ASEN 5044: State Estimation
Final Project: Statistical Orbit Determination

Ian Thomas
12/11/2020

Contents

1 Deterministic System Analysis

1.1 CT Jacobians

The spacecraft position in the orbital plane can be represented by Earth-centered coordinates X and Y, and the rates of change $\dot{X}$ and $\dot{Y}$. Let $r = \sqrt{X^2 + Y^2}$ represent the distance of the spacecraft from Earth's center.

If we assume a simple gravity point-mass force model, which obeys a simple inverse square law, we can write the full non-linear equations of motion as:

$$\ddot{X} = -\frac{\mu X}{r^3} + u_1 + \tilde{w}_1 \tag{1}$$

$$\ddot{Y} = -\frac{\mu Y}{r^3} + u_2 + \tilde{w}_2 \tag{2}$$

Defining the state vector $x = \sqrt{X, \dot{X}, Y, \dot{Y}}$, we can write:

$$\begin{aligned}
\dot{x}_1 &= \mathcal{F}_1(x, u, \tilde{w}) = x_2 \\
\dot{x}_2 &= \mathcal{F}_2(x, u, \tilde{w}) = -\frac{\mu x_1}{\sqrt{x_1^2 + x_3^2}^3} + u_1 + \tilde{w}_1 \\
\dot{x}_3 &= \mathcal{F}_3(x, u, \tilde{w}) = x_4 \\
\dot{x}_4 &= \mathcal{F}_4(x, u, \tilde{w}) = \frac{\mu x_3}{\sqrt{x_1^2 + x_3^2}^3} + u_2 + \tilde{w}_2
\end{aligned}$$

We compute the Jacobians as follows:

$$\tilde{A}\Big|_{\text{nom}} = \begin{bmatrix}
\frac{\partial \mathcal{F}_1}{\partial x_1}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_1}{\partial x_2}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_1}{\partial x_3}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_1}{\partial x_4}\Big|_{x_{\text{nom}}, u_{\text{nom}}} \\
\frac{\partial \mathcal{F}_2}{\partial x_1}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_2}{\partial x_2}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_2}{\partial x_3}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_2}{\partial x_4}\Big|_{x_{\text{nom}}, u_{\text{nom}}} \\
\frac{\partial \mathcal{F}_3}{\partial x_1}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_3}{\partial x_2}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_3}{\partial x_3}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_3}{\partial x_4}\Big|_{x_{\text{nom}}, u_{\text{nom}}} \\
\frac{\partial \mathcal{F}_4}{\partial x_1}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_4}{\partial x_2}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_4}{\partial x_3}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_4}{\partial x_4}\Big|_{x_{\text{nom}}, u_{\text{nom}}}
\end{bmatrix}$$

$$\tilde{B}\Big|_{\text{nom}} = \begin{bmatrix}
\frac{\partial \mathcal{F}_1}{\partial u_1}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_1}{\partial u_2}\Big|_{x_{\text{nom}}, u_{\text{nom}}} \\
\frac{\partial \mathcal{F}_2}{\partial u_1}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_2}{\partial u_2}\Big|_{x_{\text{nom}}, u_{\text{nom}}} \\
\frac{\partial \mathcal{F}_3}{\partial u_1}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_3}{\partial u_2}\Big|_{x_{\text{nom}}, u_{\text{nom}}} \\
\frac{\partial \mathcal{F}_4}{\partial u_1}\Big|_{x_{\text{nom}}, u_{\text{nom}}} & \frac{\partial \mathcal{F}_4}{\partial u_2}\Big|_{x_{\text{nom}}, u_{\text{nom}}}
\end{bmatrix}$$

We can convert to polar coordinates for linearization. Let θ define the right handed angle from $\hat{x}$ to $\hat{y}$. We define:

$$r = \sqrt{X^2 + Y^2}$$
$$\theta = \begin{cases} \arctan\left(\frac{y}{x}\right) & x \geq 0 \\ \arctan\left(\frac{y}{x}\right) + \frac{\pi}{2} & x < 0 \end{cases}$$

We can take the derivatives of the functions above to get:

$$\frac{\partial r}{\partial X} = \frac{x}{\sqrt{x^2 + y^2}} \qquad \frac{\partial r}{\partial Y} = \frac{y}{\sqrt{x^2 + y^2}}$$
$$\frac{\partial \theta}{\partial X} = -\frac{y}{x^2 + y^2} \qquad \frac{\partial \theta}{\partial Y} = \frac{x}{x^2 + y^2}$$

But we know:

$$X = r\cos\theta$$
$$Y = r\sin\theta$$

So we can write:

$$\frac{\partial r}{\partial X} = \cos\theta \qquad \frac{\partial r}{\partial Y} = \sin\theta$$
$$\frac{\partial \theta}{\partial X} = -\frac{\sin\theta}{r} \qquad \frac{\partial \theta}{\partial Y} = \frac{\cos\theta}{r}$$

We can then write the state dynamics as:

$$\ddot{X} = -\frac{\mu\cos(\theta)}{r^2} + u_1 + \tilde{w}_1$$
$$\ddot{Y} = -\frac{\mu\sin(\theta)}{r^2} + u_2 + \tilde{w}_2$$

We note that neither the inputs nor disturbances depend on the states, so we can write the total derivatives:

$$\begin{aligned} \Delta\ddot{X} &\approx \mu\left[\frac{2\cos\theta}{r^3}\frac{\partial r}{\partial X} + \frac{\sin\theta}{r^2}\frac{\partial \theta}{\partial X}\right]\Delta X + \mu\left[\frac{2\cos\theta}{r^3}\frac{\partial r}{\partial Y} + \frac{\sin\theta}{r^2}\frac{\partial \theta}{\partial Y}\right]\Delta Y \\ &\approx \mu\left[\frac{2\cos\theta}{r^3}\cos\theta - \frac{\sin\theta}{r^2}\frac{\sin\theta}{r}\right]\Delta X + \mu\left[\frac{2\cos\theta}{r^3}\sin\theta + \frac{\sin\theta}{r^2}\frac{\cos\theta}{r}\right]\Delta Y \\ &\approx \mu\left[\frac{2\cos^2\theta - \sin^2\theta}{r^3}\right]\Delta X + \frac{3\mu\sin\theta\cos\theta}{r^3}\Delta Y \end{aligned}$$

We also note that $\ddot{X}$ has no dependence on $\dot{X}$ or $\dot{Y}$, so this shouldn't show up in the

linearization. Now linearizing $\ddot{Y}$:

$$\begin{aligned}
\Delta\ddot{Y} &\approx \mu\left[\frac{2\sin\theta}{r^3}\frac{\partial r}{\partial X} - \frac{\cos\theta}{r^2}\frac{\partial\theta}{\partial X}\right]\Delta X + \mu\left[\frac{2\sin\theta}{r^3}\frac{\partial r}{\partial Y} - \frac{\cos\theta}{r^2}\frac{\partial\theta}{\partial Y}\right]\Delta Y \\
&\approx \mu\left[\frac{2\sin\theta}{r^3}\cos\theta + \frac{\cos\theta}{r^2}\frac{\sin\theta}{r}\right]\Delta X + \mu\left[\frac{2\sin\theta}{r^3}\sin\theta - \frac{\cos\theta}{r^2}\frac{\cos\theta}{r}\right]\Delta Y \\
&\approx \frac{3\mu\sin\theta\cos\theta}{r^3}\Delta X + \mu\left[\frac{2\sin^2\theta - \cos^2\theta}{r^3}\right]\Delta Y
\end{aligned}$$

Using some trigonometric identities, we can write:

$$\begin{aligned}
\Delta\ddot{X} &\approx \mu\left[\frac{2\cos^2\theta - \sin^2\theta}{r^3}\right]\Delta X + \frac{3\mu\sin(2\theta)}{2r^3}\Delta Y \\
\Delta\ddot{Y} &\approx \frac{3\mu\sin(2\theta)}{2r^3}\Delta X + \mu\left[\frac{2\sin^2\theta - \cos^2\theta}{r^3}\right]\Delta Y
\end{aligned}$$

From this, we can write the (unevaluated) Jacobian matrices:

$$\tilde{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \mu\left[\frac{2\cos^2\theta - \sin^2\theta}{r^3}\right] & 0 & \frac{3\mu\sin(2\theta)}{2r^3} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{3\mu\sin(2\theta)}{2r^3} & 0 & \mu\left[\frac{2\sin^2\theta - \cos^2\theta}{r^3}\right] & 0 \end{bmatrix}$$

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

We note $\theta = \omega_O t$ is a function of time in the nominal solution, where ω_O is the angular velocity of the satellite in orbit. The nominal angular velocity for a circular orbit is given by:

$$\omega_O = \sqrt{\frac{\mu_E}{r_0^3}}$$

Initially, $\theta = 0$ so we can evaluate:

$$\tilde{A}\Big|_{\text{nom},t=k=0} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2.6768\text{e}-6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1.3384\text{e}-6 & 0 \end{bmatrix}$$

$$\tilde{B}\Big|_{\text{nom}} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

For the measurements, we can write:

$$\rho^i = \mathcal{H}_1^i(x) = \sqrt{(X - X_s^i)^2 + (Y - Y_s^i)^2}$$
$$\dot{\rho}^i = \mathcal{H}_2^i(x) = \frac{(X - X_s^i)\left(\dot{X} - \dot{X}_s^i\right) + (Y - Y_s^i)\left(\dot{Y} - \dot{Y}_s^i\right)}{\sqrt{(X - X_s^i)^2 + (Y - Y_s^i)^2}}$$
$$\phi^i = \mathcal{H}_3^i(x) = \arctan\left(\frac{Y - Y_s^i}{X - X_s^i}\right)$$

The Jacobian matrix is then:

$$\tilde{C}^i = \begin{bmatrix} \frac{\partial \mathcal{H}_1^i}{\partial x_1} & \frac{\partial \mathcal{H}_1^i}{\partial x_2} & \frac{\partial \mathcal{H}_1^i}{\partial x_3} & \frac{\partial \mathcal{H}_1^i}{\partial x_4} \\ \frac{\partial \mathcal{H}_2^i}{\partial x_1} & \frac{\partial \mathcal{H}_2^i}{\partial x_2} & \frac{\partial \mathcal{H}_2^i}{\partial x_3} & \frac{\partial \mathcal{H}_2^i}{\partial x_4} \\ \frac{\partial \mathcal{H}_3^i}{\partial x_1} & \frac{\partial \mathcal{H}_3^i}{\partial x_2} & \frac{\partial \mathcal{H}_3^i}{\partial x_3} & \frac{\partial \mathcal{H}_3^i}{\partial x_4} \end{bmatrix}$$

We then have (using many product rules):

$$\frac{\partial \mathcal{H}_1^i}{\partial X} = \left[\left(X - X_s^i\right)^2 + \left(Y - Y_s^i\right)^2\right]^{-\frac{1}{2}}\left(X - X_s^i\right)$$
$$\frac{\partial \mathcal{H}_1^i}{\partial Y} = \left[\left(X - X_s^i\right)^2 + \left(Y - Y_s^i\right)^2\right]^{-\frac{1}{2}}\left(Y - Y_s^i\right)$$
$$\frac{\partial \mathcal{H}_2^i}{\partial X} = \frac{\left(\dot{X} - \dot{X}_s^i\right)\sqrt{(X - X_s^i)^2 + (Y - Y_s^i)^2}}{(X - X_s^i)^2 + (Y - Y_s^i)^2} - \cdots$$
$$\frac{\left[(X - X_s^i)\left(\dot{X} - \dot{X}_s^i\right) + (Y - Y_s^i)\left(\dot{Y} - \dot{Y}_s^i\right)\right]\left[(X - X_s^i)^2 + (Y - Y_s^i)^2\right]^{-\frac{1}{2}}(X - X_s^i)}{(X - X_s^i)^2 + (Y - Y_s^i)^2}$$
$$\frac{\partial \mathcal{H}_2^i}{\partial \dot{X}} = \frac{(X - X_s^i)\sqrt{(X - X_s^i)^2 + (Y - Y_s^i)^2}}{(X - X_s^i)^2 + (Y - Y_s^i)^2}$$
$$\frac{\partial \mathcal{H}_2^i}{\partial Y} = \frac{\left(\dot{Y} - \dot{Y}_s^i\right)\sqrt{(X - X_s^i)^2 + (Y - Y_s^i)^2}}{(X - X_s^i)^2 + (Y - Y_s^i)^2} - \cdots$$
$$\frac{\left[(X - X_s^i)\left(\dot{X} - \dot{X}_s^i\right) + (Y - Y_s^i)\left(\dot{Y} - \dot{Y}_s^i\right)\right]\left[(X - X_s^i)^2 + (Y - Y_s^i)^2\right]^{-\frac{1}{2}}(Y - Y_s^i)}{(X - X_s^i)^2 + (Y - Y_s^i)^2}$$
$$\frac{\partial \mathcal{H}_2^i}{\partial \dot{Y}} = \frac{(Y - Y_s^i)\sqrt{(X - X_s^i)^2 + (Y - Y_s^i)^2}}{(X - X_s^i)^2 + (Y - Y_s^i)^2}$$
$$\frac{\partial \mathcal{H}_3^i}{\partial X} = -\frac{1}{1 + \left(\frac{Y - Y_s^i}{X - X_s^i}\right)^2}\frac{Y - Y_s^i}{(X - X_s^i)^2}$$
$$\frac{\partial \mathcal{H}_3^i}{\partial Y} = \frac{1}{1 + \left(\frac{Y - Y_s^i}{X - X_s^i}\right)^2}\frac{1}{(X - X_s^i)}$$

The rest of the partial derivatives are 0 (no dependencies). We note that the measurement matrix depends on both time (as the nominal solution is time dependent), and which station is making the measurement (since the stations are at different locations). The value of this matrix at $k = 0$ ($t = 0$) and station $i = 1$ is:

$$\tilde{C}^1\Big|_{\text{nom},t=k=0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0.0242 & 0 \\ 0 & 0 & 0.0033 & 0 \end{bmatrix}$$

1.2 DT Linearization about Nominal Condititions

We have the Eulerized transition and measurement matrices:

$$\tilde{F}_k = I + \Delta t\ \tilde{A}\Big|_{\text{nom}}$$
$$\tilde{G}_k = \Delta t\ \tilde{B}\Big|_{\text{nom}}$$
$$\tilde{\Omega}_k = \Delta t \Gamma$$
$$\tilde{H}_k^i = \tilde{C}^i\Big|_{\text{nom}}$$

With $\Delta t = 10$ seconds, we can write the full expressions for the DT linearized dynamics and measurement model:

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

The values for the matrices are:

$$\tilde{F}_0 = \begin{bmatrix} 1.0000 & 10.0000 & 0 & 0 \\ 0.0000 & 1.0000 & 0 & 0 \\ 0 & 0 & 1.0000 & 10.0000 \\ 0 & 0 & -0.0000 & 1.0000 \end{bmatrix}$$

$$\tilde{G}_k = \begin{bmatrix} 0 & 0 \\ 10 & 0 \\ 0 & 0 \\ 0 & 10 \end{bmatrix}$$

$$\tilde{\Omega}_k = \begin{bmatrix} 0 & 0 \\ 10 & 0 \\ 0 & 0 \\ 0 & 10 \end{bmatrix}$$

$$\tilde{H}_0^1 = \begin{bmatrix} 1.0000 & 0 & 0 & 0 \\ 0 & 1.0000 & 0.0242 & 0 \\ 0 & 0 & 0.0033 & 0 \end{bmatrix}$$

1.3 Linearized DT Simulation

Now that we have a discretized linearized simulator, we can check the results it produces vs. the results a full nonlinear simulation would produce, with a perturbation of $\delta x_0 = [0, 0.075, 0, -0.021]$:

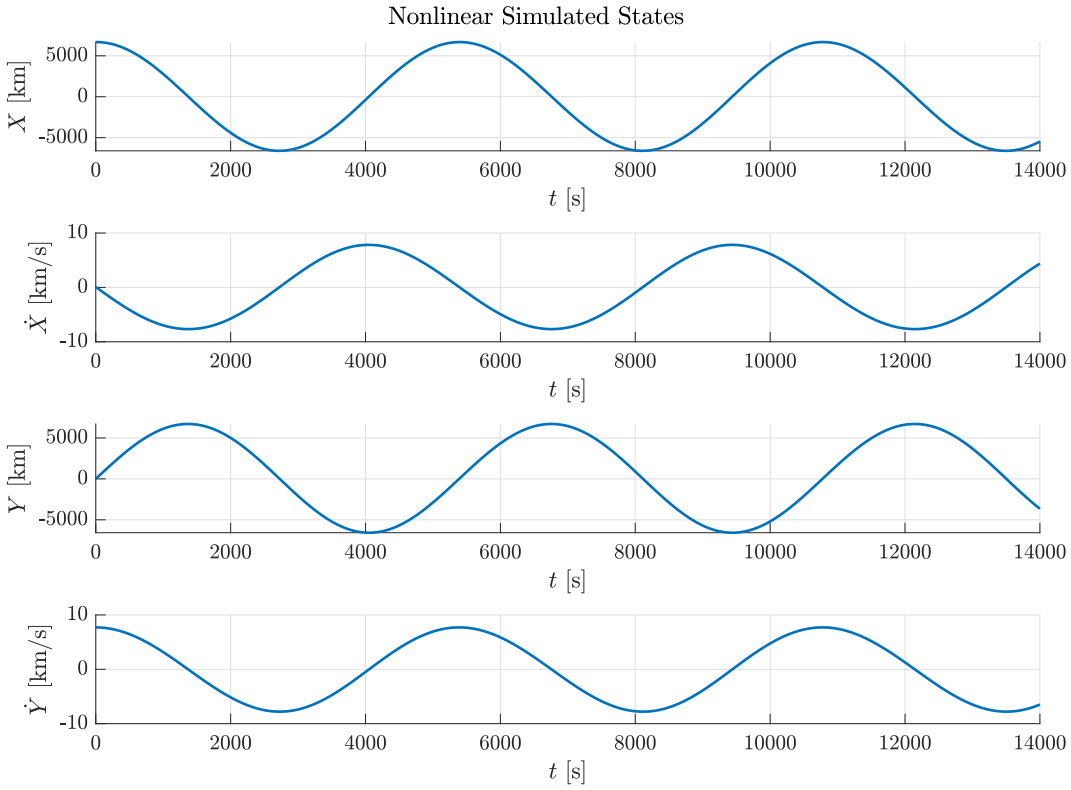


Figure 1: Nonlinear State Simulation

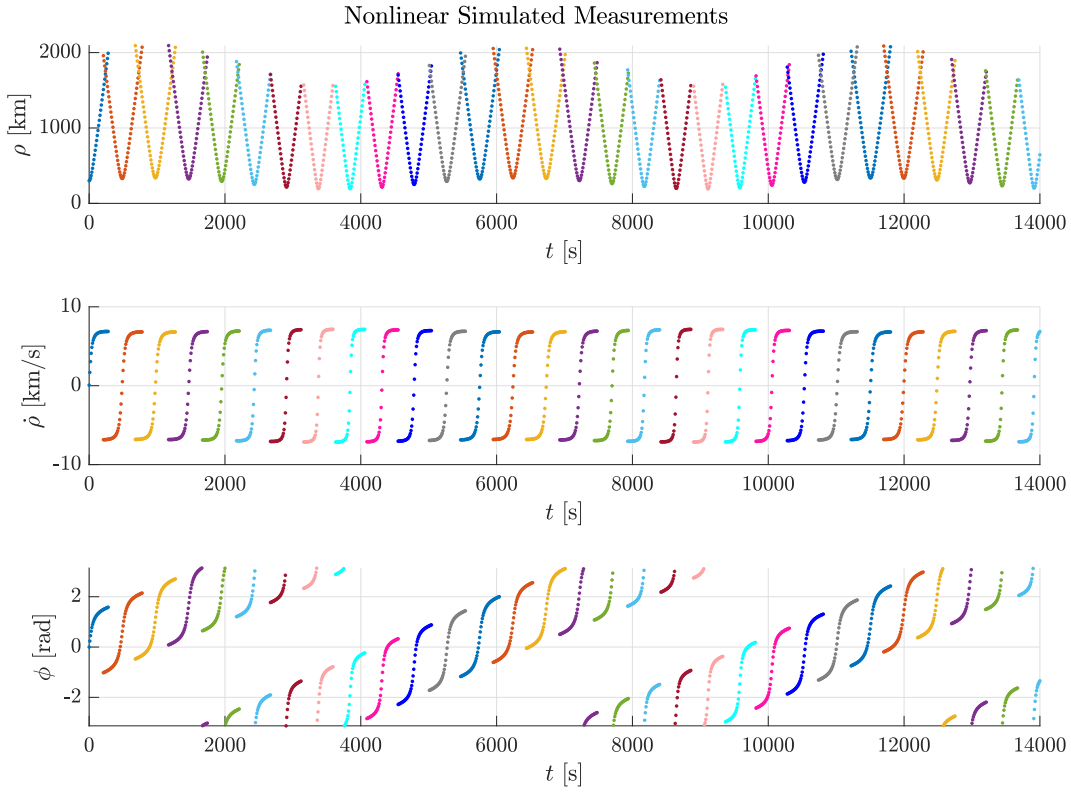


Figure 2: Nonlinear Measurement Simulation

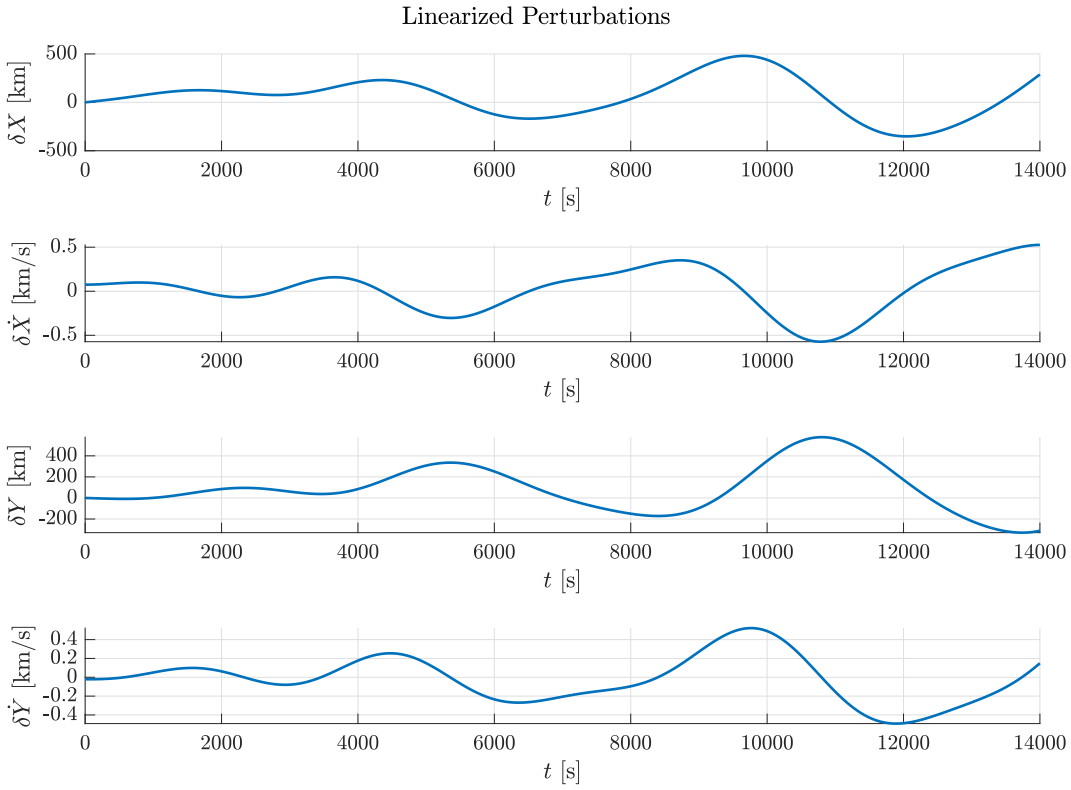


Figure 3: Linear Perturbation Simulation

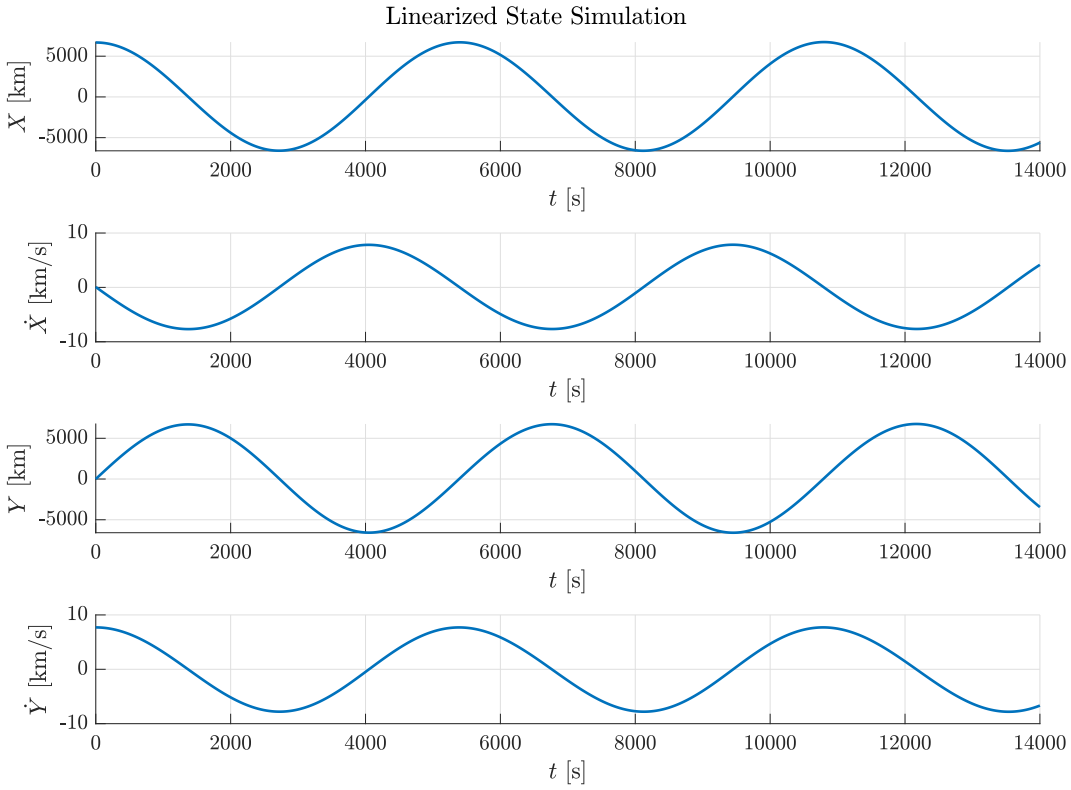


Figure 4: Linear State Simulation

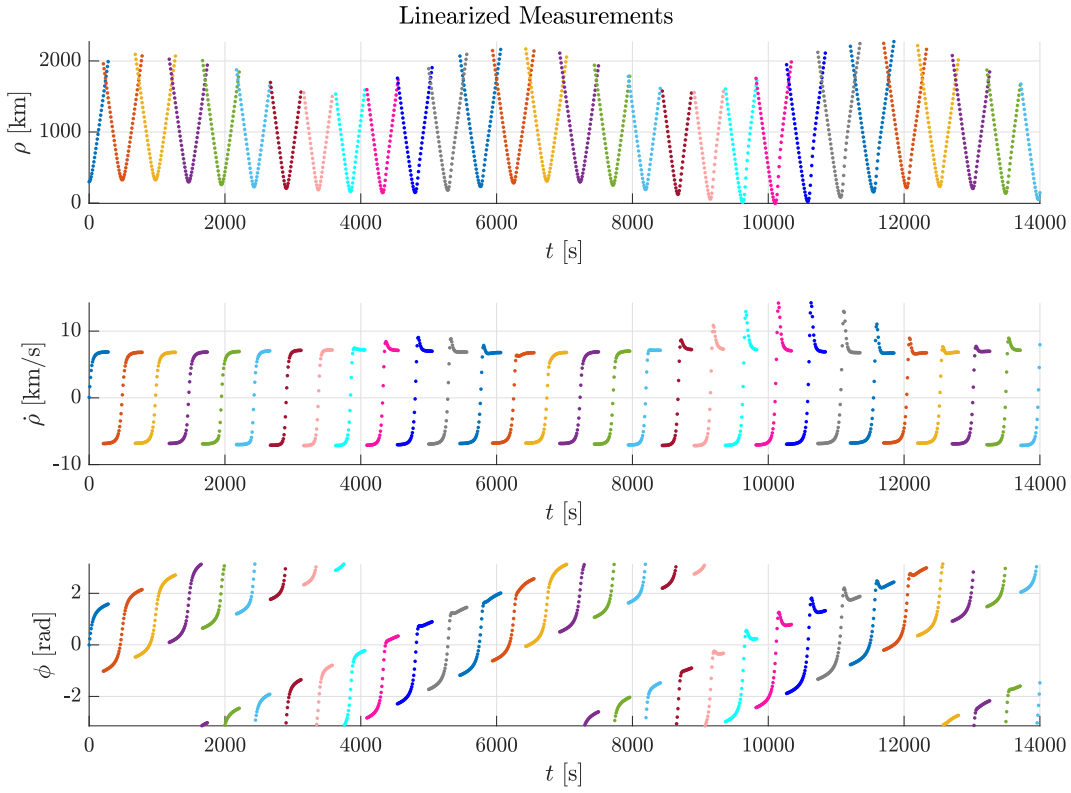


Figure 5: Linear Measurement Simulation

Both the nonlinear and linear models match the solutions given for part one, so the linearization process went well. The linear model lines up closely at the start, but drifts away as time progresses.

2 Stochastic Nonlinear Filtering

2.1 Linearized Kalman Filter

We can follow the process of the linearized Kalman Filter to model the perturbation from the nominal trajectory. We expect the LKF to work decently near the nominal trajectory, but for it to diverge when away from the nominal trajectory. For this reason, we will choose perturbations which keep the total energy of the orbit roughly the same. For instance, choosing a large initial radial velocity perturbation will not change the energy as much as choosing the same tangential velocity perturbation, as nominally 100% of the velocity is in the tangential direction and any changes in velocity at higher velocities require more energy than changes in velocity at a low velocity. For this reason, we will perturb X_0, Y_0, and $\dot{X}_0$ but we will choose a small $\dot{Y}_0$. We have $\delta x_0 = [10, 0, 10, 0]^{\mathrm{T}}$ and $P_0 = \mathrm{diag}\left([10, 0.01, 10, 1\mathrm{e}-8]\right)$. The following shows plots of the nominal trajectory and simulated truth measurements for one Monte Carlo run:

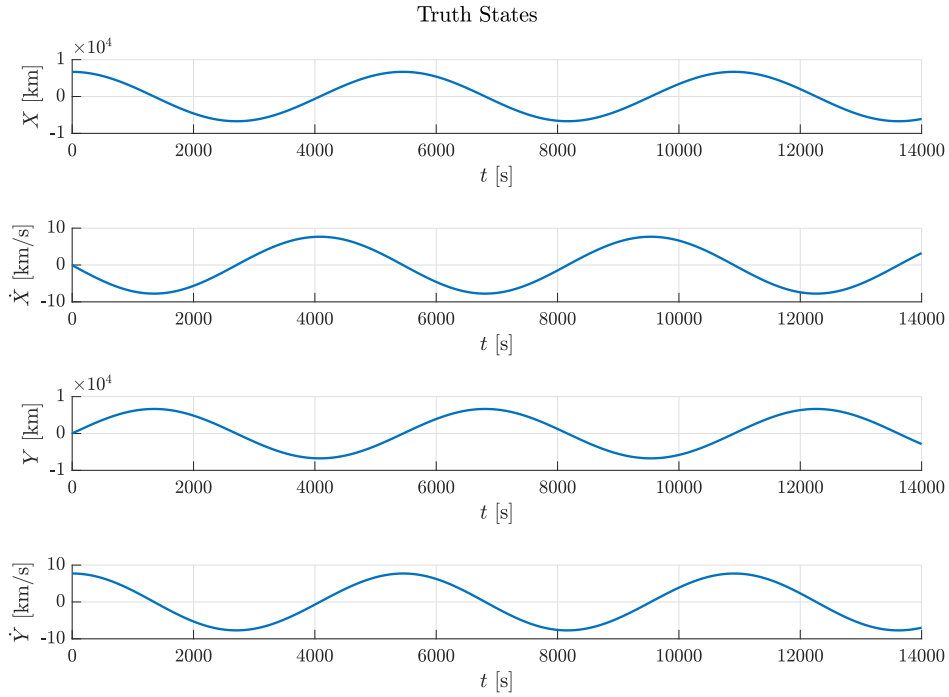


Figure 6: Sample Monte Carlo Truth State

The truth model lines up quite closely with the nominal trajectory at first, but since the dynamics are linearized about the nominal trajectory, they are bound to diverge as time progresses. The following plot shows the truth perturbations from nominal as a function of time:

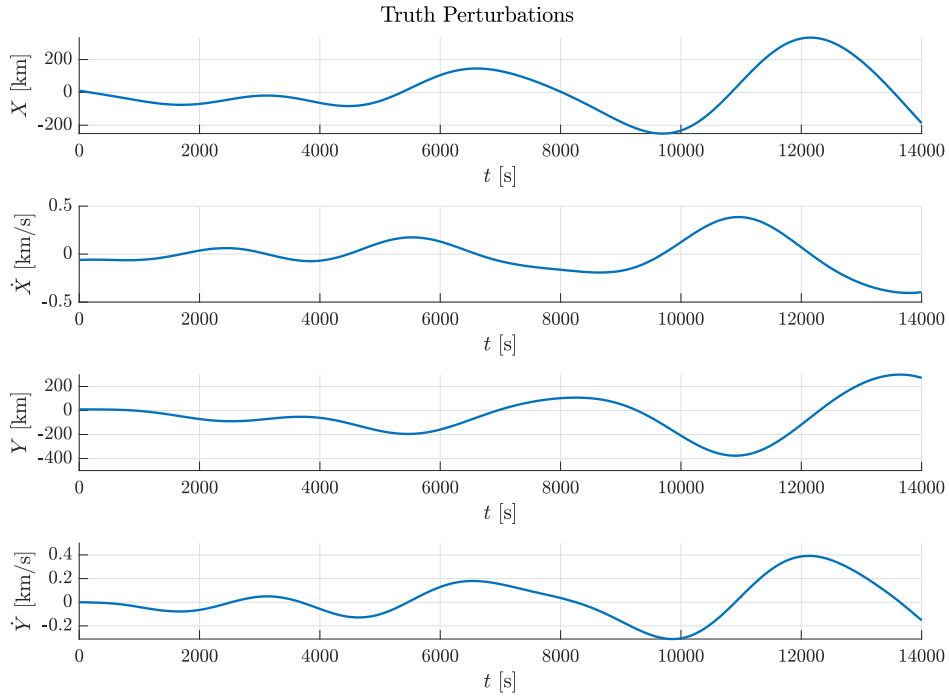


Figure 7: Sample Monte Carlo Truth Perturbations

We can also plot the truth measurements as well as the simulated measurement perturbations as well:

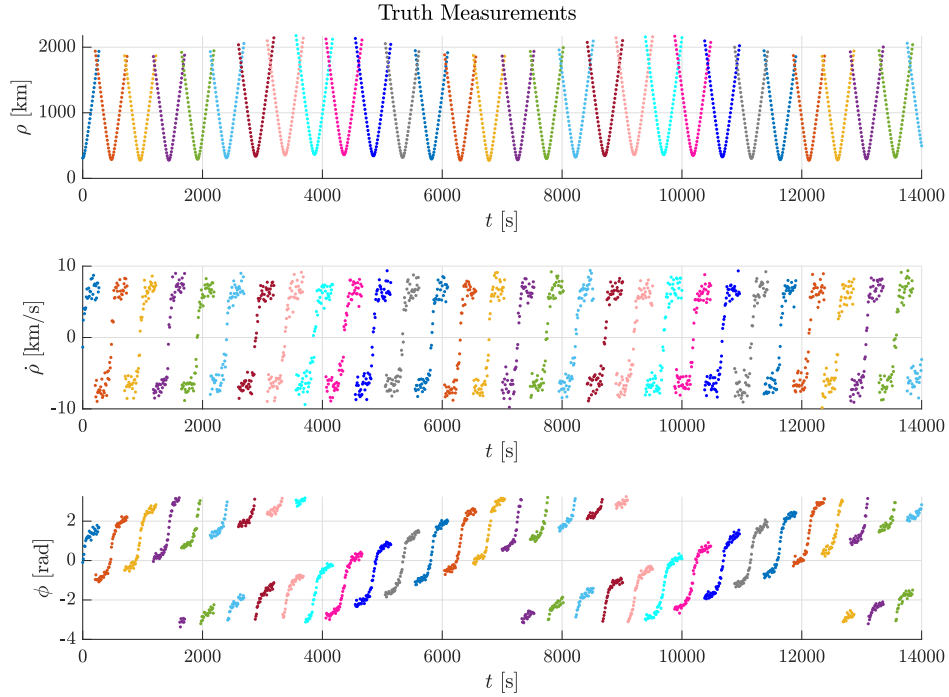


Figure 8: Sample Monte Carlo Truth Measurements

And the simulated measurement perturbations:

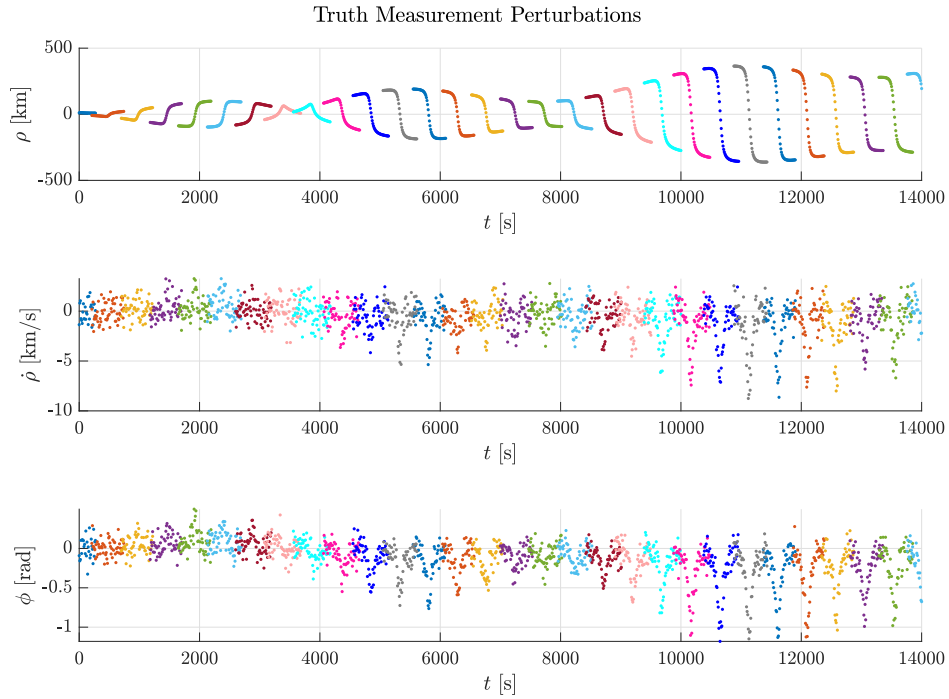


Figure 9: Sample Monte Carlo Truth Measurement Perturbations

Even with such a small initial perturbation, the perturbations grow to 100 km with $\rho/X/Y$ within 2000 seconds (less than half of an orbital period). We can now move on to filtering the noisy truth perturbations. The Q matrix used for the filter was actually a time varying logarithmically scaled matrix $3\text{diag}\left([1, 1e-3, 1, 1e-3]\right)$ from $10\text{e}-3$ to 10e1 from 0 to 2000 seconds. The filter really only holds together for the first 2000 seconds before the perturbations grow too large. Here is a plot of the state errors, with the 2σ bounds plotted alongside:

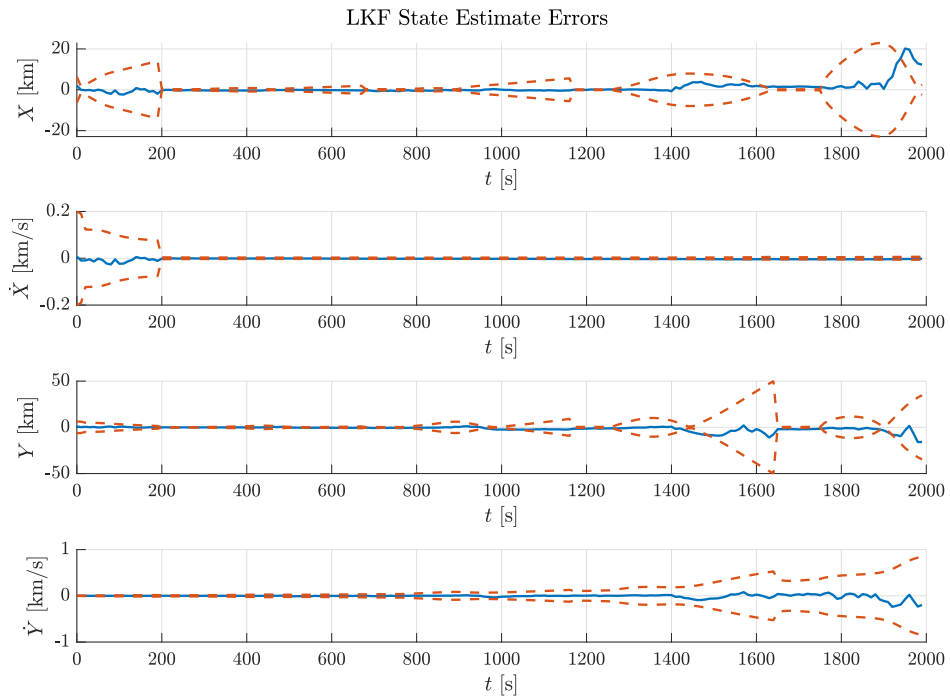


Figure 10: Sample LKF State Errors

We can plot the measurement innovations as well:

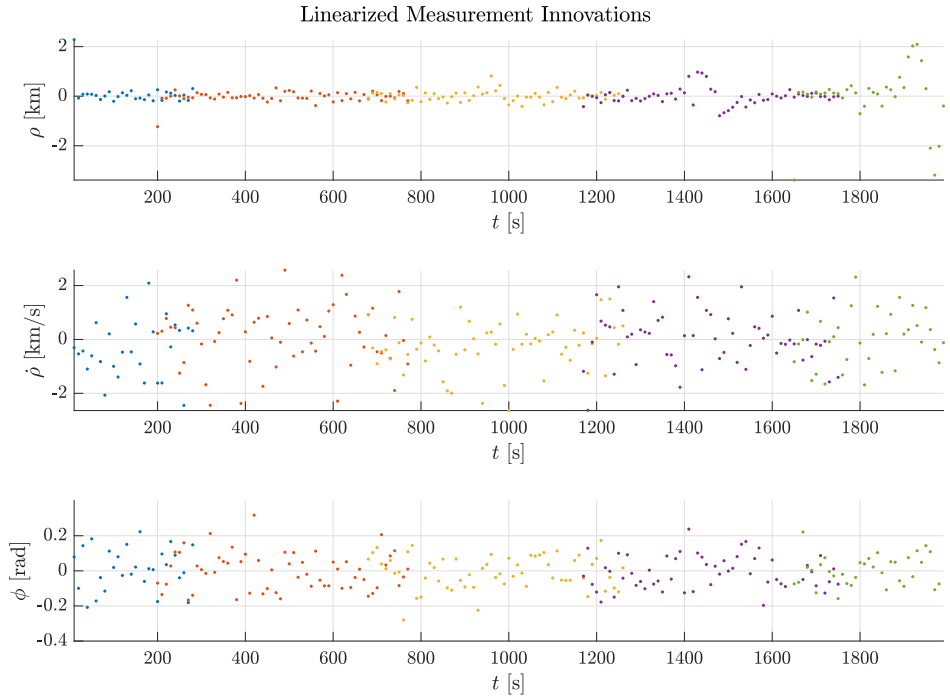
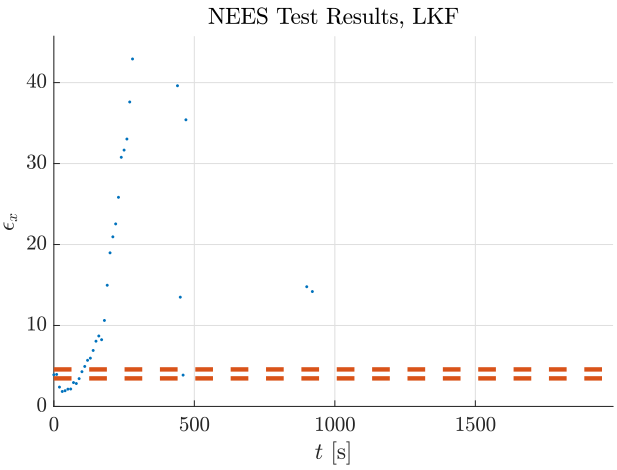


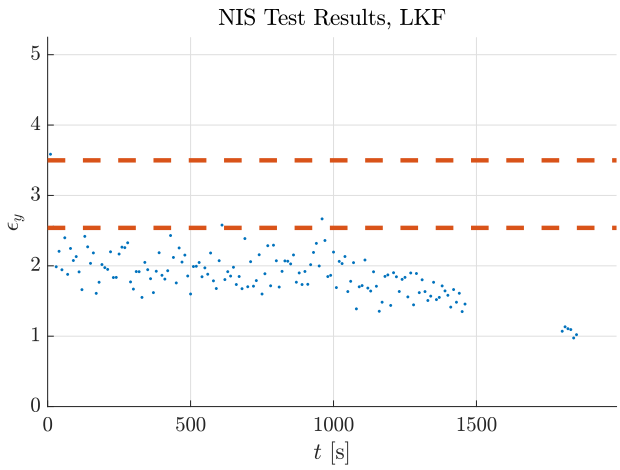
Figure 11: Sample LKF Measurement Innovations

At least from the first 2000 seconds, it seems like the measurement innovations follow a Gaussian spread about a zero mean. There are larger trends that occur at 1400 and 1900

seconds, and this is likely due to a combination of the perturbations getting larger as well as switching ground stations which adds to the error due to linearization. We can then plot the NEES and NIS statistics:



(a) LKF Tuned NEES Test



(b) LKF Tuned NIS Test

The actual process of narrowing down the process noise covariance was more of a guessing game than that of the EKF listed below, especially since no matter which covariance we pick the NEES test explodes to an average on the order of 10^7 to 10^13 (NIS averages 1.847 when the large spikes are removed). Initially, we tried using a 2×2 covariance and multiplying by $\tilde{\Omega}_k$, but it looked like the filter's covariance bounds were greater for the velocity measurements than the position measurements. We ended up going with a 4×4 diagonal covariance matrix so we could reduce the diagonal terms corresponding to the velocity measurements. The reason we opted for the time varying process noise covariance was because the state will drift away from the nominal trajectory, at which point the linearized dynamics become more and more unreliable. The reason for the logarithmically distributed scaling factor was because the filter performs quite well at first but then very quickly sees large spikes. This can be seen in the full 14000 second run of the filter:

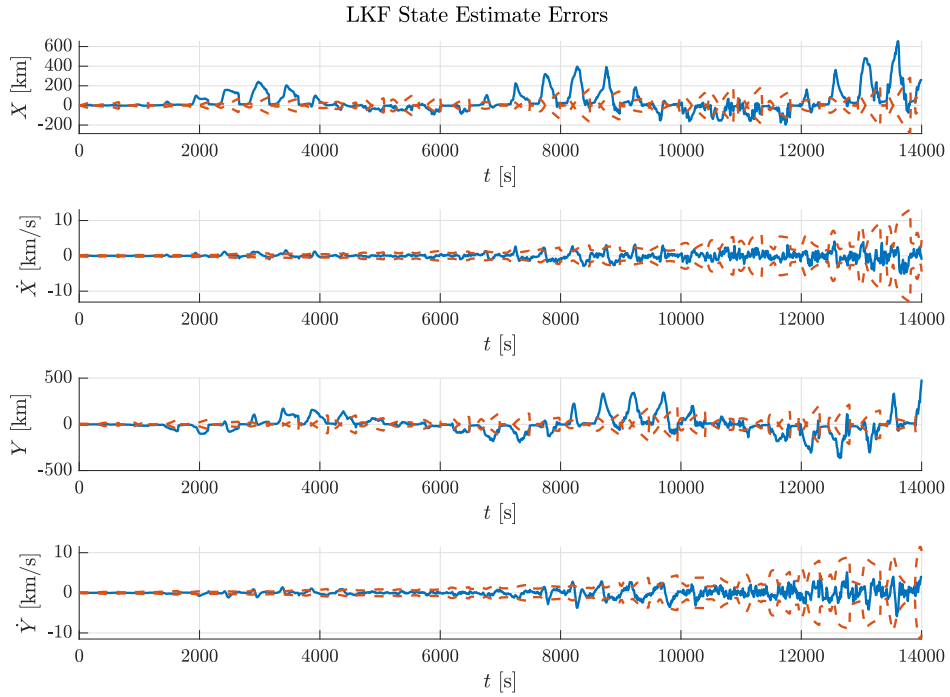


Figure 13: Full LKF State Errors

What's interesting to note here is that despite the large error spikes, the error seems to return back to centering around zero. This is reflected in the full NEES test as well:

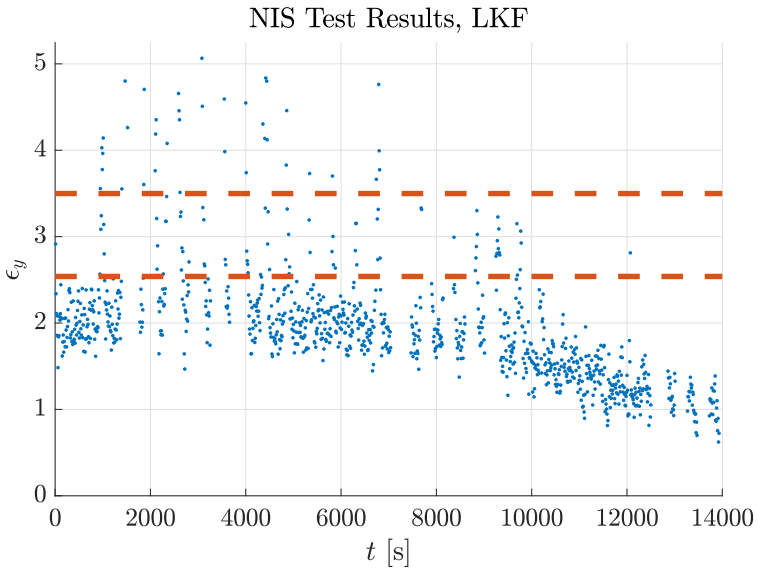


Figure 14: Full LKF NIS

We can see that there are many instances in this NIS chart without a data point, and that's because the NIS value goes up to the NEES level orders of magnitude before coming back to a reasonable value. This behavior suggests that the filter, as the perturbations increase, increasingly tries to fit the measurements with less and less regard to the actual dynamics of

the system. This is also why the state estimation errors are able to stay more or less centered around zero, as more and more weight is given to the measurements via the Kalman gain as P as a function of Q increases.

2.2 Extended Kalman Filter

We can follow the process of the extended Kalman filter on the same dataset produced by the Monte Carlo simulations run for the linearized Kalman filter. The only difference is that the data passed in is the full data (not perturbations), and the propagation of the states is done via Runge Kutta 4/5 via `ode45` in MATLAB® , with the measurements taken using the actual nonlinear measurement equations. Quantities such as the Kalman gain and the error covariance are taken by linearizing to get the matrix quantity $\tilde{H}$, but this is done again about the current state estimate and not any nominal trajectory. The EKF is a little more robust than the LKF, so we can use much larger perturbations which the EKF can handle without issue. These simulations are initialized about a perturbation of $[100, 0.2, -100, 0.2]$ with a covariance of diag $([10, 0.2, 10, 0.2])$. The following are sample truth model states and measurements sampled from the perturbation and covariance listed:

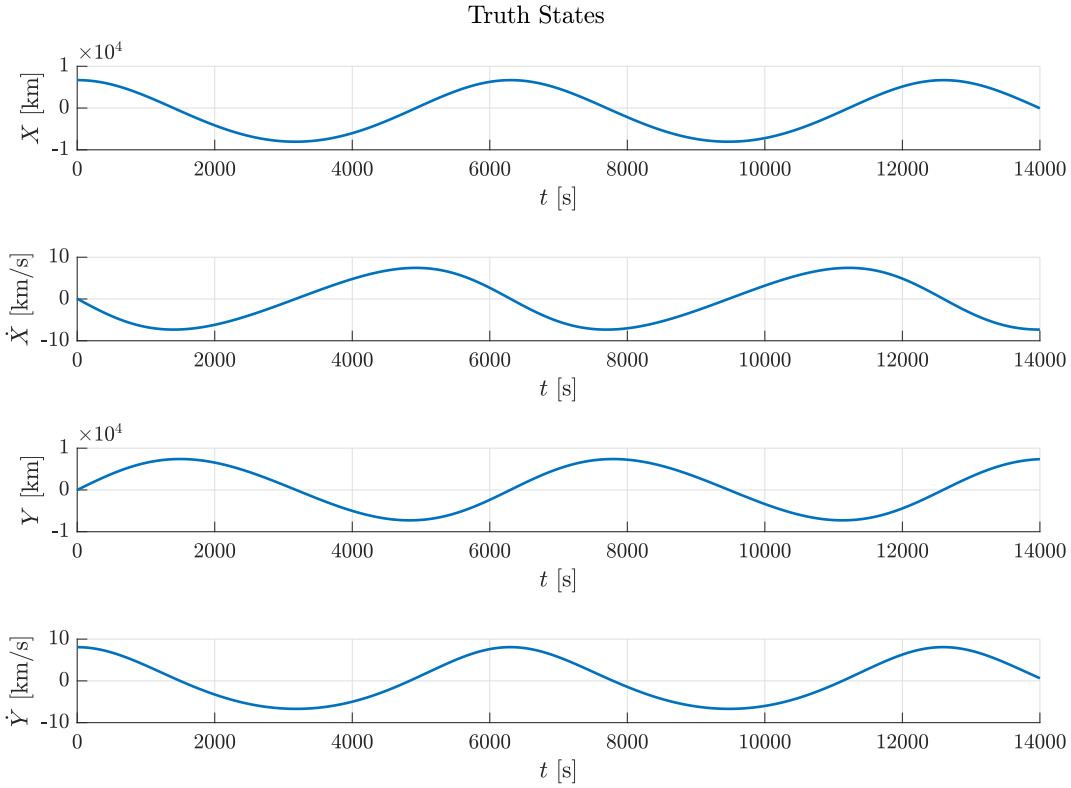


Figure 15: EKF Truth States

These are the truth states we would expect of an orbit with a nonzero eccentricity, which makes sense. We can also feed these through the measurement equations and add noise to get:

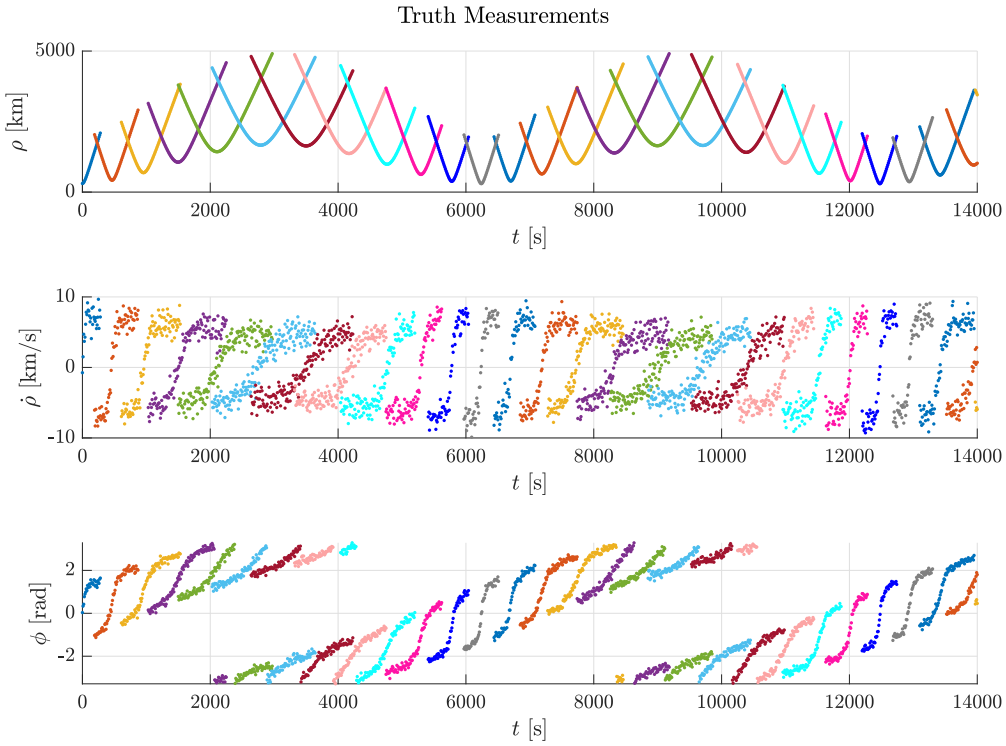


Figure 16: EKF Truth Measurements

The measurement noise isn't really noticeable on the range measurements (0.1 km variance vs 1000 km range measurements), but the noise is very noticeable on the range rate measurements and slightly noticeable on the bearing measurements. Additionally, due to the high eccentricity of the orbit, there are instances where many ground stations can see the orbiter and instances where only one ground station can see the orbiter. With these conditions, it is likely that some tests contain instances where no groundstation is visible. The following plot shows the state estimation errors generated along with the measurement innovations generated by the EKF:

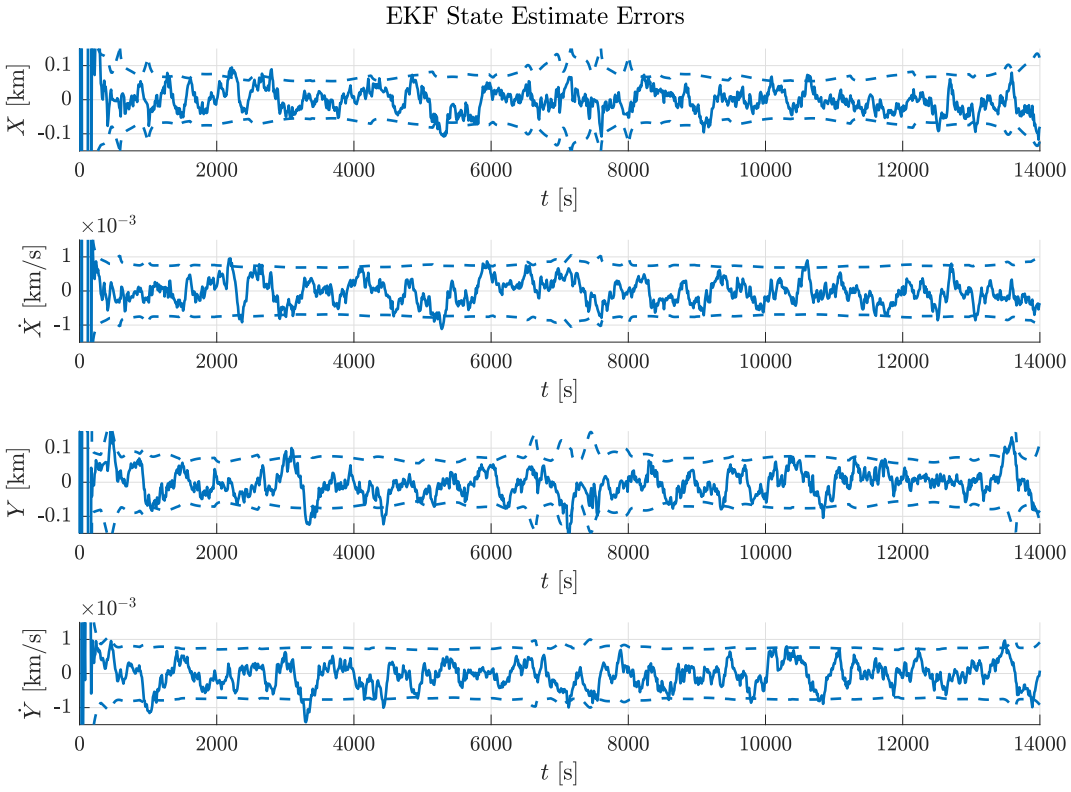


Figure 17: EKF State Estimation Errors

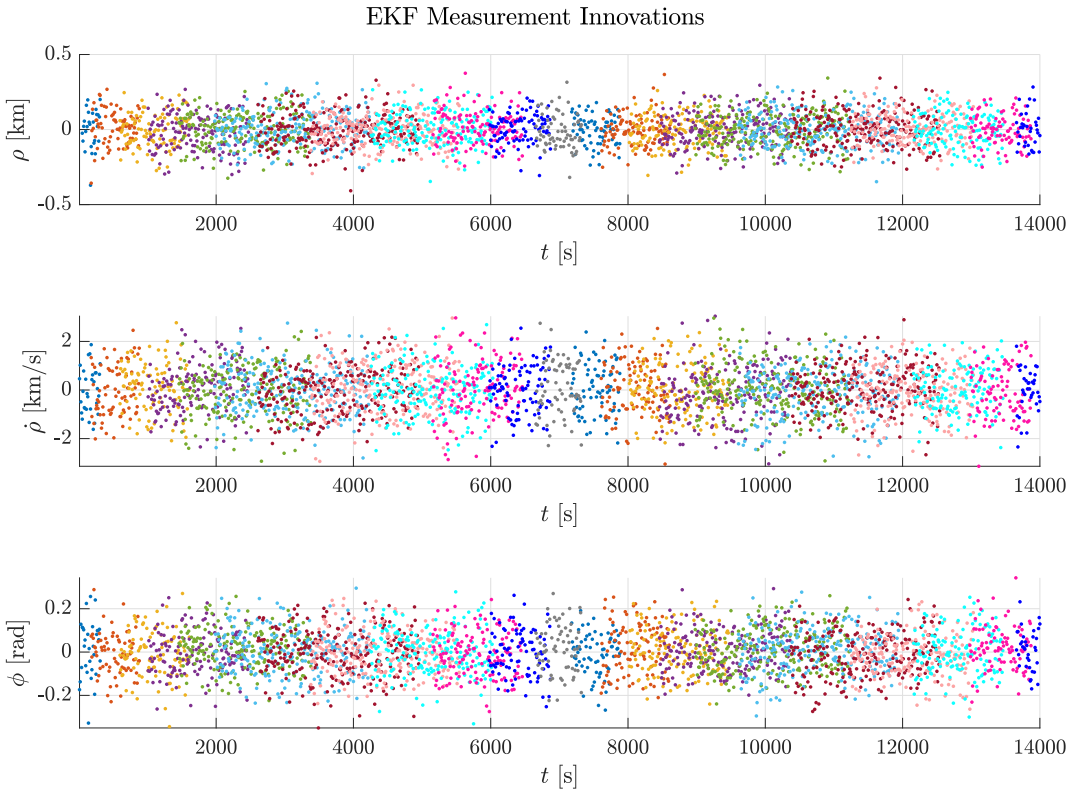
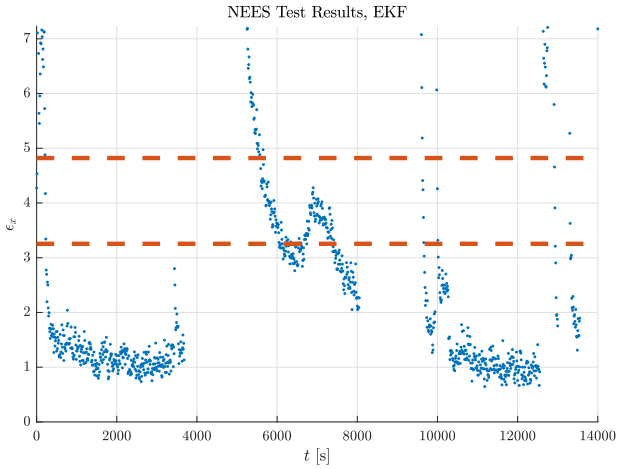


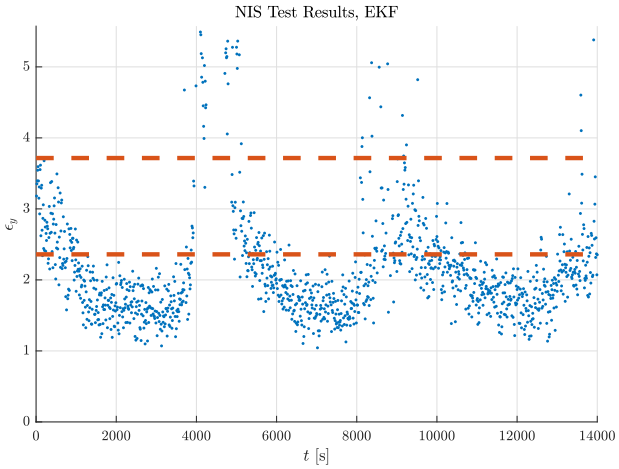
Figure 18: EKF Measurement Innnovations

From the above figures, we can see that the state estimation errors are visually unbiased and seem to fall within the 2σ bounds most of the time. The measurement innovations are also precisely what we would expect from the covariance they were generate from. The 2σ bounds for the range, range rate, and bearing are 0.2 km, 2 km/s, and 0.2 radians respectively, and this is what we see.

To tune this EKF, knowing that in orbit, the process noise we would see would come in the form of accelerations such as solar radiation pressure, drag, or gravitational anomalies, we would expect the process noise in its most basic form to be a matrix with variances for the acceleration terms. We also expect, since the magnitude of these perturbations are very small, that the actual process noise covariance will be small too. We started with a process noise covariance of $1e-7 \cdot I_{2\times 2}$:

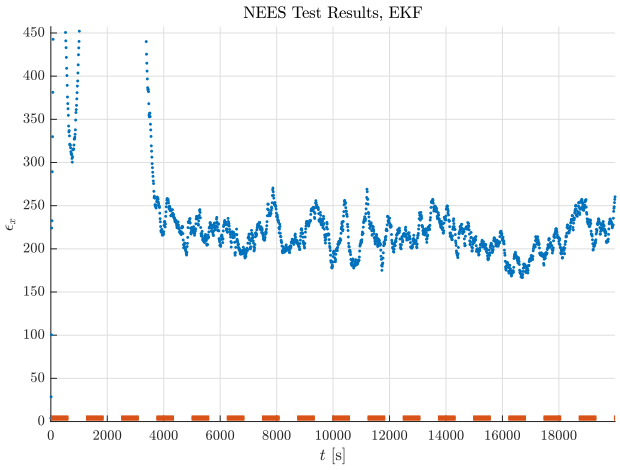


(a) EKF Untuned NEES Test

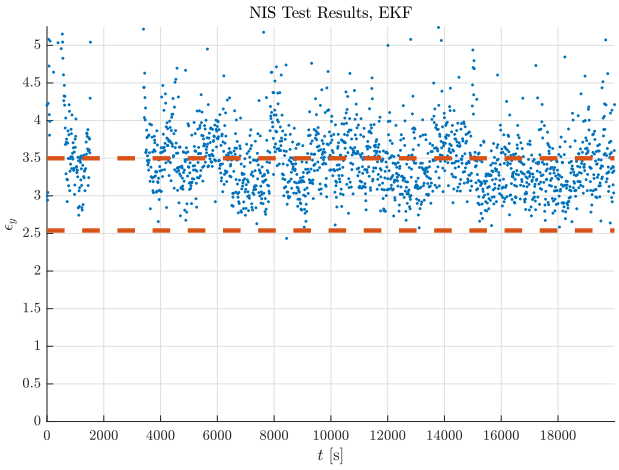


(b) EKF Untuned NIS Test

We can see that the NEES/NIS statistic points lie typically under the χ^2 bounds, so we tuned the filter by just walking the exponent down around until we got something that fit inside the lines. Here's an example with the process noise covariance set to $1e-12 \cdot I_{2\times 2}$:

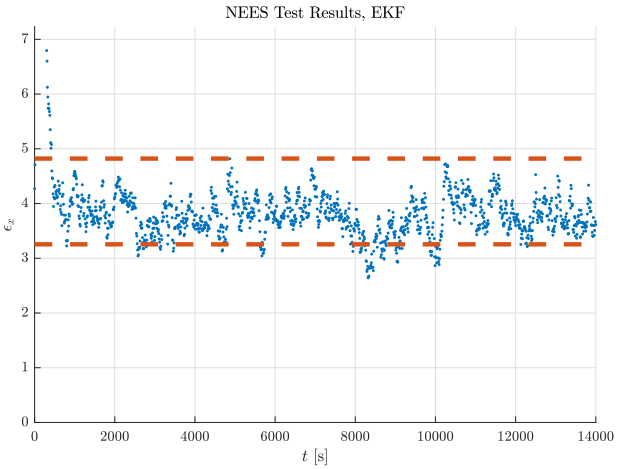


(a) EKF Tuned NEES Test

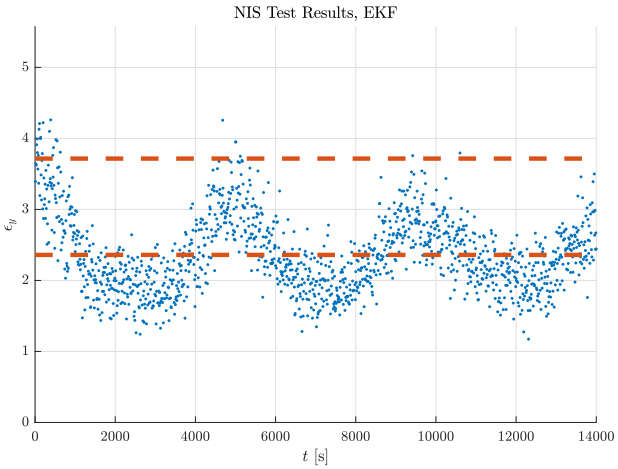


(b) EKF Tuned NIS Test

Here, we see that the points lie above our desired region. We walked the exponent around some more and finally ended up with $1e-10 \cdot I_{2\times 2}$ (yes, this is the original process noise covariance matrix). We can plot the results here:



(a) EKF Tuned NEES Test



(b) EKF Tuned NIS Test

While visually the EKF passes the NEES test, it seems to be underconfident with the measurement innovations (although they are pretty close). This is likely due to the fact that the system is a nonlinear system, and even though the noise that is added in is Gaussian, the nonlinearities in the system could result in stranger patterns in the data (for example, the sinusoidal pattern in the NIS test results). The results for the NEES test is a χ^2 of 4.003, which is within the bounds set by $\alpha = 0.05$ for four degrees of freedom (3.255, 4.821). The results for the NIS test is a χ^2 of 2.42, which is within the bounds set by $\alpha = 0.05$ for three degrees of freedom (2.359, 3.716). We can say that the EKF is dynamically consistent to a 0.05 significance level.

3 Filtering Data

We can then run both filters on the data provided in the problem statement. The following is a plot of the estimated states from the LKF:

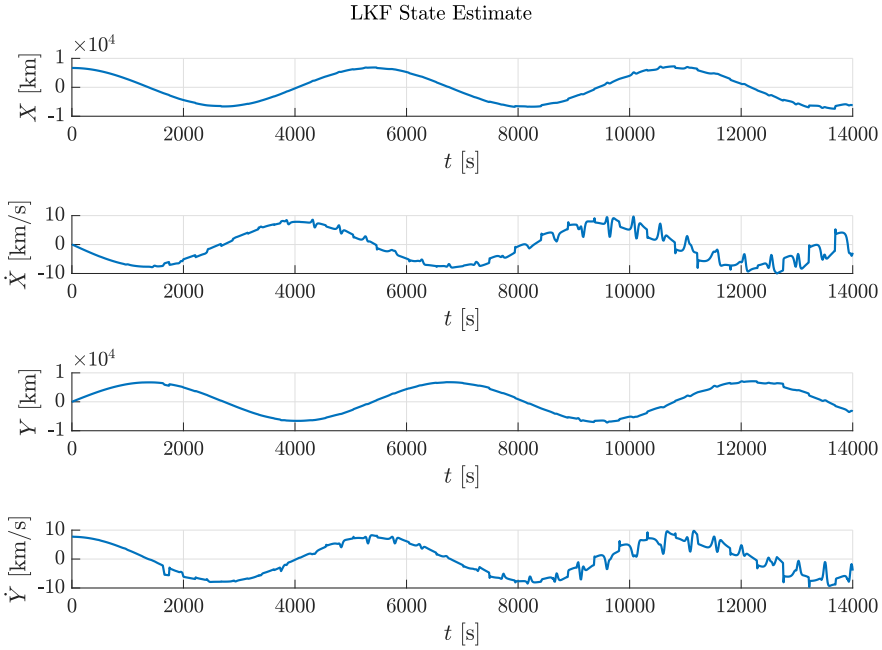


Figure 22: LKF State Estimate

Here are the 2σ bounds of the LKF filter:

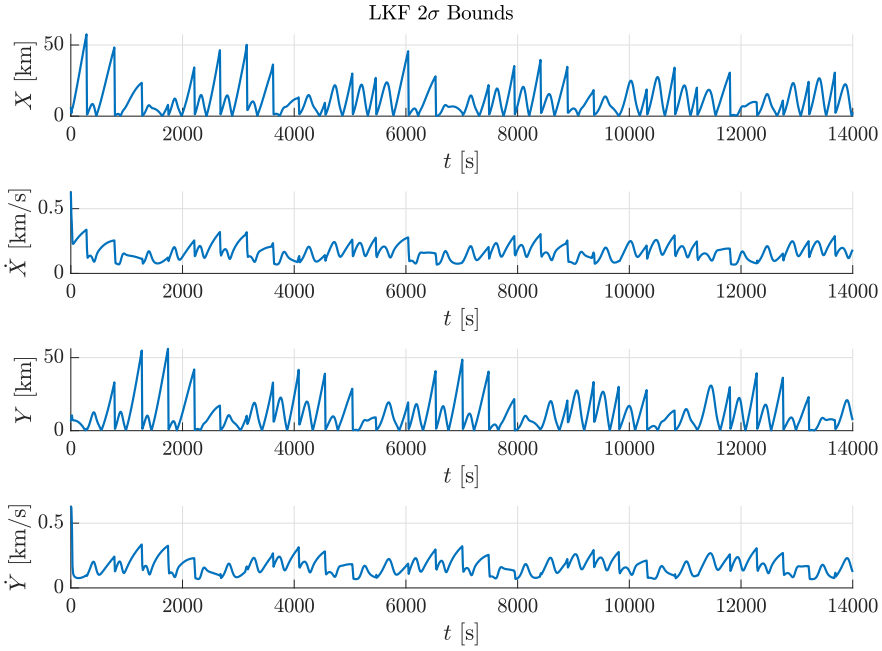


Figure 23: LKF 2σ Bounds

The following is a plot of the estimated states from the EKF:

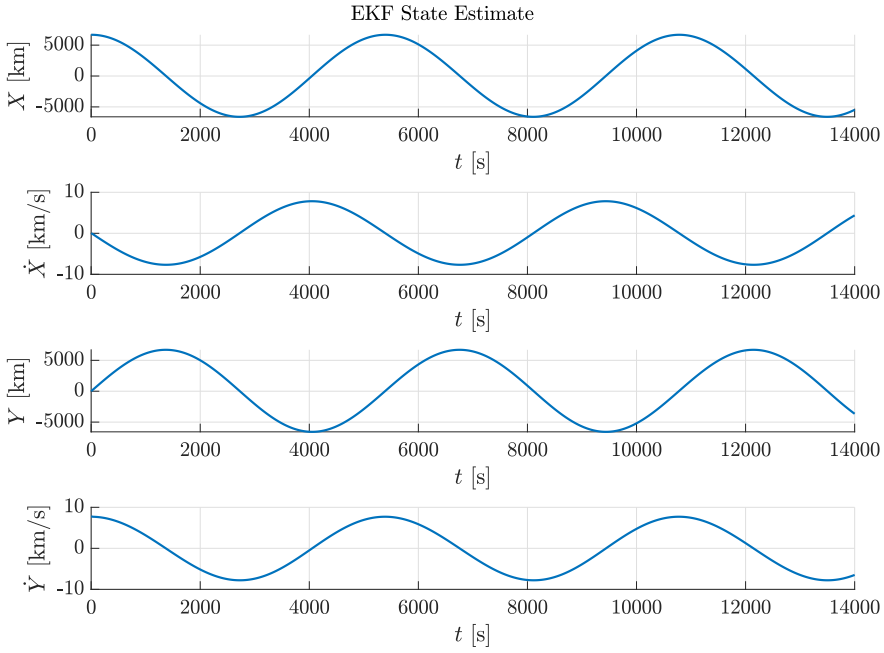


Figure 24: EKF State Estimate

And the 2σ bounds:

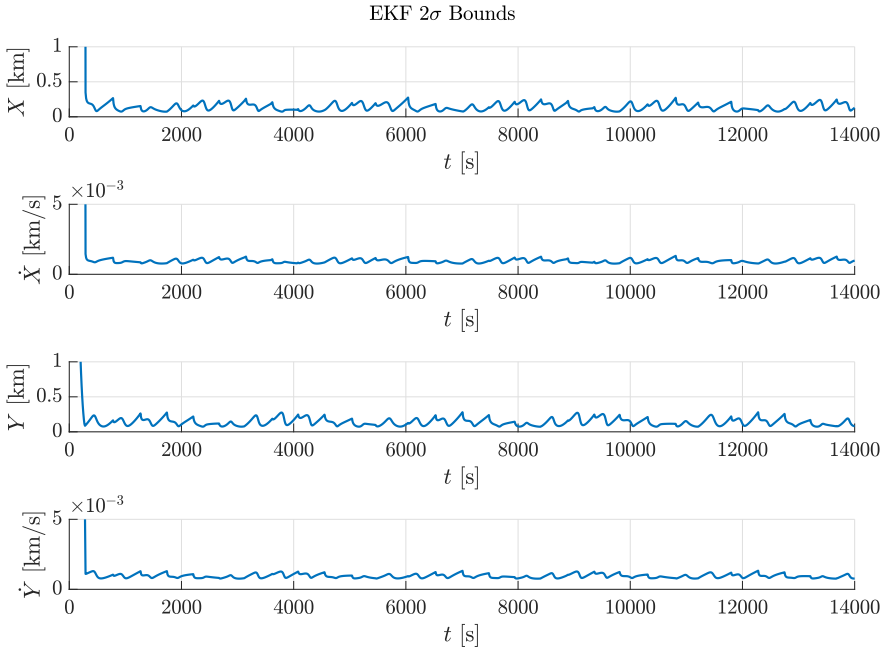


Figure 25: EKF 2σ Bounds

Both the LKF and the EKF estimate roughly the same trajectory, but that's about where the similarities end. In every other measure of filter performance, the EKF outperforms the

LKF ('smoothness' of estimate, dynamic consistency, lower covariances). While both filters linearize the matrix quantities for the update steps, the dependence on a nominal trajectory for the LKF means it is limited to cases where the states stay extremely close to the nominal trajectory, otherwise the filter breaks. The EKF on the other hand is able to estimate a wide range of quantities given the nonlinearities are relatively 'smooth' like they are in orbits. The EKF does the better job.

4 Appendix

4.1 MATLAB® Code

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

% ASEN 5044 Final Project
%
% Purpose:
%
% Author(s): Ian Thomas,
%
% Last Modified: 12/9/2020
%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


close all;
clear;
clc;

set(groot,'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');

set(0,'DefaultAxesLooseInset',[0,0,0,0])

colors = [0    0.4471    0.7412
          0.8510    0.3255    0.0980
          0.9294    0.6941    0.1255
          0.4941    0.1843    0.5569
          0.4667    0.6745    0.1882
          0.3020    0.7451    0.9333
          0.6353    0.0784    0.1843
          0.9882    0.6431    0.6431
          0.0588    1.0000    1.0000
          1.0000    0.0745    0.6510
          0     0     1
          0.5020    0.5020    0.5020];

format long

pos = [100, 100, 800, 600];
```

```
%% Part 1

% Declaration of Parameters
mu = 398600; % km^3/s^2
RE = 6378; % km

n = 4; % number of states
p = 3; % number of measurements

X0 = 6678; % km
Y0 = 0; % km
r0 = sqrt(X0^2+Y0^2); % km
dX0 = 0; % km/s
dY0 = r0*sqrt(mu/r0^3); % km/s

T = 2*pi*sqrt(r0^3/mu); % orbital period [s]
omega = 2*pi/T; % angular velocity of sc [rad/s]
omegaE = 2*pi/86400; % angular velocity of earth [rad/s]

Dt = 10; % time discretization [s]

nS = 12; % number of stations

% Nominal Solution
thetanom = @(t) omega * t;
Xnom = @(t) r0*cos(thetanom(t));
Ynom = @(t) r0*sin(thetanom(t));
dXnom = @(t) -r0*omega*sin(thetanom(t));
dYnom = @(t) r0*omega*cos(thetanom(t));
xnom = @(t) [Xnom(t), dXnom(t), Ynom(t), dYnom(t)];

Xi = @(t, i) RE*cos(omegaE*t+(i-1)*pi/6);
Yi = @(t, i) RE*sin(omegaE*t+(i-1)*pi/6);
dXi = @(t, i) -RE*omegaE*sin(omegaE*t+(i-1)*pi/6);
dYi = @(t, i) RE*omegaE*cos(omegaE*t+(i-1)*pi/6);
xi = @(t) [Xi(t), dXi(t), Yi(t), dYi(t)];

% Measurements
rho = @(t, x, i) sqrt((x(:,1)-Xi(t, i)).^2+(x(:,3)-Yi(t,i)).^2)
   ;
drho = @(t, x, i) ((x(:,1)-Xi(t,i)).*(x(:,2)-dXi(t,i))+...
    (x(:,3)-Yi(t,i)).*(x(:,4)-dYi(t,i))) ./ rho(t,x,i);
phi = @(t, x, i) atan2((x(:,3)-Yi(t,i)),(x(:,1)-Xi(t,i)));
mask = @(t, x, i) (absangdiff(atan2((x(:,3)-Yi(t,i)),(x(:,1)-Xi
   (t,i))),...
```

```
    atan2(Yi(t,i),Xi(t,i))) < (pi/2));
% mask = @(t, x, i) ~(abs(angdiff(atan2((x(:,3)-yi(t,i)),(x
   (:,1)-xi(t,i))),...
%     atan2(yi(t,i),xi(t,i)))) < (pi/2));

% Initial Perturbation
d_x0 = [0, 0.075, 0, -0.021]'; % perturbation

%% Nonlinear Simulation

% State
T = 1401;
tspan = (0:T-1)*10;
% tspan = [0, 14000];
x0_nl = [X0, dX0, Y0, dY0]' + d_x0;
tol = 1e-12;
opts = odeset('RelTol', tol);
[t_nl, x_nl] = ode45(@(t, S) nl_sim_nonoise(t, S, mu),...
    tspan, x0_nl, opts);


F_nl_states = figure('DefaultAxesPosition', [0.05, 0.05, 0.95,
   0.95]);
set(gcf, 'Position',  pos)
x_nl_ylabels = {'$X$ [km]','$\dot X$ [km/s]',...
    '$Y$ [km]','$\dot Y$ [km/s]'};
for i = 1:n
    subplot(n, 1, i);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(x_nl_ylabels{i})
    plot(t_nl, x_nl(:, i), 'LineWidth', 1.5)
    set(gca,'FontSize',12)
end
sgtitle('Nonlinear Simulated States')
saveas(gcf, '1', 'epsc')

rho_nl = rho(t_nl, x_nl, 1:nS);
drho_nl = drho(t_nl, x_nl, 1:nS);
phi_nl = phi(t_nl, x_nl, 1:nS);
mask_nl = mask(t_nl, x_nl, 1:nS);
mask_nl = permute(repmat(mask_nl, 1, 1, 3), [1, 3, 2]);

% make time first dimension, measurement index second dimension
```

```
   , and
% station index third dimension
y_nl = permute(cat(3, rho_nl, drho_nl, phi_nl), [1, 3, 2]);
y_nl(mask_nl) = NaN;

F_nl_meas = figure('DefaultAxesPosition', [0.05, 0.05, 0.95,
   0.95]);
set(gcf, 'Position',  pos)
y_nl_ylabels = {'$\rho$ [km]','$\dot \rho$ [km/s]','$\phi$ [rad
   ]'};
for j = 1:p
    ax = subplot(p, 1, j);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(y_nl_ylabels{j})
    for i = 1:nS
        plot(t_nl, y_nl(:, j, i), '.', 'Color', colors(i, :));
    end
    set(gca,'FontSize',12)
end
sgtitle('Nonlinear Simulated Measurements')
saveas(gcf, '2', 'epsc')


%% Linearized Simulation

% A tilde
A = @(a, r) [0, 1, 0, 0
          mu*(2*cos(a)^2-sin(a)^2)/r^3, 0, mu*(3*sin(2*a))/(2*r
             ^3), 0
          0, 0, 0, 1
          mu*(3*sin(2*a))/(2*r^3),0,mu*(2*sin(a)^2-cos(a)^2)/r
             ^3,0];
B = [0, 0; 1, 0; 0, 0; 0, 1]; % B tilde
Y = B; % Gamma tilde

% C tilde partial derivatives
C11 = @(t, x, i) (x(1)-Xi(t,i))/sqrt((x(1)-Xi(t,i))^2+(x(3)-Yi(
   t,i))^2);
C13 = @(t, x, i) (x(3)-Yi(t,i))/sqrt((x(1)-Xi(t,i))^2+(x(3)-Yi(
   t,i))^2);
C21 = @(t, x, i) (x(2)-dXi(t,i))*...
    sqrt((x(1)-Xi(t,i))^2+(x(3)-Yi(t,i))^2)/...
    ((x(1)-Xi(t,i))^2+(x(3)-Yi(t,i))^2) - ...
```

```
    ((x(1)-Xi(t,i))*(x(2)-dXi(t,i))+...
    (x(3)-Yi(t,i))*(x(4)-dYi(t,i)))*...
    (1/sqrt((x(1)-Xi(t,i))^2+(x(3)-Yi(t,i))^2)*...
    (x(1)-Xi(t,i)))/((x(1)-Xi(t,i))^2+(x(3)-Yi(t,i))^2);
C22 = @(t, x, i) (x(1)-Xi(t,i))*...
    sqrt((x(1)-Xi(t,i))^2+(x(3)-Yi(t,i))^2)/...
    ((x(1)-Xi(t,i))^2+(x(3)-Yi(t,i))^2);
C23 = @(t, x, i) (x(4)-dYi(t,i))*...
    sqrt((x(1)-Xi(t,i))^2+(x(3)-Yi(t,i))^2)/...
    ((x(1)-Xi(t,i))^2+(x(3)-Yi(t,i))^2) - ...
    ((x(1)-Xi(t,i))*(x(2)-dXi(t,i))+...
    (x(3)-Yi(t,i))*(x(4)-dYi(t,i)))*...
    (1/sqrt((x(1)-Xi(t,i))^2+(x(3)-Yi(t,i))^2)*...
    (x(3)-Yi(t,i)))/((x(1)-Xi(t,i))^2+(x(3)-Yi(t,i))^2);
C24 = @(t, x, i) (x(3)-Yi(t,i))*...
    sqrt((x(1)-Xi(t,i))^2+(x(3)-Yi(t,i))^2)/...
    ((x(1)-Xi(t,i))^2+(x(3)-Yi(t,i))^2);
C31 = @(t, x, i) -1/(1+((x(3)-Yi(t,i))/(x(1)-Xi(t,i)))^2)*...
    (x(3)-Yi(t,i))/(x(1)-Xi(t,i))^2;
C33 = @(t, x, i) 1/(1+((x(3)-Yi(t,i))/(x(1)-Xi(t,i)))^2)*...
    1/(x(1)-Xi(t,i));

% C tilde
C = @(t, x, i) [C11(t, x, i), 0, C13(t, x, i), 0
     C21(t, x, i), C22(t, x, i), C23(t, x, i), C24(t, x, i)
     C31(t, x, i), 0, C33(t, x, i), 0];

Fk = @(k) eye(n)+Dt*A(thetanom(k*Dt), r0);
Gk = Dt*B;
Omegak = Dt*Y;
Hk = @(k, i) C(k*Dt, xnom(Dt*k), i);


T = 1400;
d_x_lin = zeros(n, T+1);
d_xprev = d_x0;
k = (0:T)';

for i = 0:T
    d_x_lin(:, i+1) = d_xprev;
    d_xprev = Fk(i)*d_xprev;
end

d_x_lin = d_x_lin';
t_lin = k*Dt;
```

```
F_lin_delta = figure('DefaultAxesPosition', [0.05, 0.05, 0.95,
   0.95]);
set(gcf, 'Position',  pos)
d_x_lin_ylabels = {'$\delta X$ [km]','$\delta \dot X$ [km/s]',
   ...
    '$\delta Y$ [km]','$\delta \dot Y$ [km/s]'};
for i = 1:n
    subplot(n, 1, i);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(d_x_lin_ylabels{i})
    plot(t_lin, d_x_lin(:, i), 'LineWidth', 1.5)
    set(gca,'FontSize',12)
end
sgtitle('Linearized Perturbations')
saveas(gcf, '3', 'epsc')

x_nom = [Xnom(k*Dt), dXnom(k*Dt), Ynom(k*Dt), dYnom(k*Dt)];
x_lin = d_x_lin+x_nom;

F_lin_states = figure('DefaultAxesPosition', [0.05, 0.05, 0.95,
    0.95]);
set(gcf, 'Position',  pos)
x_lin_ylabels = {'$X$ [km]','$\dot X$ [km/s]',...
    '$Y$ [km]','$\dot Y$ [km/s]'};
for i = 1:n
    subplot(n, 1, i);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(x_lin_ylabels{i})
    plot(t_lin, x_lin(:, i), 'LineWidth', 1.5)
    set(gca,'FontSize',12)
end
sgtitle('Linearized State Simulation')
saveas(gcf, '4', 'epsc')

d_y_lin = zeros(T+1, nS, p);

for j = 0:T
    for i = 1:nS
        d_y_lin(j+1, i, :) = Hk(j, i)*d_x_lin(j+1, :)';
    end
```

```
end

rho_nom = rho(t_lin, x_nom, 1:nS);
drho_nom = drho(t_lin, x_nom, 1:nS);
phi_nom = phi(t_lin, x_nom, 1:nS);
ynom = permute(cat(3, rho_nom, drho_nom, phi_nom), [1, 3, 2]);

mask_lin = mask(t_lin, x_lin, 1:nS);
mask_lin = permute(repmat(mask_lin, 1, 1, 3), [1, 3, 2]);
d_y_lin = permute(d_y_lin, [1, 3, 2]);
y_lin = ynom + d_y_lin;
y_lin(mask_lin) = NaN;

% Linearized Measurements
F_lin_meas = figure('DefaultAxesPosition', [0.05, 0.05, 0.95,
   0.95]);
set(gcf, 'Position',  pos)
y_lin_ylabels = {'$\rho$ [km]','$\dot \rho$ [km/s]','$\phi$ [
   rad]'};
for j = 1:p
    ax = subplot(3, 1, j);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(y_lin_ylabels{j})
    if j == 3
        for i = 1:nS
            plot(t_lin, wrapToPi(y_lin(:, j, i)), '.', 'Color',
                colors(i, :));
        end
    else
        for i = 1:nS
            plot(t_lin, y_lin(:, j, i), '.', 'Color', colors(i,
                :));
        end
    end
    set(gca,'FontSize',12)
end
sgtitle('Linearized Measurements')
saveas(gcf, '5', 'epsc')

%% Part 2: Filtering

rng(100)
```

```
load orbitdeterm_finalproj_KFdata

% Define MC Simulation Parameters
n_trials = 100; % number of trials to generate
T = 1401; % a little over two orbital periods
k = (0:T-1)';
t = k*Dt;

% Generate Nominal Solution

x_nom = xnom(t)';
rho_nom = rho(t, x_nom', 1:nS);
drho_nom = drho(t, x_nom', 1:nS);
phi_nom = phi(t, x_nom', 1:nS);
mask_nom = mask(t, x_nom', 1:nS);
mask_nom = permute(repmat(mask_nom, 1, 1, 3), [3, 2, 1]);
y_nom = permute(cat(3, rho_nom, drho_nom, phi_nom), [3, 2, 1]);
mu0_nom = [X0, dX0, Y0, dY0]';

% Generate Monte Carlo TMT Data

% P0 = diag([100, 1, 100, 1]); % good trial for EKF
P0 = diag([10, 0.01, 10, 0.000000001]);
% P0 = diag([0, 0, 0, 0]);
d_x0 = [10, 0.00, 10, 0.0]';
mu0 = [X0, dX0, Y0, dY0]'+d_x0;
u = [0, 0]';

x_TMT = zeros(n, T, n_trials);
y_TMT = zeros(p, nS, T, n_trials);
d_y_TMT = y_TMT;

fprintf('Running Monte Carlo TMT Simulations: ')
tic
for i = 1:n_trials

    S0 = mvnrnd(mu0, P0)';
%     S0 = mu0;
    w = mvnrnd([0; 0], Qtrue, T)';
%     w = mvnrnd([0; 0], zeros(2), T)';
    [~, x] = ode45(@(t, S) nl_sim(t, S, mu, u, w, Dt), t, S0,
       opts);

    rho_mc = rho(t, x, 1:nS);
    drho_mc = drho(t, x, 1:nS);
```

```
    phi_mc = phi(t, x, 1:nS);
    mask_mc = mask(t, x, 1:nS);
    mask_mc = permute(repmat(mask_mc, 1, 1, 3), [3, 2, 1]);

    % make time first dimension, measurement index second
       dimension, and
    % station index third dimension
    y_mc = permute(cat(3, rho_mc, drho_mc, phi_mc), [3, 2, 1]);
    y_mc(mask_mc) = NaN;
    v = permute(reshape(...
        mvnrnd(zeros(p, 1), Rtrue, T*nS), [T, nS, p]), [3, 2,
           1]);
    d_y_mc = y_mc - y_nom;
    y_mc = y_mc + v;
    d_y_mc = d_y_mc + v;

    x_TMT(:, :, i) = x';
    y_TMT(:, :, :, i) = y_mc;
    d_y_TMT(:, :, :, i) = d_y_mc;

end
toc

% Generate Perturbation Data
x_nom_n = repmat(x_nom, [1, 1, n_trials]);
y_nom_n = repmat(y_nom, [1, 1, 1, n_trials]);

d_x_TMT = x_TMT - x_nom_n;

d_phi = d_y_TMT(3, :, :, :);
d_phi = wrapToPi(d_phi);
d_y_TMT(3, :, :, :) = d_phi;

% Generate NEES/NIS bounds
alpha = 0.05;
r1_NEES = chi2inv(alpha/2, n_trials*n)/n_trials;
r2_NEES = chi2inv(1-alpha/2, n_trials*n)/n_trials;
r1_NIS = chi2inv(alpha/2, n_trials*p)/n_trials;
r2_NIS = chi2inv(1-alpha/2, n_trials*p)/n_trials;


% Truth State
F_truth_states = figure('DefaultAxesPosition', [0.05, 0.05,
   0.95, 0.95]);
set(gcf, 'Position',  pos)
```

```
x_lin_ylabels = {'$X$ [km]','$\dot X$ [km/s]',...
    '$Y$ [km]','$\dot Y$ [km/s]'};
for i = 1:n
    subplot(n, 1, i);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(x_nl_ylabels{i})
    plot(t, d_x_TMT(i, :, 1), 'LineWidth', 1.5)
    set(gca,'FontSize',12)
end
sgtitle('Truth Perturbations')
saveas(gcf, 'd_ts_LKF', 'epsc')

% Truth Measurements
F_truth_meas = figure('DefaultAxesPosition', [0.05, 0.05, 0.95,
    0.95]);
set(gcf, 'Position',  pos)
y_lin_ylabels = {'$\rho$ [km]','$\dot \rho$ [km/s]','$\phi$ [
   rad]'};
for j = 1:p
    ax = subplot(3, 1, j);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(y_lin_ylabels{j})
    for i = 1:nS
        plot(t, reshape(d_y_TMT(j, i, :, 1), T, []), '.',...
            'Color', colors(i, :));
    end
    set(gca,'FontSize',12)
end
sgtitle('Truth Measurement Perturbations')
saveas(gcf, 'd_tm_LKF', 'epsc')

trial_plot = 2;

%% Linearized Kalman Filter

Qmag = permute(logspace(-3, 1, T), [1, 3, 2]);
f = 1e3;
Qframe = repmat(f*diag([1, 1e-3, 1, 1e-3]), [1, 1, T]);
Qguess = Qmag.*Qframe;

% Qguess = 1e-10*eye(2);
```

```
% P0 = 1e-4*eye(4);

x_LKF = zeros(n, T, n_trials);
P_LKF = zeros(n, n, T, n_trials);
y_LKF = zeros(p, nS, T, n_trials);
S_LKF = zeros(p*nS, p*nS, T, n_trials);

fprintf('Running LKF over MC Simulations: ')
tic
for i = 1:n_trials
    d_y_i = d_y_TMT(:, :, :, i);
%     d_y_i(:, :, 400:1200) = NaN;
    [d_x_t, P_t, d_y_m, S_t] = LKF(...
        n, d_y_i, Qguess, Rtrue,...
        mu0-mu0_nom, P0, Fk, Omegak, Hk);
    x_LKF(:, :, i) = d_x_t;
    P_LKF(:, :, :, i) = P_t;
    y_LKF(:, :, :, i) = d_y_m;
    S_LKF(:, :, :, i) = S_t;
end
toc

x_LKF = x_LKF + x_nom_n;
y_LKF = y_LKF + y_nom_n;

% phi_LKF = y_LKF(3, :, :, :);
% phi_LKF = wrapToPi(phi_LKF);
% y_LKF(3, :, :, :) = phi_LKF;

e_x_LKF = x_TMT - x_LKF;
e_y_LKF = y_TMT - y_LKF;

% Run NEES/NIS Tests

NEES_LKF = zeros(T, n_trials);
NIS_LKF = NaN * zeros(T, n_trials);

fprintf('Running NEES/NIS on LKF Data: ')
tic
for i = 1:n_trials
    e_x_dat = e_x_LKF(:, :, i);
    P_dat = P_LKF(:, :, :, i);
    e_y_dat = e_y_LKF(:, :, :, i);
    S_dat = S_LKF(:, :, :, i);
```

```
    for j = 1:T

        % Run NEES Calculation
        e_x = e_x_dat(:, j);
        eps_x = e_x' * (P_dat(:, :, j) \ e_x);
        NEES_LKF(j, i) = eps_x;

        % Run NIS Calculation
        S_i = S_dat(:, :, j);
        S_inds = find(~isnan(S_i(1, :)), 1, 'last');
        S_i = S_i(1:S_inds, 1:S_inds);
        e_y = e_y_dat(:, :, j);
        e_y = e_y(:);
        e_y = e_y(~isnan(e_y));
        [r, ~] = size(S_i);
        eps_y = NaN;
        if r == length(e_y)
            e_y_norm_one_meas = S_i \ e_y;
            if r > 3
                eps_y = e_y(1:3)'*e_y_norm_one_meas(1:3);
            else
                eps_y = e_y' * e_y_norm_one_meas;
            end
        end
        NIS_LKF(j, i) = eps_y;

    end
end
toc

figure(1)
hold on;
grid on;
xlabel('$t$ [s]')
ylabel('$\epsilon_x$')
title('NEES Test Results, LKF')
plot(t, sum(NEES_LKF, 2)/n_trials, '.', 'Color', colors(1, :));
plot([t(1), t(end)], [r1_NEES, r1_NEES], '--',...
    'LineWidth', 3, 'Color', colors(2, :));
plot([t(1), t(end)], [r2_NEES, r2_NEES], '--',...
    'LineWidth', 3, 'Color', colors(2, :), 'HandleVisibility',
       'off');
axis([-Inf, Inf, 0, 10*r2_NEES]);
set(gca,'FontSize',14)
saveas(gcf, 'NEES_LKF', 'epsc')
```

```
figure(2)
hold on;
grid on;
xlabel('$t$ [s]')
ylabel('$\epsilon_y$')
title('NIS Test Results, LKF')
plot(t, sum(NIS_LKF, 2)/n_trials, '.', 'Color', colors(1, :));
plot([t(1), t(end)], [r1_NIS, r1_NIS], '--',...
    'LineWidth', 3, 'Color', colors(2, :));
plot([t(1), t(end)], [r2_NIS, r2_NIS], '--',...
    'LineWidth', 3, 'Color', colors(2, :), 'HandleVisibility',
       'off');
axis([-Inf, Inf, 0, 1.5*r2_NIS]);
set(gca,'FontSize',14)
saveas(gcf, 'NIS_LKF_full', 'epsc')

% Sample Run State Estimate Errors
F3 = figure('DefaultAxesPosition', [0.05, 0.05, 0.95, 0.95]);
set(gcf, 'Position',  pos)
x_nl_ylabels = {'$X$ [km]','$\dot X$ [km/s]','$Y$ [km]','$\dot
   Y$ [km/s]'};
for i = 1:n
    subplot(n, 1, i);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(x_nl_ylabels{i})
    plot(t, e_x_LKF(i, :, trial_plot), 'LineWidth', 1.5)
    P = permute(P_LKF(:, :, :, trial_plot), [1, 3, 2]);
    plot(t, 2*sqrt(P(i, :, i)), '--', 'LineWidth', 1.5,...
        'Color', colors(2, :));
    plot(t, -2*sqrt(P(i, :, i)), '--', 'LineWidth', 1.5,...
        'Color', colors(2, :), 'HandleVisibility', 'off');
    set(gca,'FontSize',12)
end
sgtitle('LKF State Estimate Errors')

saveas(gcf, 'se_LKF_full', 'epsc')

axes_bounds = [-Inf, Inf, -Inf, Inf; -Inf, Inf,...
    -Inf, Inf; -Inf, Inf, -0.4, 0.4];

% Sample Run Measurements
```

```
F4 = figure('DefaultAxesPosition', [0.05, 0.05, 0.95, 0.95]);
set(gcf, 'Position',  pos)
y_lin_ylabels = {'$\rho$ [km]','$\dot \rho$ [km/s]','$\phi$ [
   rad]'};
for j = 1:p
    ax = subplot(3, 1, j);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(y_lin_ylabels{j})
    if j == 4
        for i = 1:nS
            plot(t, wrapToPi(reshape(y_TMT(j, i, :, trial_plot)
               , T, [])-...
                reshape(y_LKF(j, i, :, trial_plot), T, [])), '.
                   ',...
                'Color', colors(i, :));
        end
    else
        for i = 1:nS
            plot(t, reshape(y_TMT(j, i, :, trial_plot), T, [])-
               ...
                reshape(y_LKF(j, i, :, trial_plot), T, []), '.'
                   ,...
                'Color', colors(i, :));
        end
    end
    set(gca,'FontSize',12)
    axis(axes_bounds(j, :))
end
sgtitle('Linearized Measurement Innovations')
saveas(gcf, 'mi_LKF_full', 'epsc')

F5 = figure('DefaultAxesPosition', [0.05, 0.05, 0.95, 0.95]);
set(gcf, 'Position',  [100, 100, 1000, 800])
y_lin_ylabels = {'$\rho$ [km]','$\dot \rho$ [km/s]','$\phi$ [
   rad]'};
for j = 1:p
    ax = subplot(3, 1, j);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(y_lin_ylabels{j})
    for i = 1:nS
%         plot(t, reshape(y_TMT(j, i, :, 1), T, []), '.',...
```

```
%             'Color', colors(1, :));
        plot(t, reshape(y_LKF(j, i, :, trial_plot), T, []), '.'
           ,...
            'Color', colors(i, :));
    end
end
sgtitle('Linearized Measurements')

%% Extended Kalman Filter

diags = 1.2e-10;
offdiags = -0.1*1e-10;
Qguess = [diags, offdiags; offdiags, diags];

x_EKF = zeros(n, T, n_trials);
P_EKF = zeros(n, n, T, n_trials);
y_EKF = zeros(p, nS, T, n_trials);
S_EKF = zeros(p*nS, p*nS, T, n_trials);

Fx = @(x) eye(n)+Dt*A(atan2(x(3), x(1)), sqrt(x(1)^2+x(3)^2));
Hx = @(k, x, i) C(k*Dt, x, i);

fprintf('Running EKF over MC Simulations: ')
tic
for i = 1:n_trials
    [x_t, P_t, y_m, S_t] = EKF(...
        n, y_TMT(:, :, :, i), Qguess, 0.9*Rtrue, mu0, P0, Fx,
           Omegak,...
        Hx, Dt, rho, drho, phi, mu);
    x_EKF(:, :, i) = x_t;
    P_EKF(:, :, :, i) = P_t;
    y_EKF(:, :, :, i) = y_m;
    S_EKF(:, :, :, i) = S_t;
end
toc

e_x_EKF = x_TMT - x_EKF;
e_y_EKF = y_TMT - y_EKF;

NEES_EKF = zeros(T, n_trials);
NIS_EKF = NaN * zeros(T, n_trials);

fprintf('Running NEES/NIS over EKF Data: ')
tic
for i = 1:n_trials
```

```
    e_x_dat = e_x_EKF(:, :, i);
    P_dat = P_EKF(:, :, :, i);
    e_y_dat = e_y_EKF(:, :, :, i);
    S_dat = S_EKF(:, :, :, i);
    for j = 1:T

        % Run NEES Calculation
        e_x = e_x_dat(:, j);
        eps_x = e_x' * (P_dat(:, :, j) \ e_x);
        NEES_EKF(j, i) = eps_x;

        % Run NIS Calculation
        S_i = S_dat(:, :, j);
        S_inds = find(~isnan(S_i(1, :)), 1, 'last');
        S_i = S_i(1:S_inds, 1:S_inds);
        e_y = e_y_dat(:, :, j);
        e_y = e_y(:);
        e_y = e_y(~isnan(e_y));
        [r, ~] = size(S_i);
        eps_y = NaN;
        if r == length(e_y)
            e_y_norm_one_meas = S_i \ e_y;
            if r > 3
                eps_y = e_y(1:3)'*e_y_norm_one_meas(1:3);
            else
                eps_y = e_y' * e_y_norm_one_meas;
            end
        end
        NIS_EKF(j, i) = eps_y;

    end
end
toc

%%
figure(5)
hold on;
grid on;
xlabel('$t$ [s]')
ylabel('$\epsilon_x$')
title('NEES Test Results, EKF')
plot(t, sum(NEES_EKF, 2)/n_trials, '.', 'Color', colors(1, :));
plot([t(1), t(end)], [r1_NEES, r1_NEES], '--',...
    'LineWidth', 3, 'Color', colors(2, :));
plot([t(1), t(end)], [r2_NEES, r2_NEES], '--',...
```

```
    'LineWidth', 3, 'Color', colors(2, :), 'HandleVisibility',
       'off');
axis([-Inf, Inf, 0, 1.5*r2_NEES]);
saveas(gcf, 'NEES_EKF', 'epsc')
set(gca,'FontSize',12)

figure(6)
hold on;
grid on;
xlabel('$t$ [s]')
ylabel('$\epsilon_y$')
title('NIS Test Results, EKF')
plot(t, sum(NIS_EKF, 2)/n_trials, '.', 'Color', colors(1, :));
plot([t(1), t(end)], [r1_NIS, r1_NIS], '--',...
    'LineWidth', 3, 'Color', colors(2, :));
plot([t(1), t(end)], [r2_NIS, r2_NIS], '--',...
    'LineWidth', 3, 'Color', colors(2, :), 'HandleVisibility',
       'off');
axis([-Inf, Inf, 0, 1.5*r2_NIS]);
saveas(gcf, 'NIS_EKF', 'epsc')
set(gca,'FontSize',12)


axes_bounds = [-Inf, Inf, -0.15, 0.15
    -Inf, Inf, -1.5e-3, 1.5e-3
    -Inf, Inf, -0.15, 0.15
    -Inf, Inf, -1.5e-3, 1.5e-3];

% Sample Run State Estimate Errors
F7 = figure('DefaultAxesPosition', [0.05, 0.05, 0.95, 0.95]);
set(gcf, 'Position',  pos)
x_nl_ylabels = {'$X$ [km]','$\dot X$ [km/s]','$Y$ [km]','$\dot
   Y$ [km/s]'};
for i = 1:n
    subplot(n, 1, i);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(x_nl_ylabels{i})
    plot(t, e_x_EKF(i, :, trial_plot), 'LineWidth', 1.5)
    P = permute(P_EKF(:, :, :, trial_plot), [1, 3, 2]);
    plot(t, 2*sqrt(P(i, :, i)), '--', 'LineWidth', 1.5,...
        'Color', colors(1, :));
    plot(t, -2*sqrt(P(i, :, i)), '--', 'LineWidth', 1.5,...
        'Color', colors(1, :), 'HandleVisibility', 'off');
```

```
    axis(axes_bounds(i, :))
    set(gca,'FontSize',12)
end
sgtitle('EKF State Estimate Errors')
saveas(gcf, 'se_EKF', 'epsc')

% Sample Run Measurements

axes_bounds = [-Inf, Inf, -0.5, 0.5
    -Inf, Inf, -Inf, Inf
    -Inf, Inf, -Inf, Inf];

F8 = figure('DefaultAxesPosition', [0.05, 0.05, 0.95, 0.95]);
set(gcf, 'Position',  pos)
y_lin_ylabels = {'$\rho$ [km]','$\dot \rho$ [km/s]','$\phi$ [
   rad]'};
for j = 1:p
    ax = subplot(3, 1, j);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(y_lin_ylabels{j})
    if j == 3
        for i = 1:nS
            plot(t, wrapToPi(reshape(y_TMT(j, i, :, trial_plot)
               , T, [])-...
                reshape(y_EKF(j, i, :, trial_plot), T, [])), '.
                   ',...
                'Color', colors(i, :));
        end
    else
        for i = 1:nS
            plot(t, reshape(y_TMT(j, i, :, trial_plot), T, [])-
               ...
                reshape(y_EKF(j, i, :, trial_plot), T, []), '.'
                   ,...
                'Color', colors(i, :));
        end
    end
    axis(axes_bounds(j, :))
    set(gca,'FontSize',12)
end
sgtitle('EKF Measurement Innovations')
saveas(gcf, 'mi_EKF', 'epsc')
```

```
%% Test On 'Actual' Data

T = length(ydata);
t = (0:T-1)'*Dt;

% format ydata

y_test = NaN*zeros(p, nS, T);
for i = 1:T
    Yi = ydata{i};
    if ~isempty(Yi)
        if ~isnan(Yi(4))
            y_test(:, Yi(4), i) = Yi(1:3);
        end
    end
end

% LKF
Qmag = permute(logspace(-5, -5, T), [1, 3, 2]);
f = 1;
Qframe = repmat(f*[1 0; 0 1], [1, 1, T]);
Qguess = Qmag.*Qframe;
% Qguess = 1e-10 * [1 0; 0 1];
P0 = diag([10, 0.1, 10, 0.1]);
[d_x_LKF, P_LKF, d_y_LKF, S_LKF] = LKF(n, y_test-y_nom, Qguess,
    Rtrue,...
        mu0-mu0_nom, P0, Fk, Omegak, Hk);

x_LKF = d_x_LKF + x_nom;
y_LKF = d_y_LKF + y_nom;


% EKF
Qguess = 1e-10 * [1 0; 0 1];
[x_EKF, P_EKF, y_EKF, S_EKF] = EKF(n, y_test, Qguess, Rtrue,...
    mu0, P0, Fx, Omegak, Hx, Dt, rho, drho, phi, mu);


% Sample Run State Estimate Errors
F_s_LKF = figure('DefaultAxesPosition', [0.05, 0.05, 0.95,
   0.95]);
set(gcf, 'Position',  pos)
x_nl_ylabels = {'$X$ [km]','$\dot X$ [km/s]','$Y$ [km]','$\dot
   Y$ [km/s]'};
for i = 1:n
```

```
    subplot(n, 1, i);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(x_nl_ylabels{i})
    plot(t, x_LKF(i, :), 'LineWidth', 1.5)
    set(gca,'FontSize',14)
end
sgtitle('LKF State Estimate')
saveas(gcf, 'F_LKF_s', 'epsc')

F_c_LKF = figure('DefaultAxesPosition', [0.05, 0.05, 0.95,
   0.95]);
set(gcf, 'Position',  pos)
x_nl_ylabels = {'$X$ [km]','$\dot X$ [km/s]','$Y$ [km]','$\dot
   Y$ [km/s]'};
for i = 1:n
    subplot(n, 1, i);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(x_nl_ylabels{i})
    P = permute(P_LKF, [1, 3, 2]);
    plot(t, 2*sqrt(P(i, :, i)), 'LineWidth', 1.5,...
        'Color', colors(1, :));
    set(gca,'FontSize',14)
end
sgtitle('LKF $2\sigma$ Bounds')
saveas(gcf, 'F_LKF_c', 'epsc')



F_s_EKF = figure('DefaultAxesPosition', [0.05, 0.05, 0.95,
   0.95]);
set(gcf, 'Position',  pos)
x_nl_ylabels = {'$X$ [km]','$\dot X$ [km/s]','$Y$ [km]','$\dot
   Y$ [km/s]'};
for i = 1:n
    subplot(n, 1, i);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(x_nl_ylabels{i})
    plot(t, x_EKF(i, :), 'LineWidth', 1.5)
```

```
    set(gca,'FontSize',14)
end
sgtitle('EKF State Estimate')
saveas(gcf, 'F_EKF_s', 'epsc')

axes_bounds = [-Inf, Inf, 0, 1
    -Inf, Inf, 0, 5e-3
    -Inf, Inf, 0, 1
    -Inf, Inf, 0, 5e-3];

F_c_EKF = figure('DefaultAxesPosition', [0.05, 0.05, 0.95,
   0.95]);
set(gcf, 'Position',  pos)
x_nl_ylabels = {'$X$ [km]','$\dot X$ [km/s]','$Y$ [km]','$\dot
   Y$ [km/s]'};
for i = 1:n
    subplot(n, 1, i);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(x_nl_ylabels{i})
    P = permute(P_EKF, [1, 3, 2]);
    plot(t, 2*sqrt(P(i, :, i)), 'LineWidth', 1.5,...
        'Color', colors(1, :));
    axis(axes_bounds(i, :))
    set(gca,'FontSize',14)
end
sgtitle('EKF $2\sigma$ Bounds')
saveas(gcf, 'F_EKF_c', 'epsc')



% Sample Run Measurements
figure(15)
y_lin_ylabels = {'$\rho$ [km]','$\dot \rho$ [km/s]','$\phi$ [
   rad]'};
for j = 1:p
    ax = subplot(3, 1, j);
    hold on;
    grid on;
    xlabel('$t$ [s]')
    ylabel(y_lin_ylabels{j})
    for i = 1:nS
        plot(t, reshape(y_EKF(j, i, :), T, [])-reshape(y_LKF(j,
            i, :), T, []), '.',...
```

```
            'Color', colors(i, :));
%         plot(t, reshape(y_LKF(j, i, :), T, []), 'o',...
%             'Color', colors(i, :));
    end
end
sgtitle('Filtered Measurements')


%% Functions

% Nonlinear Simulation Function
function dS = nl_sim(t, S, mu, u, w, Dt)

    X = S(1);
    Y = S(3);
    dX = S(2);
    dY = S(4);
    r = sqrt(X^2+Y^2);
    i = round(t/Dt) + 1;
    ddX = -mu*X/r^3 + u(1) + w(1, i);
    ddY = -mu*Y/r^3 + u(2) + w(2, i);

    dS = [dX; ddX; dY; ddY];

end

function dS = nl_sim_nonoise(~, S, mu)

    X = S(1);
    Y = S(3);
    dX = S(2);
    dY = S(4);
    r = sqrt(X^2+Y^2);
    ddX = -mu*X/r^3;
    ddY = -mu*Y/r^3;

    dS = [dX; ddX; dY; ddY];

end

% absolute angular difference
function d = absangdiff(a, b)
    d = abs(b - a);
    d = abs(mod(d, 2*pi) - pi);
end
```

```
% Linearized Kalman Filter
function [dx, P, dy, S] = LKF(...
    n, dy_meas, Q, Rtrue, mu0, P0, Fk, Omega, Hk)

    [p, nS, T] = size(dy_meas);

    dx = zeros(n, T);
    dy = NaN*zeros(p, nS, T);
    P = zeros(n, n, T);
    S = NaN*zeros(p*nS, p*nS, T);

    k = (0:T-1)';

    mu_prev = mu0;
    P_prev = P0;

    dx(:, 1) = mu_prev;
    P(:, :, 1) = P_prev;

    dyframe = dy_meas(:, :, 1);
    yind = find(~isnan(dyframe(1, :)));
    n_meas = length(yind);
    if n_meas > 0
        H = zeros(p*n_meas, n);
        for j = 1:n_meas
            H((p*j-(p-1)):(p*j), :) = Hk(k(1), yind(j));
        end
        R = kron(eye(n_meas), Rtrue);
        S(1:p*n_meas, 1:p*n_meas, 1) = H*P0*H'+R;
    end

    [rQ,~,nQ] = size(Q);

    for i = 2:T
        F = Fk(i-2);

        dxi = F*mu_prev;
        Pi = F*P_prev*F';
        if nQ == T
            if rQ == 2
                Pi = Pi + Omega*Q(:, :, i)*Omega';
            else
                Pi = Pi + Q(:, :, i);
            end
```

```
        else
            if rQ == 2
                Pi = Pi + Omega*Q(:, :, 1)*Omega';
            else
                Pi = Pi + Q(:, :, 1)';
            end
        end
%         Pi = F*P_prev*F'+Omega*Q*Omega';

%         dxi = [0; 0; 0; 0];

        dyframe = dy_meas(:, :, i);
        yind = find(~isnan(dyframe(1, :)));
        n_meas = length(yind);
        if n_meas > 0
            H = zeros(p*n_meas, n);
            for j = 1:n_meas
                H((p*j-(p-1)):(p*j), :) = Hk(k(i), yind(j));
            end

            dyi = dyframe(:, yind);
            dyi = dyi(:);

            dym = H*dxi;
            ddy = dyi-dym;
%             ddy = zeros(size(ddy));
%             dym = reshape(dym, [p, n_meas]);
%             ddy = reshape(ddy, [p, n_meas]);
%             dphi = ddy(3, :);
%             inds = find(abs(dphi) < pi);
%             dym = dym(:, inds);
%             ddy = ddy(:, inds);
%             n_meas = length(inds);
%             yind = yind(inds);
%             ddy = ddy(:);
%             dym = dym(:);
%
%             H = zeros(p*n_meas, n);
%             for j = 1:n_meas
%                 H((p*j-(p-1)):(p*j), :) = Hk(k(i), yind(j));
%             end


            R = kron(eye(n_meas), Rtrue);
            Si = H*Pi*H'+R;
```

```
            S(1:p*n_meas, 1:p*n_meas, i) = Si;
            K = Pi*H'*(Si\eye(p*n_meas));
            Pi = (eye(n)-K*H)*Pi;

            dxi = dxi+K*ddy;

            dy(:, yind, i) = reshape(dym, [p, n_meas]);

        end

        mu_prev = dxi;
        P_prev = Pi;
        dx(:, i) = mu_prev;
        P(:, :, i) = P_prev;

    end

end

% Extended Kalman Filter
function [x_t, P_t, y_m, S_t] = EKF(...
    n, y, Q, R_meas, mu0, P0, Fx, Omega, Hx, Dt, rho, drho, phi
       , mu)

    [p, nS, T] = size(y);

    x_t = zeros(n, T);
    P_t = zeros(n, n, T);

    y_m = NaN * zeros(p, nS, T);

    mu_prev = mu0;
    P_prev = P0;

    x_t(:, 1) = mu_prev;
    P_t(:, :, 1) = P_prev;

    k = (0:(T-1))';

    tspan = [0, 1, Dt];
    opts = odeset('RelTol', 1e-12, 'AbsTol', 1e-12);

    S_t = NaN * zeros(p*nS, p*nS, T);

    % Find first S
```

```
yinds = find(~isnan(y(1, :, 1)));
n_meas = length(yinds);
if n_meas > 0
    H = zeros(n_meas*p, n);
    for j = 1:n_meas
        H((p*j-2):(p*j),:) = Hx(k(1), mu_prev, yinds(j));
    end
    R = kron(eye(n_meas), R_meas);
    S_t(1:n_meas*p, 1:n_meas*p, 1) = H*P0*H'+R;
end

for i = 2:T

    [~, x] = ode45(@(t, S) nl_sim_nonoise(t, S, mu),...
        tspan, mu_prev, opts);

    x = x(end, :);

    F = Fx(mu_prev);
    P = F*P_prev*F' + Omega*Q*Omega';

    yframe = y(:, :, i);
    yinds = find(~isnan(yframe(1, :)));
    n_meas = length(yinds);
    mu_prev = x';
    P_prev = P;
    if n_meas > 0
        R = kron(eye(n_meas), R_meas);
        H = zeros(p*n_meas, n);
        for j = 1:n_meas
            H((p*j-2):(p*j), :) = Hx(k(i), x, yinds(j));
        end

        yi = yframe(:, yinds);
        yi = yi(:);

        rho_mi = rho(k(i)*Dt, x, yinds);
        drho_mi = drho(k(i)*Dt, x, yinds);
        phi_mi = phi(k(i)*Dt, x, yinds);
        ymi2 = [rho_mi; drho_mi; phi_mi];
        ymi = ymi2(:);

        e_yi = yi - ymi;

        S = H*P*H'+R;
```

```
            K = P*H'*(S\eye(p*n_meas));
            mu_prev = x' + K*e_yi;
            P_prev = (eye(n)-K*H)*P;

            S_t(1:p*n_meas, 1:p*n_meas, i) = S;

            y_m(:, yinds, i) = ymi2;
        end

        x_t(:, i) = mu_prev;
        P_t(:, :, i) = P_prev;

    end

end
```