_i = outputs_i.x(2:end,:);
80 x_true(i,:,2:end) =x_i';
81 % -----
82
83 %% Run Filters (LKF and EKF)
84
85 %Super bad implementation for now <---- CHANGE AT SOME POINT
86 tvec_ekf = 0:Dt:K*Dt;
87 %pos_variance = 100;
88 %vel_variance = 0.1;
89 %ICs = diag([pos_variance^2, vel_variance^2, pos_variance^2, vel_variance^2]);
90
91 % "for each test trajectory sample in the NEES and NIS tests, you should initialize the
92 % filter with exactly the same initial perturbation state estimate [...]
93 % and covariance"
94 % Linearized Kalman Filter (LKF)
95 [P_plus_lkf, x_hat_plus_lkf, ~, ~, innov_lkf, S_lkf] = LKF(ICs, x_hat_plus_0', K, Dt, Q_KF_guess, ←
R_KF_guess, y_true(i,2:end), KF_params, t_0);
96
97 % Extended Kalman Filter (EKF)
98 [x_hat_plus_ekf, P_plus_ekf, innov_ekf, S_ekf] = EKF(y_true(i,:), tvec_ekf, x_hat_plus_0', ICs, ←
Q_KF_guess, R_KF_guess, params.mu, params.R_E, params.omega_E);
99
100 %% Calculate NEES/NIS for this run
101
102 % Loop over K time steps to calculate NEES/NIS
103 for k = 1:K
104 % LKF NEES/NIS (uses perturbation)
105 error_lkf = (x_true(i,:,k) - x_hat_plus_lkf(k,:))';
106 NEES_LKF(i, k) = error_lkf' * (P_plus_lkf(:, :, k) \ error_lkf);
107 if ~isempty(innov_lkf{k})
108 NIS_LKF(i, k) = innov_lkf{k}' * (S_lkf{k} \ innov_lkf{k});
109 else
110 % No measurement available, exclude from mean
111 NIS_LKF(i, k) = NaN;
112 end
113
114 % EKF NEES/NIS (uses total state)
115 error_ekf = x_true(i,:,k)' - x_hat_plus_ekf(:,k);
116 NEES_EKF(i, k) = error_ekf' * (P_plus_ekf(:, :, k) \ error_ekf);
117 if ~isempty(innov_ekf{k})
118 NIS_EKF(i, k) = innov_ekf{k}' * (S_ekf{k} \ innov_ekf{k});
119 else
120 % No measurement available, exclude from mean
121 NIS_EKF(i, k) = NaN;
122 end
123 end
124
125 if mod(i, 10) == 0
126 disp(['Completed run ', num2str(i), ' of ', num2str(N)]);
127 end
128 end
129 disp('Monte Carlo simulation complete.');
```

```

130 assignin("base",'y_montecarlo_end',y_true(end,:))
131 assignin("base",'x_montecarlo_end',reshape(x_true(end,:,:),4,[]))
132 assignin("base",'x_hat_plus_lkf_end',x_hat_plus_lkf)
133 assignin("base",'x_hat_plus_lkf_end',x_hat_plus_lkf)
134 assignin("base",'x_hat_plus_ekf_1',x_hat_plus_ekf)
135 assignin("base",'P_plus_lkf_montecarlo_end',P_plus_lkf)
136
137 %% Statistical Test (ANEES & ANIS)
138 % Calculate the average mean across all runs
139 ANEES_LKF = nanmean(NEES_LKF, 1);
140 ANIS_LKF = nanmean(NIS_LKF, 1);
141 ANEES_EKF = nanmean(NEES_EKF, 1);
142 ANIS_EKF = nanmean(NIS_EKF, 1);
143 end

```

Listing 24: Code for TruthModelTesting_rmh.m

File: compareDTLinearizedAndNonLinear.m

```

1 %% Simulation Comparison
2
3 figure; hold on;
4 sgttitle("DT Linearized vs. Nonlinear Simulation States")
5 plotSimulationStates(dt_LTI_outputs.t,dt_LTI_outputs.x)
6 plotSimulationStates(t,outputs.x);
7 figure; hold on;
8 sgttitle("DT Linearized States error vs. Nonlinear Simulation States")
9 dt_LTI_outputs.error = dt_LTI_outputs.x - outputs.x;
10 plotSimulationStates(dt_LTI_outputs.t,dt_LTI_outputs.error);

```

Listing 25: Code for compareDTLinearizedAndNonLinear.m

File: convertOutput2Ydata.m

```

1 function ydata_simulated = convertOutput2Ydata(outputs)
2 ydata_simulated = cell(size(outputs.y));
3 for i_y = 1:numel(outputs.y)
4     y_k = outputs.y{i_y};
5     visible_stations_k = outputs.visible_stations{i_y};
6     ydata_simulated_k = [reshape(y_k,3,[]); visible_stations_k];
7     ydata_simulated{i_y} = ydata_simulated_k;
8 end
9 end

```

Listing 26: Code for convertOutput2Ydata.m

File: convertYdata2Output.m

```

1 function outputs = convertYdata2Output(ydata,Dt)
2 t = Dt*(0:(numel(ydata)-1));
3 outputs.t = t;
4 outputs.y = cell(size(ydata));
5 outputs.visible_stations = cell(size(ydata));
6 for i_ydata = 1:numel(ydata)
7     ydata_k = ydata{i_ydata};
8     if ~isempty(ydata_k)
9         outputs.y{i_ydata} = reshape(ydata_k(1:3,:),1,[]);
10    end
11 end
12 for i_ydata = 1:numel(ydata)
13     visible_stations_k = ydata{i_ydata};
14     if ~isempty(visible_stations_k)
15         outputs.visible_stations{i_ydata} = reshape(visible_stations_k(end,:),1,[]);
16    end
17 end

```

File: getLinearizedMatrices.m

```

1 function [A_k, B_k, C_k, D_k, Gamma_k, F_k, G_k, H_k, M_k] = getLinearizedMatrices(x_nom, time, delta_t, ←
    stationID, mu)
2
3 %Pull out linearization point variables
4 x1 = x_nom(1); %X Position [km]
5 x2 = x_nom(2); %X Velocity [km/s]
6 x3 = x_nom(3); %Y Position [km]
7 x4 = x_nom(4); %Y Velocity [km/s]
8
9 %Get Location and Velocity of Tracking Station
10 [X_i, Y_i, X_dot_i, Y_dot_i, ~] = GetGroundStationState(stationID, time);
11
12 %Define Commonly Used Variables
13 r = sqrt(x1^2 + x3^2); %Earth-centered radial distance [km]
14 r5 = r^5; %Earth-centered radial distance to the fifth power [km^5]
15
16 delta_X_i = x1 - X_i; %Relative spacecraft-station X position [km]
17 delta_X_dot_i = x2 - X_dot_i; %Relative spacecraft-station X velocity [km/s]
18 delta_Y_i = x3 - Y_i; %Relative spacecraft-station Y position [km]
19 delta_Y_dot_i = x4 - Y_dot_i; %Relative spacecraft-station Y velocity [km/s]
20
21 a = delta_X_i * delta_X_dot_i + delta_Y_i * delta_Y_dot_i; %Range rate numerator
22
23 rho_i = sqrt(delta_X_i^2 + delta_Y_i^2); %Spacecraft-Station Range [km]
24
25 %% Define Continuous Time Matrices
26 %Define CT Dynamics Jacobian A(x) at Linearization Point (inertial frame)
27 A_k = zeros(4, 4);
28 A_k(1, 2) = 1;
29 A_k(3, 4) = 1;
30 A_k(2, 1) = mu * (2*x1^2 - x3^2) / r5;
31 A_k(2, 3) = 3 * mu * x1 * x3 / r5;
32 A_k(4, 1) = 3 * mu * x1 * x3 / r5;
33 A_k(4, 3) = mu * (2*x3^2 - x1^2) / r5;
34
35 %Define CT Input Jacobian B(x) at Linearization Point
36 B_k = [0 0;
37        1 0;
38        0 0;
39        0 1];
40
41 %Define Measurement Jacobian C(x) at Linearization Point
42 C_k = zeros(3, 4);
43 C_k(1, 1) = delta_X_i / rho_i;
44 C_k(1, 3) = delta_Y_i / rho_i;
45 C_k(2, 1) = (delta_X_dot_i / rho_i) - ((a * delta_X_i) / rho_i^3);
46 C_k(2, 2) = delta_X_i / rho_i;
47 C_k(2, 3) = (delta_Y_dot_i / rho_i) - ((a * delta_Y_i) / rho_i^3);
48 C_k(2, 4) = delta_Y_i / rho_i;
49 C_k(3, 1) = -1 * delta_Y_i / rho_i^2;
50 C_k(3, 3) = delta_X_i / rho_i^2;
51
52 %Define Force Influence on Measurement Jacobian D(x) at Linearization Point
53 D_k = zeros(3, 2);
54
55 %Define CT Disturbance Jacobian Gamma(x) at Linearization Point
56 Gamma_k = [0 0;
57            1 0;
58            0 0;
59            0 1];
60
61 %% Determine Discrete Time Matrices With Eulerization

```

```

62 | %Determine Eulerized DT F and G matrices
63 | F_k = eye(size(A_k)) + delta_t * A_k;
64 | G_k = delta_t * B_k;
65 |
66 | %Copy C and D matrices to H and M
67 | H_k = C_k; %CT -> DT is identical for H matrix
68 | M_k = D_k; %CT -> DT is identical for M matrix
69 |
70 |
71 | end

```

Listing 28: Code for getLinearizedMatrices.m

File: getSigmas.m

```

1 | function [sigma_x_hat_plus, sigma_x_hat_minus] = getSigmas(n, N, P_plus, P_minus)
2 | sigma_2_x_hat_plus = zeros(n,N);
3 | sigma_2_x_hat_minus = zeros(n,N);
4 | for k = 1:N
5 |     sigma_2_x_hat_plus(:,k) = diag(P_plus(:, :, k));
6 |     sigma_2_x_hat_minus(:,k) = diag(P_minus(:, :, k));
7 | end
8 | sigma_x_hat_plus = sqrt(sigma_2_x_hat_plus)';
9 | sigma_x_hat_minus = sqrt(sigma_2_x_hat_minus)';
10 | end

```

Listing 29: Code for getSigmas.m

File: getXNom.m

```

1 | function x = getXNom(t)
2 | mu = 398600; %Gravitational Constant [km^3/s^2]
3 | Re = 6378; %Earth radius [km]
4 | omegaE = (2*pi) / 86400; %Earth rotational rate [rad/s]
5 | r = Re + 300;
6 | theta_dot = sqrt(mu/(r^3));
7 | theta_0 = 0; % TODO: allow non-zero
8 | theta = theta_dot.*t + theta_0;
9 | x(:,1) = r*cos(theta);
10 | x(:,2) = -r*theta_dot.*sin(theta);
11 | x(:,3) = r*sin(theta);
12 | x(:,4) = r*theta_dot.*cos(theta);
13 | end

```

Listing 30: Code for getXNom.m

File: orbit_dynamics.m

```

1 | function dx = orbit_dynamics(~, x, u, mu)
2 | %Calculates continuous-time spacecraft dynamics derivatives.
3 | %Inputs:
4 | % ~ - Placeholder for used time variable, kept for ODE45
5 | % x - 4x1 state vector [X; Xdot; Y; Ydot] in km and km/s
6 | % u - 2x1 control input [u1; u2] (accelerations in km/s^2)
7 | % mu - gravitational parameter [km^3/s^2]
8 | %
9 | %Output:
10 | % dx - 4x1 time derivative of the state
11 |
12 | %Pull Out States
13 | x1 = x(1); x2 = x(2);
14 | x3 = x(3); x4 = x(4);
15 |
16 | %Calculate Derivatives
17 | x2_dot = -mu * x1 / (sqrt(x1^2 + x3^2))^3 + u(1);

```

```

18 | x4_dot = -mu * x3 / (sqrt(x1^2 + x3^2))^3 + u(2);
19 |
20 | %Assemble Output Derivative
21 | dx = [x2; x2_dot; x4; x4_dot];
22 |
23 | end

```

Listing 31: Code for orbit_dynamics.m

File: plotLastTrial.m

```

1 | %% Plot Last trial
2 |
3 | e_x_montecarlo_end = x_montecarlo_end - x_hat_plus_lkf_end';
4 | e_x_montecarlo_end = e_x_montecarlo_end';
5 | [sigma_plus_lkf_montecarlo_end, ~] = getSigmas(n, K, P_plus_lkf_montecarlo_end, P_plus_lkf_montecarlo_end);
6 |
7 | t = Dt*0:K;
8 | figure; hold on;
9 | sgttitle("DT Linearized Kalman Filter");
10 | plotSimulationStates(t(2:end), e_x_montecarlo_end(2:end,:));
11 | plotSimulationStates(t(2:end), 2*sigma_plus_lkf_montecarlo_end);
12 | plotSimulationStates(t(2:end), -2*sigma_plus_lkf_montecarlo_end);
13 | for subplot_i = 1:n
14 |     subplot(n,1,subplot_i); hold on;
15 |     legend("$x$", "$+2\sigma$", "$+2\sigma$", "$-2\sigma$", "Location", "eastoutside", "Interpreter", 'latex');
16 | end

```

Listing 32: Code for plotLastTrial.m

File: plotSimulationStates.m

```

1 | function plotSimulationStates(t,x)
2 | %% Plot
3 | subplot_ylabels = {'$$x [km]$$';...
4 |     '$$\dot{x} [km/s]$$';...
5 |     '$$y [km]$$'
6 |     '$$\dot{y} [km/s]$$';};
7 | [~, n] = size(x);
8 | for subplot_i = 1:n
9 |     subplot(n,1,subplot_i); hold on;
10 |    plot(t,x(:,subplot_i));
11 |    % i_x_A = x_A_indices(subplot_i);
12 |    % scatter(t_window,x_A(i_x_A,k_window+1));
13 |    ylabel(subplot_ylabels(subplot_i), "Interpreter", "latex")
14 |    xlabel('t [s]')
15 | end
16 | end

```

Listing 33: Code for plotSimulationStates.m

File: plotSimulationStatesCartesian.m

```

1 | function plotSimulationStatesCartesian(t,x)
2 | %% Plot
3 |
4 | [~, n]= size(x);
5 | plot(x(:,1),x(:,3));
6 | axis equal;
7 | grid on;
8 | % i_x_A = x_A_indices(subplot_i);
9 | % scatter(t_window,x_A(i_x_A,k_window+1));
10 |
11 | end

```

Listing 34: Code for plotSimulationStatesCartesian.m

File: plotStationMeasurements.m

```

1 function plotStationMeasurements(outputs) % GENAI
2 % PLOTSTATIONMEASUREMENTS Plots Rho, Rho_Dot, and Phi measurements from various stations, plus a subplot ←
   showing station visibility over time, using matching colors. % GENAI
3 % % GENAI
4 % Inputs: % GENAI
5 %     outputs.t : 1xM vector of time instances (s). % GENAI
6 %     outputs.visible_stations : Mx1 cell array; cell k contains a vector of station IDs (double) visible ←
   at time outputs.t(k). % GENAI
7 %     outputs.y : Mx1 cell array; cell k contains a vector of measurements [rho1, rdot1, phi1, rho2, rdot2, ←
   phi2, ...] corresponding to the station IDs in outputs.visible_stations{k}. % GENAI
8 ptSize=20; % GENAI
9 stationMap=containers.Map('KeyType','double','ValueType','any'); % GENAI
10 visibilityMap=containers.Map('KeyType','double','ValueType','any'); % GENAI
11 for k=1:length(outputs.t) % GENAI
12     ids=outputs.visible_stations{k}; % GENAI
13     measurements=outputs.y{k}; % GENAI
14     if isempty(ids) % GENAI
15         continue; % GENAI
16     end % GENAI
17     for i=1:length(ids) % GENAI
18         current_id=ids(i); % GENAI
19         vals=measurements((i-1)*3+1:(i-1)*3+3); % GENAI
20         new_row=[outputs.t(k),vals]; % GENAI
21         if isKey(stationMap,current_id) % GENAI
22             stationMap(current_id)=[stationMap(current_id);new_row]; % GENAI
23         else % GENAI
24             stationMap(current_id)=new_row; % GENAI
25         end % GENAI
26         if isKey(visibilityMap,current_id) % GENAI
27             visibilityMap(current_id)=[visibilityMap(current_id);outputs.t(k),current_id]; % GENAI
28         else % GENAI
29             visibilityMap(current_id)=[outputs.t(k),current_id]; % GENAI
30         end % GENAI
31     end % GENAI
32 end % GENAI
33 all_ids=sort(cell2mat(keys(stationMap))); % GENAI
34 if isempty(all_ids) % GENAI
35     warning('No station data found to plot.');
```


File: plotStationMeasurements2.m

```

1 function plotStationMeasurements(outputs) % GENAI
2 % PLOTSTATIONMEASUREMENTS Plots Rho, Rho_Dot, and Phi measurements from various stations, plus a subplot ←
   showing station visibility over time, using matching colors. % GENAI
3 % % GENAI
4 % Inputs: % GENAI
5 %     outputs.t : 1xM vector of time instances (s). % GENAI
6 %     outputs.visible_stations : Mx1 cell array; cell k contains a vector of station IDs (double) visible ←
   at time outputs.t(k). % GENAI
7 %     outputs.y : Mx1 cell array; cell k contains a vector of measurements [rho1, rdot1, phi1, rho2, rdot2, ←
   phi2, ...] corresponding to the station IDs in outputs.visible_stations{k}. % GENAI
8 ptSize=20; % GENAI
9 stationMap=containers.Map('KeyType','double','ValueType','any'); % GENAI
10 visibilityMap=containers.Map('KeyType','double','ValueType','any'); % GENAI
11 for k=1:length(outputs.t) % GENAI
12     ids=outputs.visible_stations{k}; % GENAI
13     measurements=outputs.y{k}; % GENAI
14     if isempty(ids) % GENAI
15         continue; % GENAI
16     end % GENAI
17     for i=1:length(ids) % GENAI
18         current_id=ids(i); % GENAI
19         vals=measurements((i-1)*3+1:(i-1)*3+3); % GENAI
20         new_row=[outputs.t(k),vals]; % GENAI
21         if isKey(stationMap,current_id) % GENAI
22             stationMap(current_id)=[stationMap(current_id);new_row]; % GENAI
23         else % GENAI
24             stationMap(current_id)=new_row; % GENAI
25         end % GENAI
26         if isKey(visibilityMap,current_id) % GENAI
27             visibilityMap(current_id)=[visibilityMap(current_id);outputs.t(k),current_id]; % GENAI
28         else % GENAI
29             visibilityMap(current_id)=[outputs.t(k),current_id]; % GENAI
30         end % GENAI
31     end % GENAI
32 end % GENAI
33 all_ids=sort(cell2mat(keys(stationMap))); % GENAI
34 if isempty(all_ids) % GENAI
35     warning('No station data found to plot.');
```

```

60 ylim([min(all_ids)-0.5,max(all_ids)+0.5]); % GENAI
61 % TODO: link axes is really slow. Consider uncommenting.
62 % linkaxes([ax1,ax2,ax3,ax4],'x'); % GENAI
63
64 % --- X-Position Alignment --- % GENAI
65 p1=get(ax1,'Position'); % GENAI
66 p2=get(ax2,'Position'); % GENAI
67 p3=get(ax3,'Position'); % GENAI
68 p4=get(ax4,'Position'); % GENAI
69 left_x=max([p1(1),p2(1),p3(1),p4(1)]); % Find the rightmost left edge % GENAI
70 width_x=min([p1(3),p2(3),p3(3),p4(3)]); % Find the narrowest width % GENAI
71 set(ax1,'Position',[left_x,p1(2),width_x,p1(4)]); % GENAI
72 set(ax2,'Position',[left_x,p2(2),width_x,p2(4)]); % GENAI
73 set(ax3,'Position',[left_x,p3(2),width_x,p3(4)]); % GENAI
74 set(ax4,'Position',[left_x,p4(2),width_x,p4(4)]); % GENAI
75 end % GENAI

```

Listing 36: Code for plotStationMeasurements2.m

File: plotStationMeasurementsNoColor.m

```

1 function plotStationMeasurementsNoColor(outputs)
2 % PLOTSTATIONMEASUREMENTS Plots Rho, Rho_Dot, and Phi measurements from various stations, plus a subplot ←
   showing station visibility over time, using matching colors. % GENAI
3 % % GENAI
4 % Inputs: % GENAI
5 %     outputs.t : 1xM vector of time instances (s). % GENAI
6 %     outputs.visible_stations : Mx1 cell array; cell k contains a vector of station IDs (double) visible ←
   at time outputs.t(k). % GENAI
7 %     outputs.y : Mx1 cell array; cell k contains a vector of measurements [rho1, rdot1, phi1, rho2, rdot2, ←
   phi2, ...] corresponding to the station IDs in outputs.visible_stations[k]. % GENAI
8 ptSize=20; % GENAI
9 stationMap=containers.Map('KeyType','double','ValueType','any'); % GENAI
10 visibilityMap=containers.Map('KeyType','double','ValueType','any'); % GENAI
11 for k=1:length(outputs.t) % GENAI
12     ids=outputs.visible_stations{k}; % GENAI
13     measurements=outputs.y{k}; % GENAI
14     if isempty(ids) % GENAI
15         continue; % GENAI
16     end % GENAI
17     for i=1:length(ids) % GENAI
18         current_id=ids(i); % GENAI
19         vals=measurements((i-1)*3+1:(i-1)*3+3); % GENAI
20         new_row=[outputs.t(k),vals]; % GENAI
21         if isKey(stationMap,current_id) % GENAI
22             stationMap(current_id)=[stationMap(current_id);new_row]; % GENAI
23         else % GENAI
24             stationMap(current_id)=new_row; % GENAI
25         end % GENAI
26         if isKey(visibilityMap,current_id) % GENAI
27             visibilityMap(current_id)=[visibilityMap(current_id);outputs.t(k),current_id]; % GENAI
28         else % GENAI
29             visibilityMap(current_id)=[outputs.t(k),current_id]; % GENAI
30         end % GENAI
31     end % GENAI
32 end % GENAI
33 all_ids=sort(cell2mat(keys(stationMap))); % GENAI
34 if isempty(all_ids) % GENAI
35     warning('No station data found to plot.');
```

```

45 ylabel('Station ID'); % GENAI
46 xlabel('Time (t)'); % GENAI
47 for i=1:length(all_ids) % GENAI
48     id=all_ids(i); % GENAI
49     data=stationMap(id); % GENAI
50     vis_data=visibilityMap(id); % GENAI
51     legName=sprintf('Station %d',id); % GENAI
52     scatter(ax1,data(:,1),data(:,2),ptSize,'filled','.', 'DisplayName',legName); % GENAI
53     scatter(ax2,data(:,1),data(:,3),ptSize,'filled','.', 'DisplayName',legName); % GENAI
54     scatter(ax3,data(:,1),data(:,4),ptSize,'filled','.', 'DisplayName',legName); % GENAI
55     scatter(ax4,vis_data(:,1),vis_data(:,2),ptSize,'filled','.', 'DisplayName',legName); % GENAI
56 end % GENAI
57 yticks(all_ids); % GENAI
58 ylim([min(all_ids)-0.5,max(all_ids)+0.5]); % GENAI
59 legend(ax1,'show','Location','eastoutside'); % GENAI
60 linkaxes([ax1,ax2,ax3,ax4],'x'); % GENAI
61 end % GENAI

```

Listing 37: Code for plotStationMeasurementsNoColor.m

File: propagate_orbit.m

```

1 function x_next = propagate_orbit(x_curr, u_curr, dt, mu)
2 %Propagates the nonlinear dynamics one time step at a time (dt)
3 %Inputs:
4 % x_curr - 4x1 current state at t_k
5 % u_curr - 2x1 control input assumed ZOH (constant over [t_k, t_k+1])
6 % dt      - time step [seconds]
7 % mu      - gravitational parameter [km^3/s^2]
8 %
9 %Output:
10 % x_next - 4x1 state at t_k+1 after integrating dynamics
11
12 % Anonymous handle for ODE45 (U_curr and Mu are held constant)
13 dynfun = @(t, x) orbit_dynamics(t, x, u_curr, mu);
14
15 % Integrate from t=0 to t=dt
16 tspan = [0, dt];
17 opts = odeset('RelTol',1e-9,'AbsTol',1e-9);
18 [~, x_traj] = ode45(dynfun, tspan, x_curr, opts);
19
20 % Take final state
21 x_next = x_traj(end, :);
22 end

```

Listing 38: Code for propagate_orbit.m