ASEN 5070 Project

Johnathan Clouse*
University of Colorado, Boulder, CO, 80309-0429, USA

*Graduate Students, Aerospace Engineering Sciences, 1111 Engineering Drive, Boulder, CO, 80309-0429

Contents

I. The Orbit Determination Problem

The process of orbit determination is one that combines observations and known dynamics to produce the most likely state of a satellite body in some reference frame. By creating and propagating a reference state based on *a priori* information, one can estimate the deviation from the reference state such that the estimated state is the best fit with observations. If the reference state is sufficiently close to the true state, the problem can be linearized to map the states from epoch to epoch. This problem becomes nontrivial in the presence of perturbations due to gravity fields, drag, third bodies, etc. Born et al. is the primary source for the formulations that follow.

The following equations define the variables needed to linearize the orbit determination problem

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{X}(t) - \mathbf{X}^*(t) \\ \mathbf{y}(t) = \mathbf{Y}(t) - \mathbf{Y}^*(t) &= G(\mathbf{X}, t) + \boldsymbol{\epsilon} - G(\mathbf{X}^*, t) \end{aligned} \tag{1}$$

where $\mathbf{X}(t)$ is the true state, $\mathbf{X}^*(t)$ is the propagated reference state, and $\mathbf{x}(t)$ is the deviation between the two. For the observations, $\mathbf{Y}(t)$ represents the measurements taken of the state, $\mathbf{Y}^*(t)$ is the computed measurements from $\mathbf{X}^*(t)$ at the same time, and $\mathbf{y}(t)$ is the deviation between them. G is the computed measurement based on the state, and $\boldsymbol{\epsilon}$ is the random error associated with the measurement. The state vectors have n elements, and the observation matrices are p measurement types by l observations. A measurement sensitivity matrix $\tilde{H}$ that maps the state to a computed measurement at a given time is implemented as

$$\begin{aligned} \mathbf{y}_i &= \tilde{H}_i \mathbf{x}_i + \boldsymbol{\epsilon}_i \quad (i = 1, ..., l) \\ \tilde{H} &= [\tfrac{\partial G}{\partial \mathbf{X}}]^* \end{aligned} \tag{2}$$

A state transition matrix can map the state from one time to another time. It is expressed as

$$\mathbf{x}(t) = \Phi(t, t_k)\mathbf{x}(t_k) \tag{3}$$

Finally, the linearized process is used to create a least-squares solution that is minimum norm. This solution is one that minimizes the cost function

$$J(\mathbf{x}_k) = 1/2\boldsymbol{\epsilon}^T W \boldsymbol{\epsilon} \tag{4}$$

where W is a matrix of weights for each measurement type.

II. Batch Processing Algorithm

The batch processing algorithm takes *a priori* information and processes it simultaneously with the observations to obtain an estimation of the state deviation and covariance. Minimizing J in equation 4 results in

$$(H^T W H)\hat{\mathbf{x}}_0 = H^T W \mathbf{y} \quad H_i = \tilde{H}_i \Phi(t_i, t_0) \tag{5}$$

where $\hat{\mathbf{x}}_0$ is the estimated initial state deviation. If W is the inverse of the measurement covariance matrix, this is the minimum variance solution as well. Incorporating *a priori* information is expressed by

$$(H^T W H)\hat{\mathbf{x}}_0 = H^T W \mathbf{y} + \bar{P}_0 \bar{\mathbf{x}}_0 \tag{6}$$

where $\bar{P}_0$ is the *a priori* covariance of the state, and $\bar{\mathbf{x}}_0$ is the *a priori* state deviation. The quantities $(H^T W H)$ and $(H^T W \mathbf{y})$ (known as the information matrix and normal matrix, respectively) are accumulated and $\hat{\mathbf{x}}_0$ is solved for. The covariance matrix of the estimate, P, is simply $(H^T W H)^{-1}$.

III. Conventional Kalman Filter

The conventional Kalman filter (CKF) is mathematically equivalent to the batch processor, but processes the observations sequentially. The sequential processing has realtime applications. The algorithm is

as follows: first, the estimated state deviation and covariance matrix from the previous observations are propagated to the time of the current observation to be used as *a priori* information:

$$\begin{aligned} \bar{\mathbf{x}}_i &= \Phi(t_i, t_{i-1})\hat{\mathbf{x}}_{i-1} \\ \bar{P}_i &= \Phi(t_i, t_{i-1})P_{i-1}\Phi(t_i, t_{i-1})^T \end{aligned} \tag{7}$$

This is known as the *time update* step. Next, the Kalman gain is introduced:

$$K = \bar{P}_i\tilde{H}_i^T(\tilde{H}_i\bar{P}_i\tilde{H}_i^T + R_i)^{-1} \tag{8}$$

where R_i is the measurement covariance matrix at the time of observation. K is then used to generate P and $\bar{\mathbf{x}}_i$ in the *measurement update* step

$$\begin{aligned} P_i &= (I - K_i\tilde{H}_i)\bar{P}_i \\ \hat{\mathbf{x}}_i &= \bar{\mathbf{x}}_i + K_i(\mathbf{y}_i - \tilde{H}_i\bar{\mathbf{x}}_i) \end{aligned} \tag{9}$$

The CKF algorithm can be repeated any time new data is acquired. The state transition matrix can be used to map the ith estimated deviation to the initial one.

A major difference between batch processing and the CKF is that P is computed for each observation. While the two algorithms are mathematically equivalent, in practice the solutions diverge due to finite machine precision. In the event of accurate observations with a large *a priori* covariance, P can become asymmetric and non-positive-definite. The Joseph formulation of P rectifies this:

$$P_i = (I - K_i\tilde{H}_i)\bar{P}_i(I - K_i\tilde{H}_i)^T + K_iR_iK_i^T \tag{10}$$

The condition number of a matrix is the ratio of its maximum eigenvalue to its minimum eigenvalue.[1] Large condition numbers result in numerical errors when manipulated with finite precision (i.e., computers!). If P is poorly conditioned, machine precision error can be magnified as the filter continues, contributing to saturation. Square-root formulations can be used to update the covariance matrix in this case.

IV. Simulation Setup and Results: Batch vs. CKF

Observations of a simulated Earth-orbiting spacecraft were obtained. The measurements were taken from three ground sites, including one that is considered fixed (by having a very small *a priori* position variance) on the equator in the Pacific. J_2 and drag effects were considered in the reference trajectory propagation. The state being solved for includes the spacecraft ECI position and velocity, μ, J_2, C_D, and the ECF positions of the ground sites. The *a priori* conditions for the project can be found at the ASEN 5070 Project webpage. The data was processed by the batch and CKF algorithms; the CKF calculated P conventionally and with the Joseph formulation. The prefit residuals, seen in Figure 1, show that the *a priori* state produces a trajectory that can disagree with the measurements by more than 1 km and 10 m/s.

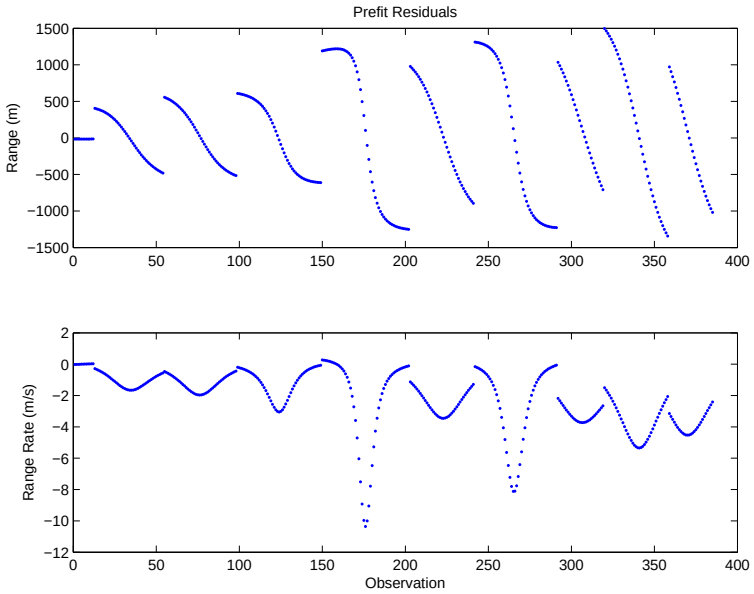


Figure 1. Prefit residuals for range and range rate.

The estimated initial state (Table 1) for both algorithms largely agrees with the *a priori* state, save for the drag coefficient (Z_{site1}'s percent difference is an artifact due values close to 0). Table 2 shows that the estimated final states for both aglorithms, as well as the percent difference between the both CKF P formulations and the batch algorithm.

Table 1. Initial state estimates.

Element	Batch	CKF	%-Diff	Units
x	757.7003e+003	757.7063e+003	789.9839e-006	m
y	5.2226e+006	5.2226e+006	68.7216e-006	m
z	4.8515e+006	4.8515e+006	34.3716e-006	m
$\dot{x}$	2.2133e+003	2.2132e+003	1.0155e-003	m/s
$\dot{y}$	4.6784e+003	4.6783e+003	947.1405e-006	m/s
$\dot{z}$	-5.3713e+003	-5.3713e+003	164.5647e-006	m/s
μ	398.6003e+012	398.6004e+012	34.5500e-006	m^3/s^2
J_2	1.0801e-003	1.0826e-003	235.1557e-003	–
C_D	2.7080e+000	2.0000e+000	26.1444e+000	–
X_{site1}	-5.1275e+006	-5.1275e+006	90.8163e-015	m
Y_{site1}	-3.7942e+006	-3.7942e+006	110.4580e-015	m
Z_{site1}	1.4680e-009	0.0000e+000	100.0000e+000	m
X_{site2}	3.8609e+006	3.8609e+006	888.7788e-006	m
Y_{site2}	3.2385e+006	3.2385e+006	1.1111e-003	m
Z_{site2}	3.8981e+006	3.8981e+006	559.1565e-006	m
X_{site3}	549.4842e+003	549.5050e+003	3.7872e-003	m
Y_{site3}	-1.3809e+006	-1.3809e+006	609.0338e-006	m
Z_{site3}	6.1822e+006	6.1822e+006	168.8545e-006	m

Table 2. Final state estimates.

Element	Batch	CKF	CKF-Batch %-Diff	CKF-Joseph	Joseph-Batch %-Diff	Units
x	1.1286e+006	1.1286e+006	24.7049e-006	1.1286e+006	35.7141e-006	m
y	5.9901e+006	5.9901e+006	1.2996e-006	5.9901e+006	1.5631e-006	m
z	3.7754e+006	3.7754e+006	17.5277e-006	3.7754e+006	15.7053e-006	m
$\dot{x}$	2.0092e+003	2.0092e+003	41.6678e-006	2.0092e+003	28.9850e-006	m/s
$\dot{y}$	3.5630e+003	3.5630e+003	11.4093e-006	3.5630e+003	11.0051e-006	m/s
$\dot{z}$	-6.2376e+003	-6.2376e+003	5.8644e-006	-6.2376e+003	7.7204e-006	m/s
μ	398.6003e+012	398.6004e+012	21.5797e-006	398.6004e+012	22.1143e-006	m^3/s^2
J_2	1.0807e-003	1.0820e-003	116.2359e-003	1.0820e-003	116.5768e-003	–
C_D	2.5212e+000	2.1555e+000	14.5059e+000	2.1446e+000	14.9369e+000	–
X_{site1}	-5.1275e+006	-5.1275e+006	35.7634e-012	-5.1275e+006	33.7655e-012	m
Y_{site1}	-3.7942e+006	-3.7942e+006	35.7638e-012	-3.7942e+006	33.7756e-012	m
Z_{site1}	676.0602e-012	-249.3974e-009	36.9898e+003	-276.4921e-009	40.9976e+003	m
X_{site2}	3.8609e+006	3.8609e+006	518.7581e-006	3.8609e+006	519.4818e-006	m
Y_{site2}	3.2385e+006	3.2385e+006	618.9084e-006	3.2385e+006	623.4447e-006	m
Z_{site2}	3.8981e+006	3.8981e+006	306.5495e-006	3.8981e+006	305.3395e-006	m
X_{site3}	549.4892e+003	549.4992e+003	1.8291e-003	549.4993e+003	1.8465e-003	m
Y_{site3}	-1.3809e+006	-1.3809e+006	296.6497e-006	-1.3809e+006	313.1818e-006	m
Z_{site3}	6.1822e+006	6.1822e+006	96.6940e-006	6.1822e+006	96.9202e-006	m

The postfit residuals can be seen in Figures 2 and 3:

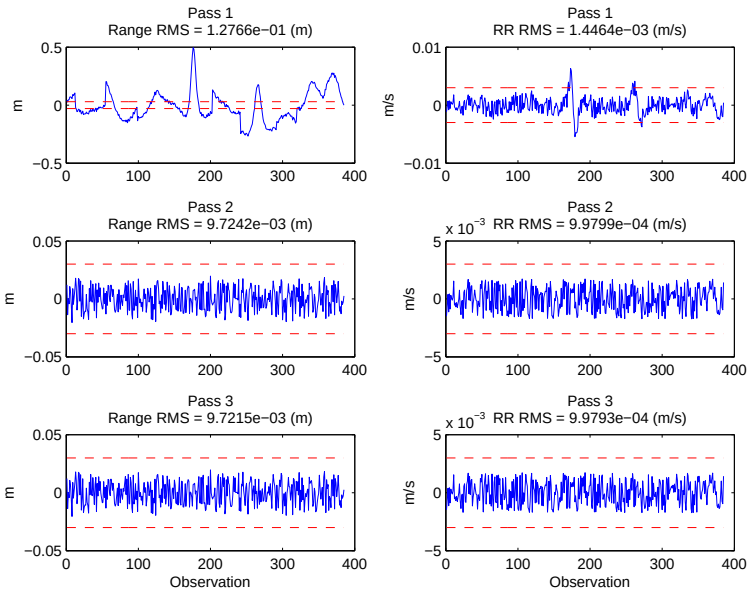
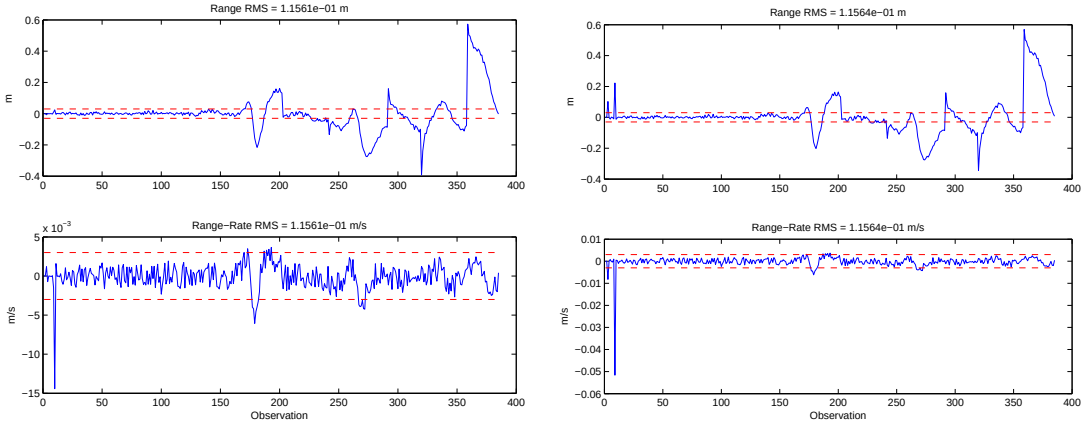


Figure 2. Postfit residuals for range, range rate using batch algorithm. Shown with 3σ measurement envelopes.



(a) Conventional covariance formulation (b) Joseph covariance formulation

Figure 3. Postfit residuals for range, range rate using batch algorithm. Shown with 3σ measurement envelopes.

The traces of the covariance matrix is shown in Figure 4 for both the conventional and Joseph formulations of P:

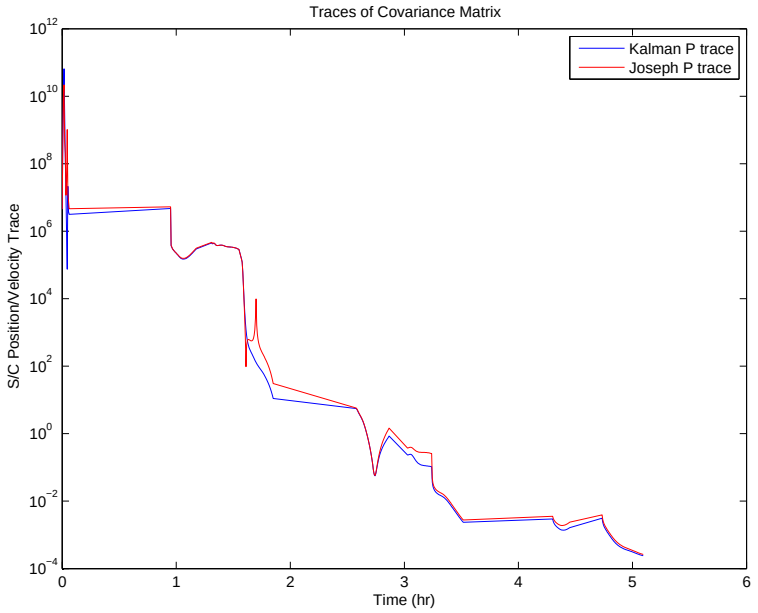
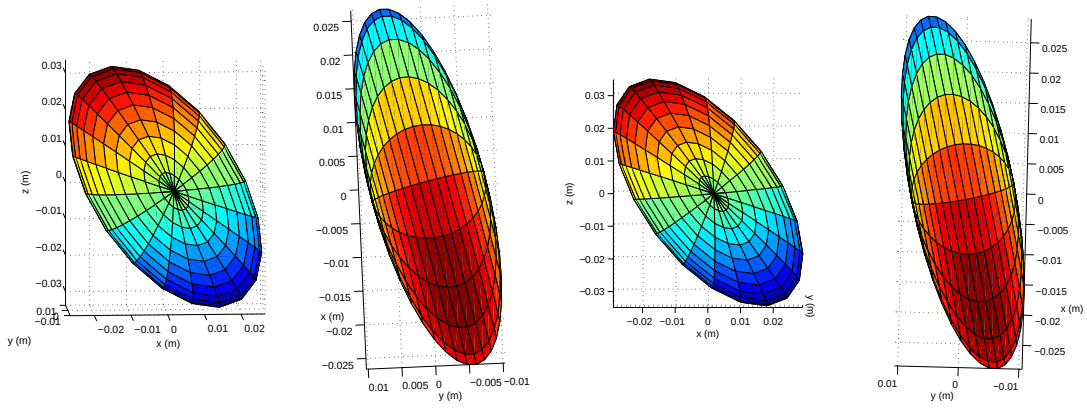


Figure 4. Covariance trace for position and velocity components, CKF.

Finally, the error ellipsoids for both algorithms can be seen in Figure 5.



(a) Batch Algorithm (b) CKF Algorithm

Figure 5. 3σ error ellipsoids.

V. Discussion: Batch vs. CKF

The postfit residuals indicate whether or not an algorithm was within the measurement 3σ error bars. The batch algorithm performed well in this regard; all residuals were within the error envelope within the second pass. Neither implementation of the CKF stayed within the 3σ envelopes in one pass, although both had better range RMS values than the first pass of the batch algorithm. Batch had a lower RMS for range rate in the first pass compared to both CKF formulations, mostly due to an outlier residual on the 9th observation. At this point, the CKF was still susceptible to noise in the measurements because it has had few prior measurements process beforehand. The Joseph formulation of the CKF encountered similar issues in the range residuals, although they were quite small compared to the divergence of the residuals after $\sim$160 observations. The Joseph formulation kept the covariance matrix symmetric, which explains the difference. Symmetry can be lost in the conventional formulation of the covariance when a large *a priori* covariance is processed with accurate measurements due to finite machine precision.[1]

The impact of the Joseph formulation of the CKF covariance can be seen in Figure 4. The Joseph formulation stayed slightly higher throughout, as P kept its symmetry. The variance of the CKF collapses as measurements were processed, making the filter consider measurements less as time progresses. The batch algorithm did not have this issue because it processed *a priori* information like it would a measurement. Adding process noise, which would keep the covariance from collapsing, would help the CKF keep considering new measurements as time progresses.

Additionally, the batch algorithm was less susceptible to errors induced by covariance matrices, because a poorly conditioned information matrix (which required one inversion of the *a priori* covariance matrix) was only inverted once to get the covariance matrix. For this implementation of the batch algorithm, Cholesky decomposition was performed to reduce the error resulting in performing an inverse of a poorly conditioned matrix. The CKF, on the other hand, multiplied the *a priori* three times per observation epoch, as well as inverted one of the products. Error in the covariance was thus introduced and propagated at every observation epoch, even with the Joseph formulation (which became non-positive definite at times). Thus, the batch algorithm had a more accurate solution for P due to computer precision. A square-root method for the CKF would reduce the condition of the matrix containing the uncertainty information by a factor of 2. Such a method would result in a more accurate covariance matrix that would stay symmetric and positive definite.

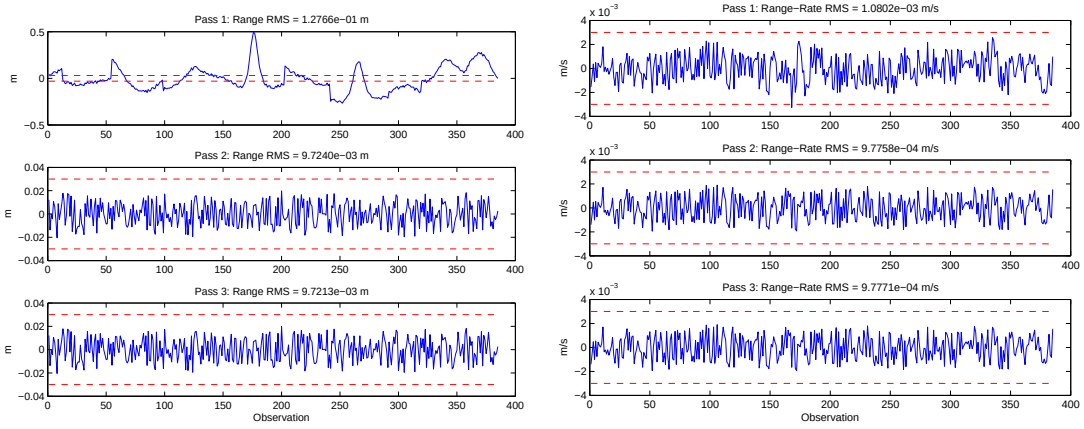
The final error ellipsoids for the spacecraft position between the two algorithms were close in size and direction. The CKF had more uncertainty in the x and z directions. As discussed previously, the batch result was the more reliable of the two.

VI. Processing of Only One Measurement Type

The effects of measurement type (range or range rate) were analyzed. The batch algorithm was run with the same conditions and *a priori* knowledge, but processed once with only range measurements, and once with only range rate measurements. The relative differences of the estimated initial and final states with the dual-measurement batch solution are shown in Table 3. Postfit residuals for both runs are shown in Figure 6, and the 3σ position error are shown in Figure 7.

Table 3. Single-measurement type percent differences with both measurement types.

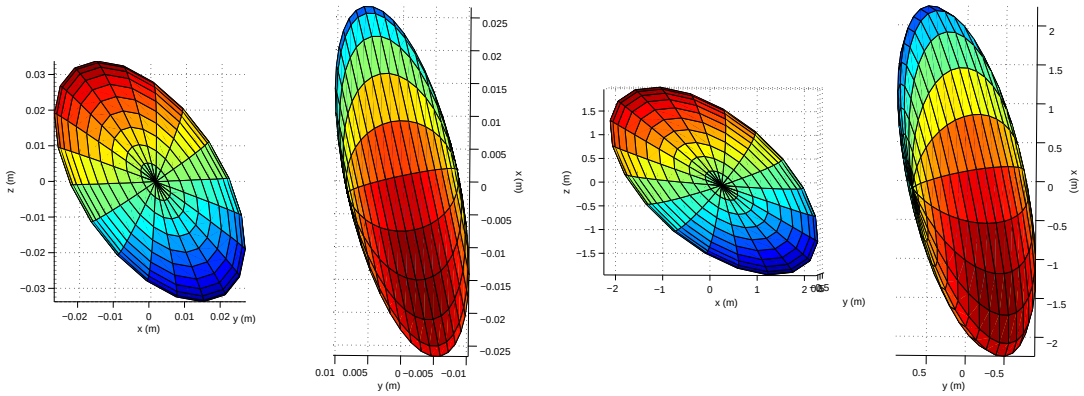
Element	Init. State Range Only %-Diff	Init. State Range-Rate Only %-Diff	Final State Range Only %-Diff	Final State Range-Rate Only %-Diff
x	13.1303e-009	9.6290e-006	14.8102e-006	14.8102e-006
y	882.1571e-012	13.6724e-006	9.3189e-006	9.3189e-006
z	7.5071e-009	4.3868e-006	1.0470e-006	1.0470e-006
$\dot{x}$	442.1213e-012	3.6962e-006	10.4436e-006	10.4436e-006
$\dot{y}$	877.2484e-012	3.7360e-006	2.6213e-006	2.6213e-006
$\dot{z}$	5.4108e-009	10.8486e-006	6.2054e-006	6.2054e-006
μ	1.7415e-009	152.0529e-006	135.7060e-006	135.7060e-006
J_2	1.0009e-006	11.8443e-003	11.0662e-003	11.0662e-003
C_D	19.2734e-003	15.4194e+000	15.7782e+000	15.7782e+000
X_{site1}	18.1633e-015	90.8163e-015	90.8163e-015	90.8163e-015
Y_{site1}	24.5462e-015	110.4580e-015	98.1848e-015	98.1848e-015
Z_{site1}	9.4487e+000	109.5868e+000	121.9467e+000	121.9467e+000
X_{site2}	42.9717e-009	35.1569e-006	28.0806e-006	28.0806e-006
Y_{site2}	24.9975e-009	50.5199e-006	38.5169e-006	38.5169e-006
Z_{site2}	7.2785e-009	31.4754e-006	37.1453e-006	37.1453e-006
X_{site3}	78.5123e-009	26.2115e-006	67.0482e-006	67.0482e-006
Y_{site3}	48.9074e-009	187.7865e-006	155.8026e-006	155.8026e-006
Z_{site3}	12.6907e-009	34.7815e-006	28.4767e-006	28.4767e-006



(a) Range only

(b) Range rate only

Figure 6. Postfit residuals from processing one measurement type, batch algorithm.



(a) Range only (b) Range rate only

Figure 7. 3σ error ellipsoids using only one measurement type, batch algorithm.

The drag coefficient showed the most error with the previous solution (Z_{site1}'s error was on the scale of nanometers). This was expected since the previous solution was so far off from *a priori* knowledge anyway, as neither measurement type was very sensitive to C_D in the first place (compared to other state elements).

The postfit residuals were within the error bars of the measurements in the final pass of the algorithm. The range RMS values were very close to that of the batch process with both measurement types. The range-rate RMS, however, was slightly smaller. This was due to the smaller variance of the range-rate measurement, which made the postfit resuduals less tightly bound around zero in the presence of larger variance data.

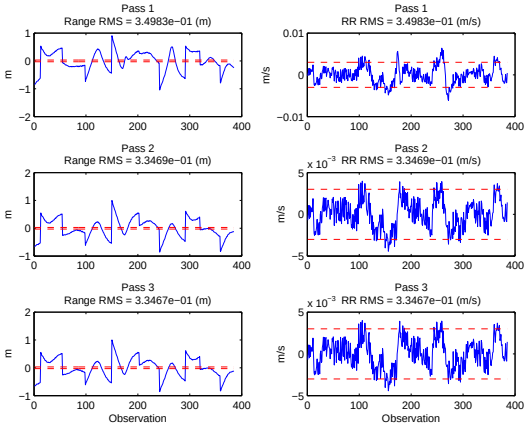
The positions error ellipsoids demonstrated the effect that measurement sensitivity can have, despite the accuracy of the measurements. The range-only ellipsoid was quite similar to the dual-measurement case. However, the range-rate-only ellipsoid was orders of magnitude larger, despite the smaller variance in the measurement. Recall Equation 5. The measurement sensitivity matrix was the culprit here, with the accumulated values for each observation epoch being an order of magnitude smaller for range rate than for range. So despite the accurate data, the range-rate-only solution had more variance simply because the measurements were less sensitive to changes in the state elements. Table 4 demonstrates the overall measurement sensitivity to the state elements by summing all of the elements of H for all measurement epochs.

Table 4. Measurement senisitivity sums.

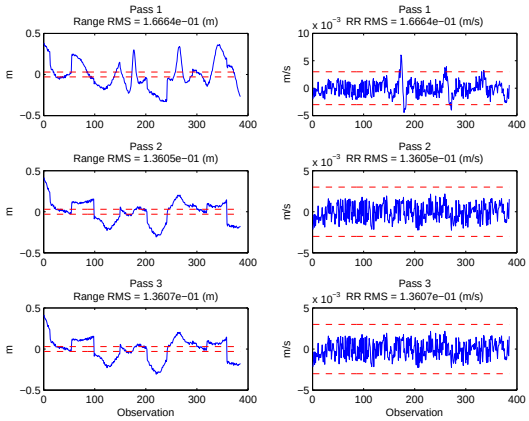
ΣH_{range}	-264.0064e+006
$\Sigma H_{rangerate}$	-45.5771e+006

VII. Ground Station Knowledge

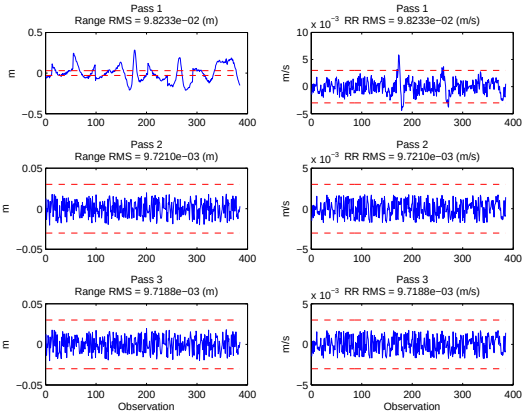
The effects of ground station position confidence was tested by fixing each of the ground stations individually by reducing their position variances to 10^{-10}. The case of all ground stations having a large position variance was also examined. The estimated state relative differences are shown in Table 5 in the Appendix; the postfit residuals and error ellipsoids are displayed in Figures 8 and 9.



(a) Turkey site fixed

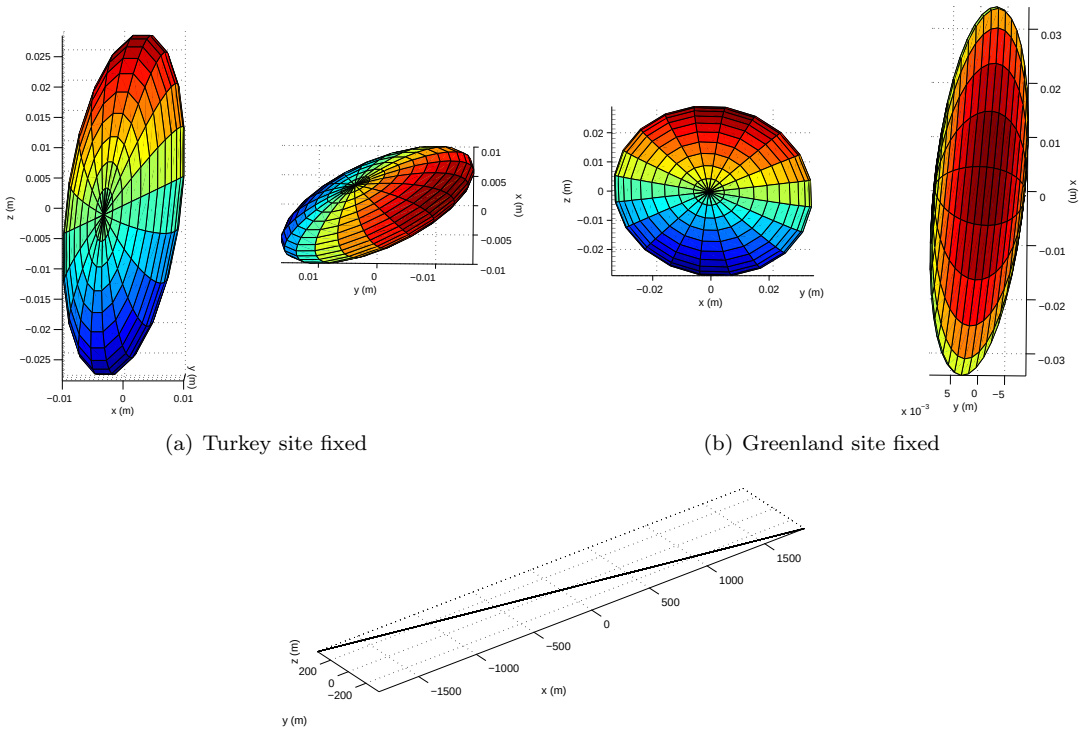


(b) Greenland site fixed



(c) No sites fixed

Figure 8. Postfit residuals from fixing various ground sites, batch algorithm.



(a) Turkey site fixed

(b) Greenland site fixed

(c) No sites fixed

Figure 9. 3σ error ellipsoids from fixing various ground sites, batch algorithm.

For the Turkey and Greenland sites being fixed, the postfit residuals for range weren't within the measurement error envelope. The position error ellipsoids were similar in magnitude to that of the Pacific-fixed site. In contrast, fixing none of the sites resulted in residuals within measurement error, but a position error ellipsoid with a semi-major axis of almost 4 km! An analysis of the ground track during the observations offered insight to these findings. Figure 10 shows where each ground site observed the spacecraft.

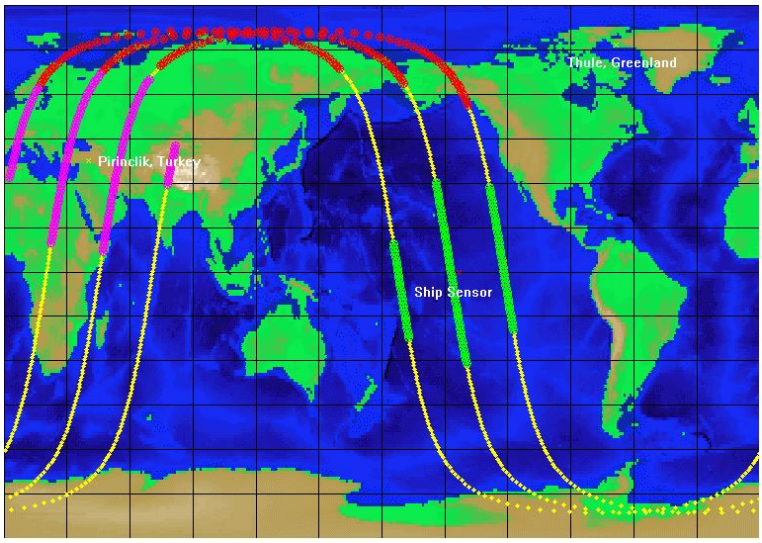


Figure 10. Ground track of spacecraft during observations. Courtesy CCAR.

The Pacific site was near-directly under one of the one of the passes, and it observed the satellite at its maximum rate of latitude change. This gave the site the best geometry for range measurements on that

pass. Additional study into the effect of measurement site and satellite geometry that included azimuth and elevation measurements would help test this hypothesis.

VIII. Potter Square Root Formulation for the CKF

The Potter formulation of the CKF is one of a few square-root methods used to avoid saturation due to numerical error in the measurement update of P. Such methods define $P = WW^T$ so that it can work on the square root of the covariance matrix, W. The result is that W has half the log of the conditioning number of P, and the generation of P from W leads to a symmetric, positive definite matrix. From the measurement update in equation 9, the update can become

$$WW^T = \bar{W}[I - \tilde{F}\alpha\tilde{F}]W^T = \bar{W}\tilde{A}\tilde{A}^T W^T \tag{11}$$

where

$$\tilde{F} = \bar{W}^T H^T, \alpha = (\tilde{F}^T\tilde{F} + R)^{-1} \tag{12}$$

Potter found that by processing measurements independently during an observation epoch, $\tilde{A}$ could be found to be

$$\tilde{A} = [I - \gamma\alpha\tilde{F}\tilde{F}] \tag{13}$$

due to α being scalar. Using the quadratic formula, γ is found to be

$$\gamma = (1 + \sqrt{R\alpha})^{-1} \tag{14}$$

The Potter formulation of the CKF was implemented and processed the same data as the CKF. The final state estimate's relative difference to the batch solution is found in Table 6. The postfit residuals and covariance trace are shown in Figures 11 and 12, respectively

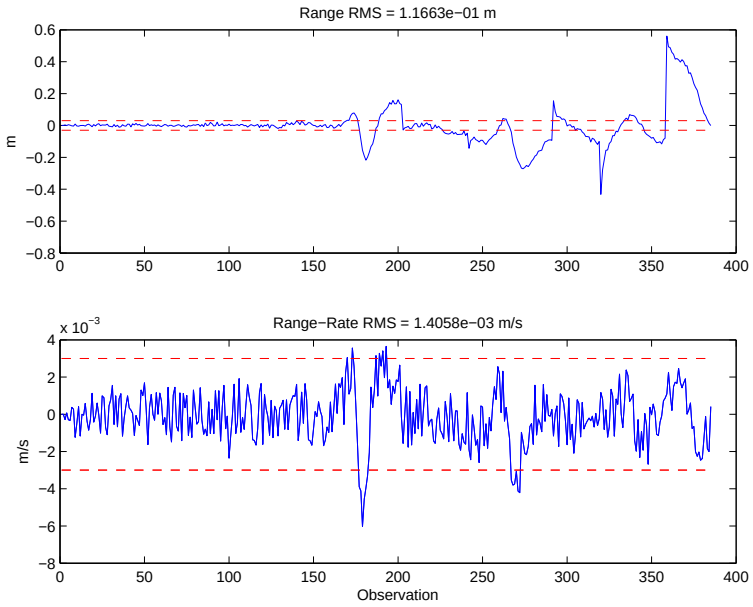


Figure 11. CKF postfit residuals using Potter formulation

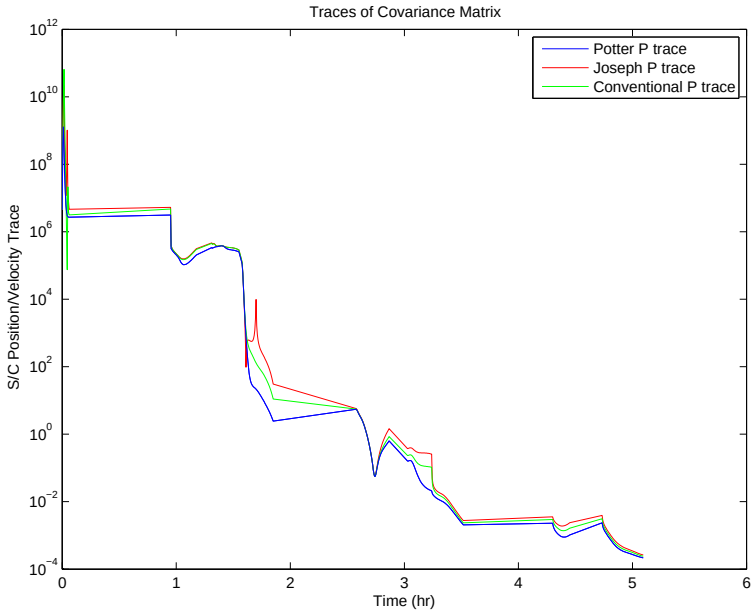


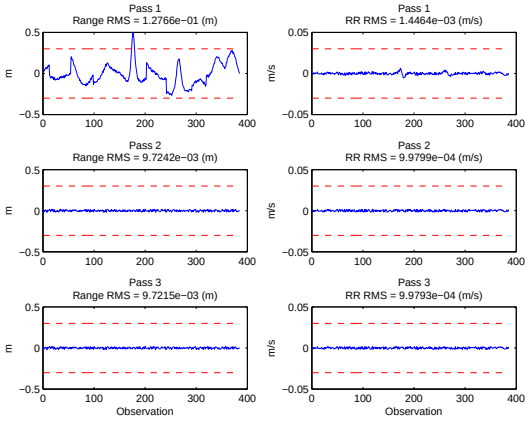
Figure 12. Potter-formulated covariance vs. other CKF formulations.

The postfit residuals stayed within the measurement accuracy envelope up to 160 observations, after which the filter diverges. This result was notable because the conventional and Joseph formulations of P led to large residuals in the first several observations. The estimated states and residuals weren't necessarily better than the previous CKF solutions, but the covariance was more accurate since it stayed symmetric and positive definite while suffering from less machine-precision error. This advantage did not stave off filter divergence, however; process noise and/or use of the extended Kalman filter should be used to rectify the issue.

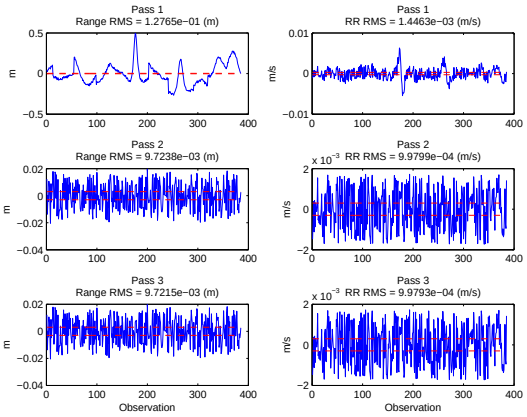
The effect of the calculation and use of W instead of P is seen in Figure 12. Note that the covariance trace wasn't better because it was generally smaller than the others. The Potter trace was a better representation than the others due to how it is calculated. There were observation epochs where the other algorithms diverged from the Potter covariance trace by an order of magnitude or more. Such errors could lead the filter to diverge, if the covariance matrix was so small measurements were ignored.

IX. Varying Data Noise and *A Priori* Covariance Matrices

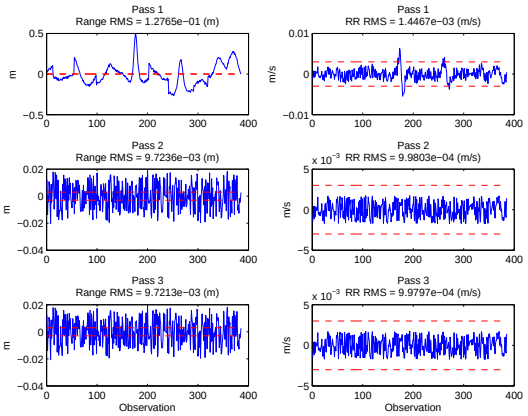
The data noise covariance R was varied by increasing both measurement standard deviations by a factor of 10, decreasing them by a factor of ten, and decreasing only the range measurement standard deviation by a factor of ten. The postfit residuals/RMS and error ellipsoids for the batch solution are shown in Figures 13 and 14.



(a) Increased both noise variance

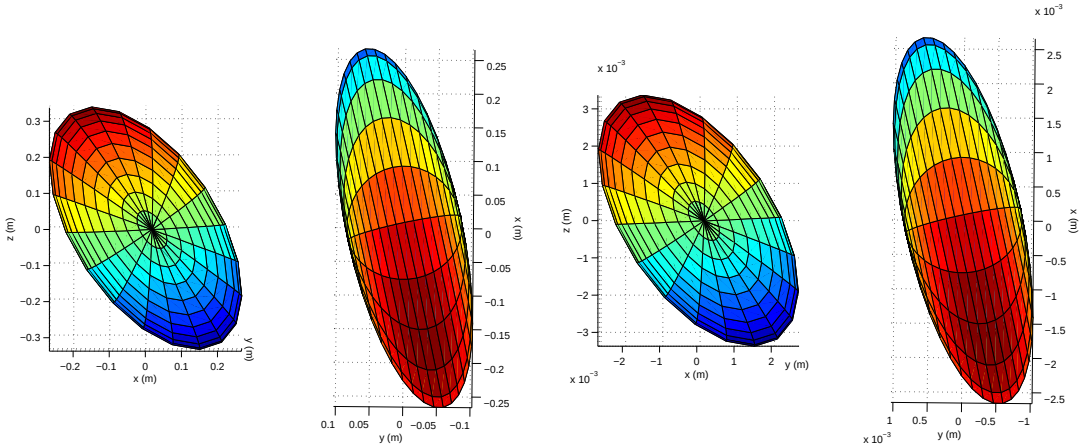


(b) Decreased both noise variance



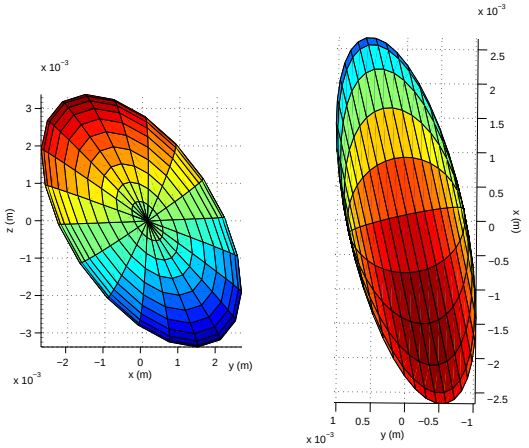
(c) Decreased range noise variance

Figure 13. Postfit residuals from varied data noise, batch algorithm.



(a) Increased both noise variance

(b) Decreased both noise variance



(c) Decreased range noise variance

Figure 14. 3σ error ellipsoids from varied data noise, batch algorithm.

Changing both data noise variances only resulted in the measurement error envelopes changing; the RMS of the residuals stayed the same as before. This was a result of Equation 5 and $W = R^{-1}$. W is a weighting matrix that assigns more or less value on different measurement types at the same epoch. Since the weights were changed by the same factor, the measurements had the same effect on the algorithm and the changes were cancelled out. The range-only change resulted in different final RMS values for the last pass of the batch algorithm. The range-residual RMS decreased, while the range-rate-residual RMS increased. With a higher relative weighting compared to previous runs, the algorithm fit to the range data more than before, at the expense of the range-rate residuals. Note: these noise variances were not changed in the data, which is why the noise in the residuals exceeds the error envelopes. If the data noise actually changed, the residuals and RMS would change as well.

The *a priori* covariance for the satellite position and velocity were also varied: first they were squared from their original value, then set to 10 m^2 and 25 m^2/s^2 respectively. The postfit residuals had the same RMS as the original case, and the relative differences between the cases were vanishingly small. Looking to Equation 6 showed why: *a priori* data are treated like any other measurement. As the number of measurements increase, the batch algorithm is less dependent on the *a priori* information (as it would be for any single observation). Subtracting $\bar{P}_0^{-1}$ from the information matrix and $\bar{P}_0^{-1}\bar{\mathbf{x}}_0$ from the normal matrix had no noticeable effect on the matrices. This confirmed that there would be no appreciable change in the estimated position and velocity of the satellite.

X. Conclusion and Recommendations

The batch processor, using Cholesky decomposition, had the best solution when looking at the residuals and error ellipsoid. Due to propagated errors and a collapsing covariance matrix, the CKF formulations all performed slightly worse. The Potter formulation of the CKF fixed some of the residual outliers the other CKF formulations had. However, all CKF formulations succumbed to saturation. The range measurements provided most of the confidence in the solution (when looking at the covariance), but the range-rate measurements were still useful in finding an estimate with a smaller standard deviation. Fixing of the Pacific site was crucial to the algorithms, as the batch processor only performed well when its position was well-known. Changing the variance of the data noise only had an effect when the weight ratio of the measurement types changed. Changing the *a priori* state covariance had no affect on estimated position and velocity, since it was treated as one measurement among hundreds.

From the results, a few areas of further research are recommended. First would be to determine the effect of azimuth and elevation measurements from the sites. This might improve measurement geometry, and not require the Pacific site to be fixed as it was. Next would be to include process noise in an attempt to thwart filter divergence in the CKF. Finally, EKF results may perform better than the CKF if it were implemented.

There were a few trade-offs identified when considering which algorithm to implement for a given situation. The batch agorithm is built for post-processing, not ideal for autonomous navigation. The CKF can process measurements sequentially, which would be good for autonomous navigation. However, filter saturation can happen if the CKF has too small a covariance. The Joseph and Potter formulations each use more computations, and thus more CPU load. Modern computers can handle such things in realtime, but radiation-resistant/hardened computers for space applications may not be able to.

This project made me aware of several ways that estimation can be performed and improved. I would have liked to see how process noise and the EKF would affect my solutions, but time at the end of this semester was a valuable resource (for everyone, I'm sure). The project format was also good because it allowed a lot of time to explore, compared to homeworks. My recommendation would be to include azimuth/elevation data, perhaps, to see the effect on ground site geometry.

References

[1]Born, G., Schutz, B., and Tapley, B. *Statistical Orbit Determination*, Elsevier Academic Press, 2004.

XI. Appendix A: Supplemental Data

This appendix contains data to further support the conclusions drawn previously.

Table 5. Effects of ground sites being fixed: percent differences with Pacific site fixed.

Element	Init. State None Fixed %-Diff	Init. State Turk. Fixed %-Diff	Init. State Gnld Fixed %-Diff	Final State None Fixed %-Diff	Final State Turk. Fixed %-Diff	Final State Gnld Fixed %-Diff
x	621.2236e-006	2.0129e-003	1.4113e-003	360.0386e-006	1.4995e-003	1.0444e-003
y	13.1088e-006	7.1406e-006	25.0578e-006	12.7798e-006	55.9252e-006	55.3506e-006
z	12.9819e-009	162.5944e-006	45.6413e-006	60.3596e-009	144.1028e-006	68.4470e-006
$\dot{x}$	190.4063e-006	705.3215e-006	623.0584e-006	120.1942e-006	639.2765e-006	632.2569e-006
$\dot{y}$	42.5877e-006	219.2785e-006	197.9999e-006	38.2558e-006	221.0029e-006	111.4226e-006
$\dot{z}$	28.6739e-009	86.5533e-006	32.7989e-006	2.4637e-009	26.4105e-006	35.9176e-006
μ	13.8104e-006	175.0525e-006	6.2032e-006	13.8493e-006	132.0406e-006	6.4772e-006
J_2	2.0694e-003	4.6110e-003	6.9971e-003	2.0742e-003	3.7851e-003	4.8352e-003
C_D	1.5264e+000	22.0637e+000	2.3791e+000	1.6609e+000	19.1942e+000	1.9327e+000
X_{site1}	117.9602e-006	804.1027e-006	796.4735e-006	67.8348e-006	601.9432e-006	607.2670e-006
Y_{site1}	175.8016e-006	1.5927e-003	1.3662e-003	84.1457e-006	1.1994e-003	1.0358e-003
Z_{site1}	126.4290e+009	1.1550e+012	629.9792e+009	275.2865e+009	1.9193e+012	1.0408e+012
X_{site2}	110.5614e-006	1.0513e-003	720.7838e-006	53.6770e-006	792.0839e-006	549.5881e-006
Y_{site2}	148.7567e-006	1.2350e-003	907.0205e-006	67.8854e-006	926.0770e-006	691.1627e-006
Z_{site2}	13.7672e-006	608.6156e-006	12.5391e-006	13.8041e-006	455.2844e-006	8.4452e-006
X_{site3}	353.8359e-006	3.9967e-003	3.7872e-003	183.4620e-006	3.0113e-003	2.8756e-003
Y_{site3}	26.0506e-006	327.3688e-006	609.0337e-006	1.0045e-006	250.4703e-006	462.6257e-006
Z_{site3}	28.7080e-006	383.3949e-006	168.8545e-006	28.7864e-006	290.1459e-006	120.7160e-006

Table 6. Effects of ground sites being fixed: percent differences with Pacific site fixed.

Element	Final State CKF-Potter %-Diff
x	21.6987e-006
y	1.1305e-006
z	18.4758e-006
$\dot{x}$	45.0495e-006
$\dot{y}$	11.8521e-006
$\dot{z}$	5.2066e-006
μ	21.4489e-006
J_2	116.1292e-003
C_D	14.8210e+000
X_{site1}	36.4355e-012
Y_{site1}	36.4389e-012
Z_{site1}	37.6443e+003
X_{site2}	518.4772e-006
Y_{site2}	617.8071e-006
Z_{site2}	306.6402e-006
X_{site3}	1.8233e-003
Y_{site3}	292.8506e-006
Z_{site3}	96.2513e-006

XII. Appendix B: Code

This appendix contains all Matlab code created by the author for the project.

HW 6: Setting Up the Term Project

Table of Contents

Initialize

```
clearvars -except function_list pub_opt CKF_joseph_init_state CKF_joseph_final_sta
global function_list;
function_list = {};
% close all

stat_od_proj_init
ObsData = load('ObsData.txt');
```

Compute information matrix, normal matrix, x_hat

```
consts.Re = Re;
consts.area = drag.A;
consts.rho = compute_density(ri);
consts.theta_dot = theta_dot;
consts.m = drag.m;
consts.state_len = 18;

P0 = eye(consts.state_len)*1e6;
P0(7,7) = 1e20;

% Things to change for further analysis
sig_meas_up = 0;
sig_meas_dn = 0;

fix_ship = 1;
fix_turk = 0;
fix_gnld = 0;

range_only = 0;
rangerate_only = 0;

cov_down = 1;
cov_up = 0;

if fix_ship
    P0(10:12,10:12) = eye(3)*1e-10;
end
```

```
if fix_turk
    P0(13:15,13:15) = eye(3)*1e-10;
end
if fix_gnld
    P0(16:18,16:18) = eye(3)*1e-10;
end

if cov_down
    P0(1:3,1:3) = eye(3)*10; %sigma = 50 m
    P0(4:6,4:6) = eye(3)*25; %sigma = 50 m
%     P0(7,7) = 1e15;
end
if cov_up
    P0(1:3,1:3) = eye(3)*1e12; %sigma = 50 m
    P0(4:6,4:6) = eye(3)*1e12; %sigma = 50 m
%     P0(7,7) = 1e15;
end

x0_ap = zeros(consts.state_len,1);

sig_range = 0.01; % m
sig_rangerate = 0.001; %m/s
if sig_meas_up
    sig_range = sig_range * 10;
    sig_rangerate = sig_rangerate * 10
elseif sig_meas_dn
    sig_range = sig_range / 10;
    sig_rangerate = sig_rangerate / 10;
end
W = [1/(sig_range*sig_range) 0; 0 1/(sig_rangerate*sig_rangerate)];


dt = 0.1;
times = 0:dt:18340;
ode_opts = odeset('RelTol', 1e-12, 'AbsTol', 1e-20);

pfr_plots = figure;
if range_only
    W = 1/(sig_range*sig_range);
elseif rangerate_only
    W = 1/(sig_rangerate*sig_rangerate);
end

[num_obs, ~] = size(ObsData);
y_store = zeros(2,num_obs);
for iter = 1:3
[T,X] = ode45(@two_body_state_dot, times, state, ode_opts, propagator_opts);

% Store off every 20 seconds of data
X_store = X(mod(times,20) == 0,:);
T_store = T(mod(times,20) == 0);

% Accumulate the information and normal matrices
% info_mat = zeros(consts.state_len);
```

```
chol_P0 = chol(P0,'lower');
P0_inv = eye(18)/(chol_P0')/(chol_P0);
info_mat = P0_inv;
norm_mat = P0_inv*x0_ap;
% H_tilda_given = load('BatchHtilda.mat');
cntr =1 ;
% Obs. deviation
y1 = zeros(num_obs,1);
y2 = zeros(num_obs,1);
for ii = 1:num_obs
    site_num = 0;
    for jj = 1:3
        if ObsData(ii, 2) == site(jj).id
            site_num = jj;
            break
        end
    end
    t_obs = ObsData(ii,1);
    ostate = X(T(:,1)==t_obs,1:6);

    r_comp = compute_range_ECFsite(ostate(1:3),...
        site(site_num).r,theta_dot*t_obs);
    rr_comp = compute_range_rate_ECFsite(ostate(1:6),...
        site(site_num).r,theta_dot*t_obs, theta_dot);

    y1(ii) = (ObsData(ii,3)-r_comp);
    y2(ii) = (ObsData(ii,4)-rr_comp);

end

RMS_accum = 0;
num_RMS_meas = 0;
H_store = zeros(2,consts.state_len,num_obs);
pfr_store = zeros(2,num_obs);
if range_only || rangerate_only
    H_store = zeros(1,consts.state_len,num_obs);
    pfr_store = zeros(1,num_obs);
end
obs_time_store = zeros(num_obs,1);


% Begin batch
for ii = 1:num_obs
    obs_time = ObsData(ii,1);
    obs_site = ObsData(ii,2);
    obs_time_store(ii) = obs_time;
    % STM
    STM = eye(consts.state_len);
    STM(1:important_block(1),1:important_block(2)) = ...
        reshape(X_store(T_store == obs_time,consts.state_len+1:end), ...
        important_block(1), important_block(2));

    % H~
    consts.t = obs_time;
```

```
        for xx = 1:3
            if site(xx).id == obs_site
                consts.site = xx;
                break
            end
        end
        state_at_obs = X_store(T_store == obs_time,1:consts.state_len);
        H_tilda = stat_od_proj_H_tilda(state_at_obs, consts);
        if range_only
            H_tilda(2,:) = [];
        elseif rangerate_only
            H_tilda(1,:) = [];
        end

        %H
        H = H_tilda*STM;
        H_store(:,:,ii) = H;

        % Accumulate information matrix
        info_mat = info_mat + H'*W*H;

        % Accumulate normal matrix
        y = [y1(ii); y2(ii)];
        if range_only
            y = y1(ii);
        elseif rangerate_only
            y = y2(ii); %#ok<*UNRCH>
        end
        norm_mat = norm_mat + H'*W*y;

        if iter == 1
            y_store(:,ii) = y;
        end
    end
    x_est = cholesky_linear_solver(info_mat,norm_mat)

    % RMS
    for ii = 1:num_obs
        y = [y1(ii); y2(ii)];
        if range_only
            y = y1(ii);
        elseif rangerate_only
            y = y2(ii); %#ok<*UNRCH>
        end
        H = H_store(:,:,ii);
        postfit_res = y - H*x_est;
        pfr_store(:,ii) = postfit_res;
        RMS_accum = RMS_accum + postfit_res'*W*postfit_res;
    end
    RMS = sqrt(RMS_accum/num_obs)
    if ~range_only
        rangerate_RMS = sqrt(sum(pfr_store(1,:).*pfr_store(1,:))/num_obs)
    end
    if ~rangerate_only
```

```
    range_RMS = sqrt(sum(pfr_store(1,:).*pfr_store(1,:))/num_obs)
end
if ~range_only && ~rangerate_only
    rangerate_RMS = sqrt(sum(pfr_store(2,:).*pfr_store(2,:))/num_obs)
    range_RMS = sqrt(sum(pfr_store(1,:).*pfr_store(1,:))/num_obs)
end

x0_ap = x0_ap-x_est;
state(1:18) = state(1:18) + x_est;

if ~range_only && ~rangerate_only
    figure(pfr_plots);
    subplot(3,2,iter*2-1)
    plot(1:num_obs,pfr_store(1,:))
    hold on
    plot(1:num_obs,3*sig_range*ones(1,num_obs),'r--')
    plot(1:num_obs,-3*sig_range*ones(1,num_obs),'r--')
    ylabel('m')
    title({sprintf('Pass %d',iter),...
        sprintf('Range RMS = %.4e (m)',range_RMS)})
    subplot(3,2,iter*2)
    plot(1:num_obs,pfr_store(2,:))
    hold on
    plot(1:num_obs,3*sig_rangerate*ones(1,num_obs),'r--')
    plot(1:num_obs,-3*sig_rangerate*ones(1,num_obs),'r--')
    ylabel('m/s')
    title({sprintf('Pass %d',iter),...
        sprintf('RR RMS = %.4e (m/s)',rangerate_RMS)})
else

    if rangerate_only
        units = 'm/s';
        my_title = 'Range-Rate';
        this_RMS = rangerate_RMS;
        this_sig = sig_rangerate;
    else
        units = 'm';
        my_title = 'Range';
        this_RMS = range_RMS;
        this_sig = sig_range;
    end
    figure(pfr_plots);
    subplot(3,1,iter)
    plot(1:num_obs,pfr_store(1,:))
    hold on
    plot(1:num_obs,3*this_sig*ones(1,num_obs),'r--')
    plot(1:num_obs,-3*this_sig*ones(1,num_obs),'r--')
    ylabel(units)
    title(sprintf('Pass %d: %s RMS = %.4e %s',...
        iter, my_title, this_RMS, units))
end

end
```

```
if ~range_only && ~rangerate_only
    figure(pfr_plots);
    subplot(3,2,1)
%     title('Range Postfit Residuals')
    subplot(3,2,2)
%     title('Range-Rate Postfit Residuals')
    subplot(3,2,5)
    xlabel('Observation')
    subplot(3,2,6)
    xlabel('Observation')
else
    if rangerate_only
    end
    figure(pfr_plots);
    subplot(3,1,1)
    subplot(3,1,3)
    xlabel('Observation')
end

figure
subplot(2,1,1)
plot(1:num_obs,y_store(1,:),'.')
title('Prefit Residuals')
ylabel('Range (m)')
subplot(2,1,2)
plot(1:num_obs,y_store(2,:),'.')
xlabel('Observation'),ylabel('Range Rate (m/s)')
%
if ~range_only && ~rangerate_only && fix_ship && ~fix_gnld && ~fix_turk
X_est_batch = x_est + state(1:consts.state_len);
X_est_final_batch = STM*x_est + X_store(end,1:consts.state_len)';
elseif range_only
X_est_batch_r = x_est + state(1:consts.state_len);
X_est_final_batch_r = STM*x_est + X_store(end,1:consts.state_len)';
elseif rangerate_only
X_est_batch_rr = x_est + state(1:consts.state_len);
X_est_final_batch_rr = STM*x_est + X_store(end,1:consts.state_len)';
elseif fix_turk
X_est_batch_tfix = x_est + state(1:consts.state_len);
X_est_final_batch_tfix = STM*x_est + X_store(end,1:consts.state_len)';
elseif fix_gnld
X_est_batch_gfix = x_est + state(1:consts.state_len);
X_est_final_batch_gfix = STM*x_est + X_store(end,1:consts.state_len)';
elseif ~fix_ship
X_est_batch_nfix = x_est + state(1:consts.state_len);
X_est_final_batch_nfix = STM*x_est + X_store(end,1:consts.state_len)';
end



chol_info_mat = chol(info_mat,'lower');
P0_est_batch = eye(18)/(chol_info_mat')/(chol_info_mat);
% The last STM in the loop is the 0,t_end one.
P_final_batch = STM*P0_est_batch*STM';
```

```
fprintf('Final Covariance:\n')
disp(P_final_batch(1:6,1:6))

% Ellipsoid
[U,Pprime] = eigs(P_final_batch(1:3,1:3));
Semi = [sqrt(9*Pprime(1,1)) sqrt(9*Pprime(2,2)) sqrt(9*Pprime(3,3))];
figure
subplot(1,2,1)
plotEllipsoid(U,Semi);
% title('3-sigma Position Error Probability Ellipsoid - Batch')
xlabel('x (m)'),ylabel('y (m)'),zlabel('z (m)')
subplot(1,2,2)
plotEllipsoid(U,Semi);
% title('3-sigma Position Error Probability Ellipsoid - Batch')
xlabel('x (m)'),ylabel('y (m)'),zlabel('z (m)')
```

Published with MATLAB® R2013b

HW 10: Sequential Processor for the Term Project

Table of Contents

Initialize

```
clearvars -except function_list pub_opt P_joseph_store x_est_batch P0_est_batch P_
global function_list;
function_list = {};
% close all

stat_od_proj_init
ObsData = load('ObsData.txt');

consts.Re = Re;
consts.area = drag.A;
consts.rho = compute_density(ri);
consts.theta_dot = theta_dot;
consts.m = drag.m;
consts.state_len = 18;

P0 = eye(consts.state_len)*1e6;
P0(7,7) = 1e20;
P0(10:12,10:12) = eye(3)*1e-10;

x0_ap = zeros(consts.state_len,1);

sig_range = 0.01; % m
sig_rangerate = 0.001; %m/s
W = [1/(sig_range*sig_range) 0; 0 1/(sig_rangerate*sig_rangerate)];
R = [(sig_range*sig_range) 0; 0 (sig_rangerate*sig_rangerate)];
```

Sequential Processor

```
dt = 0.1;
times = 0:dt:18340;
ode_opts = odeset('RelTol', 1e-12, 'AbsTol', 1e-20);

% for iter = 1:3
[T,X] = ode45(@two_body_state_dot, times, state, ode_opts, propagator_opts);

% Store off every 20 seconds of data
```

```
X_store = X(mod(times,20) == 0,:);
T_store = T(mod(times,20) == 0);

[num_obs, ~] = size(ObsData);
chol_P0 = chol(P0,'lower');
P0_inv = chol_P0'\inv(chol_P0);
info_mat = P0_inv;
norm_mat = P0_inv*x0_ap;

% Obs. deviation
y1 = zeros(num_obs,1);
y2 = zeros(num_obs,1);
for ii = 1:num_obs
    site_num = 0;
    for jj = 1:3
        if ObsData(ii, 2) == site(jj).id
            site_num = jj;
            break
        end
    end
    t_obs = ObsData(ii,1);
    ostate = X(T(:,1)==t_obs,1:6);

    r_comp = compute_range_ECFsite(ostate(1:3),...
        site(site_num).r,theta_dot*t_obs);
    rr_comp = compute_range_rate_ECFsite(ostate(1:6),...
        site(site_num).r,theta_dot*t_obs, theta_dot);

    y1(ii) = (ObsData(ii,3)-r_comp);
    y2(ii) = (ObsData(ii,4)-rr_comp);

end

% CKF init
x_est = x0_ap;
P = P0;
obs_time_last = ObsData(ii,1);

use_joseph = 1;
if use_joseph
    P_joseph_store = zeros(num_obs,1);
else
    P_trace_store = zeros(num_obs,1);
end
STM_accum = eye(consts.state_len);
pfr_store = zeros(2,num_obs);
% Run CKF
for ii = 1:num_obs
    obs_time = ObsData(ii,1);
    obs_site = ObsData(ii,2);

    % STM from last obs to this one.
    % Not very efficient, since I'm running the integrator again.
    if ii == 1
```

```
    STM_obs2obs = eye(consts.state_len);
else
    times_temp = obs_time_last:dt:obs_time;
    last_state = X_store(T_store == ObsData(ii-1,1),:)';
    STM_obs2obs = eye(consts.state_len);
    % Make the STM reflect an epoch time == the last msmnt time
    last_state(consts.state_len+1:end) = ...
        reshape(STM_obs2obs(1:important_block(1),1:important_block(2)),...
        important_block(1)*important_block(2),1);

    [T_temp,X_temp] = ...
        ode45(@two_body_state_dot, times_temp, last_state, ...
        ode_opts, propagator_opts);
    STM_obs2obs(1:important_block(1),1:important_block(2)) = ...
        reshape(X_temp(end,consts.state_len+1:end), ...
        important_block(1), important_block(2));
end
obs_time_last = obs_time;

% Time update
STM_accum = STM_obs2obs*STM_accum;
x_ap = STM_obs2obs*x_est;
P_ap = STM_obs2obs*P*STM_obs2obs';

% H~
consts.t = obs_time;
for xx = 1:3
    if site(xx).id == obs_site
        consts.site = xx;
        break
    end
end
state_at_obs = X_store(T_store == obs_time,1:consts.state_len);
H_tilda = stat_od_proj_H_tilda(state_at_obs, consts);

% Kalman gain
K = P_ap*H_tilda'/(H_tilda*P_ap*H_tilda'+R);

% Measurement Update
y = [y1(ii);y2(ii)];
x_est = x_ap + K*(y - H_tilda*x_ap);
I = eye(consts.state_len);
if use_joseph
    P = (I-K*H_tilda)*P_ap*(I-K*H_tilda)' + K*R*K';
    P_joseph_store(ii) = trace(P(1:6,1:6));
else
    P = (I-K*H_tilda)*P_ap;
    P_trace_store(ii) = trace(P(1:6,1:6));
end

if ii == 1
    x_est_CKF = x_est;
end
pfr = y-H_tilda*x_est;
```

```
    pfr_store(:,ii) = pfr;

end

RMS_accum = 0;
for ii = 1:num_obs
    RMS_accum = RMS_accum + pfr_store(:,ii)'/R*pfr_store(:,ii);
end
RMS = sqrt(RMS_accum/num_obs)

rangerate_RMS = sqrt(sum(pfr_store(2,:).*pfr_store(2,:))/num_obs)
range_RMS = sqrt(sum(pfr_store(1,:).*pfr_store(1,:))/num_obs)


figure
subplot(2,1,1)
plot(1:num_obs, pfr_store(1,:))
hold on
plot(1:num_obs,3*sig_range*ones(1,num_obs),'r--')
plot(1:num_obs,-3*sig_range*ones(1,num_obs),'r--')
title(sprintf('Range RMS = %.4e m',range_RMS))
ylabel('m')
subplot(2,1,2)
plot(1:num_obs, pfr_store(2,:))
hold on
plot(1:num_obs,3*sig_rangerate*ones(1,num_obs),'r--')
plot(1:num_obs,-3*sig_rangerate*ones(1,num_obs),'r--')
title(sprintf('Range-Rate RMS = %.4e m/s',range_RMS))
ylabel('m/s'),xlabel('Observation')
```

Covariance Matrix Traces

The Joseph formulation follows the same basic shape of the regular Kalman P formulation, but is generally slightly larger. The Joseph formulation will better consider measurements because of this.

```
figure
semilogy(ObsData(:,1)/3600,abs(P_trace_store))
hold on
semilogy(ObsData(:,1)/3600,abs(P_joseph_store),'r')
legend('Kalman P trace', 'Joseph P trace')
title('Traces of Covariance Matrix')
xlabel('Time (hr)'), ylabel('S/C Position/Velocity Trace')

if use_joseph
    CKF_joseph_init_state = x_est_CKF+state(1:consts.state_len);
    CKF_joseph_final_state = x_est+X_store(end,1:consts.state_len)';
else
    CKF_init_state = x_est_CKF+state(1:consts.state_len);
    CKF_final_state = x_est+X_store(end,1:consts.state_len)';
    conv_P_trace_store = P_trace_store;
end

% Ellipsoid
```

```
[U,Pprime] = eigs(P(1:3,1:3));
Semi = [sqrt(9*Pprime(1,1)) sqrt(9*Pprime(2,2)) sqrt(9*Pprime(3,3))];
figure
subplot(1,2,1)
plotEllipsoid(U,Semi);
% title('3-sigma Position Error Probability Ellipsoid - Batch')
xlabel('x (m)'),ylabel('y (m)'),zlabel('z (m)')
subplot(1,2,2)
plotEllipsoid(U,Semi);
% title('3-sigma Position Error Probability Ellipsoid - Batch')
xlabel('x (m)'),ylabel('y (m)'),zlabel('z (m)')
```

Published with MATLAB® R2013b

HW 10: Sequential Processor for the Term Project

Table of Contents

Initialize

```
clearvars -except function_list pub_opt P_joseph_store x_est_batch P0_est_batch P_
global function_list;
function_list = {};
% close all

stat_od_proj_init
ObsData = load('ObsData.txt');

consts.Re = Re;
consts.area = drag.A;
consts.rho = compute_density(ri);
consts.theta_dot = theta_dot;
consts.m = drag.m;
consts.state_len = 18;

P0 = eye(consts.state_len)*1e6;
P0(7,7) = 1e20;
P0(10:12,10:12) = eye(3)*1e-10;
W0 = sqrt(P0);

x0_ap = zeros(consts.state_len,1);

sig_range = 0.01; % m
sig_rangerate = 0.001; %m/s
R = [(sig_range*sig_range) 0; 0 (sig_rangerate*sig_rangerate)];
```

Sequential Processor

```
dt = 0.1;
times = 0:dt:18340;
ode_opts = odeset('RelTol', 1e-12, 'AbsTol', 1e-20);

% for iter = 1:3
[T,X] = ode45(@two_body_state_dot, times, state, ode_opts, propagator_opts);

% Store off every 20 seconds of data
```

```
X_store = X(mod(times,20) == 0,:);
T_store = T(mod(times,20) == 0);

[num_obs, ~] = size(ObsData);

% Obs. deviation
y1 = zeros(num_obs,1);
y2 = zeros(num_obs,1);
for ii = 1:num_obs
    site_num = 0;
    for jj = 1:3
        if ObsData(ii, 2) == site(jj).id
            site_num = jj;
            break
        end
    end
    t_obs = ObsData(ii,1);
    ostate = X(T(:,1)==t_obs,1:6);

    r_comp = compute_range_ECFsite(ostate(1:3),...
        site(site_num).r,theta_dot*t_obs);
    rr_comp = compute_range_rate_ECFsite(ostate(1:6),...
        site(site_num).r,theta_dot*t_obs, theta_dot);

    y1(ii) = (ObsData(ii,3)-r_comp);
    y2(ii) = (ObsData(ii,4)-rr_comp);

end
```

CKF init

```
x_est = x0_ap;
P = P0;
W = W0;
obs_time_last = ObsData(ii,1);

use_joseph = 0;
if use_joseph
    P_joseph_store = zeros(num_obs,1);
else
    P_trace_store = zeros(num_obs,1);
end
STM_accum = eye(consts.state_len);
pfr_store = zeros(2,num_obs);
% Run CKF
for ii = 1:num_obs
    obs_time = ObsData(ii,1);
    obs_site = ObsData(ii,2);

    % STM from last obs to this one.
    % Not very efficient, since I'm running the integrator again.
    if ii == 1
        STM_obs2obs = eye(consts.state_len);
    else
```

```
            times_temp = obs_time_last:dt:obs_time;
            last_state = X_store(T_store == ObsData(ii-1,1),:)';
            STM_obs2obs = eye(consts.state_len);
            % Make the STM reflect an epoch time == the last msmnt time
            last_state(consts.state_len+1:end) = ...
                reshape(STM_obs2obs(1:important_block(1),1:important_block(2)),...
                important_block(1)*important_block(2),1);

            [T_temp,X_temp] = ...
                ode45(@two_body_state_dot, times_temp, last_state, ...
                ode_opts, propagator_opts);
            STM_obs2obs(1:important_block(1),1:important_block(2)) = ...
                reshape(X_temp(end,consts.state_len+1:end), ...
                important_block(1), important_block(2));
        end
        obs_time_last = obs_time;

        % Time update
        STM_accum = STM_obs2obs*STM_accum;
        x_ap = STM_obs2obs*x_est;
        W_ap = STM_obs2obs*W;

        % H~
        consts.t = obs_time;
        for xx = 1:3
            if site(xx).id == obs_site
                consts.site = xx;
                break
            end
        end
        state_at_obs = X_store(T_store == obs_time,1:consts.state_len);
        H_tilda = stat_od_proj_H_tilda(state_at_obs, consts);

        Hx = H_tilda*x_ap;
        % Process measurements individually
        for meas = 1:2

        F = W_ap'*H_tilda(meas,:)';
        alpha = 1/(F'*F+R(meas,meas));
        gamma = 1/(1+sqrt(R(meas,meas)*alpha));

        % Kalman gain
        K = alpha*W_ap*F;

        % Measurement Update
        if meas ==1
            y = y1(ii);
        else
            y = y2(ii);
        end
    %     x_est = x_ap + K*(y - Hx(meas));
        x_est = x_ap + K*(y - H_tilda(meas,:)*x_ap);

        % sqrt of the covariance:
```

```
    W = W_ap - gamma*K*F';

    x_ap = x_est; % for the next round
    W_ap = W;
    end

    P = W*W';
    P_trace_store(ii) = trace(P(1:6,1:6));

    pfr = [y1(ii); y2(ii)]-H_tilda*x_est;
    pfr_store(:,ii) = pfr;

    if ii == 1
        x_est_Potter = x_est;
    end
end

RMS_accum = 0;
for ii = 1:num_obs
    RMS_accum = RMS_accum + pfr_store(:,ii)'/R*pfr_store(:,ii);
end
RMS = sqrt(RMS_accum/num_obs)

rangerate_RMS = sqrt(sum(pfr_store(2,:).*pfr_store(2,:))/num_obs)
range_RMS = sqrt(sum(pfr_store(1,:).*pfr_store(1,:))/num_obs)

figure
subplot(2,1,1)
plot(1:num_obs, pfr_store(1,:))
hold on
plot(1:num_obs,3*sig_range*ones(1,num_obs),'r--')
plot(1:num_obs,-3*sig_range*ones(1,num_obs),'r--')
title(sprintf('Range RMS = %.4e m',range_RMS))
ylabel('m')
subplot(2,1,2)
plot(1:num_obs, pfr_store(2,:))
hold on
plot(1:num_obs,3*sig_rangerate*ones(1,num_obs),'r--')
plot(1:num_obs,-3*sig_rangerate*ones(1,num_obs),'r--')
title(sprintf('Range-Rate RMS = %.4e m/s',rangerate_RMS))
ylabel('m/s'),xlabel('Observation')



%
```

Covariance Matrix Traces

The Joseph formulation follows the same basic shape of the regular Kalman P formulation, but is generally slightly larger. The Joseph formulation will better consider measurements because of this.

```
figure
semilogy(ObsData(:,1)/3600,abs(P_trace_store))
```

```
hold on
semilogy(ObsData(:,1)/3600,abs(P_joseph_store),'r')
semilogy(ObsData(:,1)/3600,abs(conv_P_trace_store),'g')
legend('Potter P trace', 'Joseph P trace', 'Conventional P trace')
title('Traces of Covariance Matrix')
xlabel('Time (hr)'), ylabel('S/C Position/Velocity Trace')


Potter_init_state = x_est_Potter+state(1:consts.state_len);
Potter_final_state = x_est+X_store(end,1:consts.state_len)';
```

Published with MATLAB® R2013b

```
function x = cholesky_linear_solver(Y,N)
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

vlen = length(N);
z = zeros(vlen,1);
x = zeros(vlen,1);
U = chol(Y); % Upper
UT = U';
for ii = 1:vlen
    tmp = N(ii);
    for jj = 1:(ii-1)
        tmp = tmp - UT(ii,jj)*z(jj);
    end
    z(ii) = tmp/UT(ii,ii);
end

for ii = vlen:-1:1
    tmp = z(ii);
    for jj = (ii+1):vlen
        tmp = tmp - U(ii,jj)*x(jj);
    end
    x(ii) = tmp/U(ii,ii);
end
```

Published with MATLAB® R2013b

```
function range = compute_range_ECFsite( inrtl_pos, ecf_site, theta )
%compute_range_ECFsite Summary of this function goes here
%   Detailed explanation goes here
fcnPrintQueue(mfilename('fullpath'))

x = inrtl_pos(1);
y = inrtl_pos(2);
z = inrtl_pos(3);
xs = ecf_site(1);
ys = ecf_site(2);
zs = ecf_site(3);

range = sqrt(...
    (x-(xs*cos(theta)-ys*sin(theta)))*(x-(xs*cos(theta)-ys*sin(theta))) + ...
    (y-(xs*sin(theta)+ys*cos(theta)))*(y-(xs*sin(theta)+ys*cos(theta))) + ...
    (z-zs)*(z-zs));

end
```

Published with MATLAB® R2013b

```
function range_rate = compute_range_rate_ECFsite( inrtl_state, ...
    ecf_site, theta, theta_dot )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
fcnPrintQueue(mfilename('fullpath'))

x = inrtl_state(1);
y = inrtl_state(2);
z = inrtl_state(3);
xdot = inrtl_state(4);
ydot = inrtl_state(5);
zdot = inrtl_state(6);
xs = ecf_site(1);
ys = ecf_site(2);
zs = ecf_site(3);

range_rate = (x*xdot+y*ydot+z*zdot...
    - (xdot*xs+ydot*ys)*cos(theta) + theta_dot*(x*xs + y*ys)*sin(theta) ...
    + (xdot*ys-ydot*xs)*sin(theta) + theta_dot*(x*ys-y*xs)*cos(theta) ...
    - zdot*zs)/compute_range_ECFsite(inrtl_state(1:3), ecf_site, theta);

end
```

Published with MATLAB® R2013b

```
% Plot an ellipsoid given an orthonormal, right handed
% transformation matrix, R and the semi - axis, semi
%
% For the Stat. O.D. project R is made up of the eigenvectors
% of the upper 3x3 portion of the covariance matrix.  semi
% contains sigma_x, sigma_y, sigma_z in a column vector.
%
% Downloaded from ASEN 5070 HW site.

function plotEllipsoid(R,semi)
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

[x,y,z] = sphere(20);

x = x * semi(1);
y = y * semi(2);
z = z * semi(3);

[mm,nn] = size(x);

C = (R * [x(:) y(:) z(:)]')';

x = reshape(C(:,1),mm,nn);
y = reshape(C(:,2),mm,nn);
z = reshape(C(:,3),mm,nn);

surf(x,y,z)
axis equal
```

Published with MATLAB® R2013b

```
function A = stat_od_proj_A(state, consts)
%stat_od_proj_A Calculate A matrix for Stat OD project
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

% Init A, set up local vars
A = zeros(consts.state_len);
x = state(1);
y = state(2);
z = state(3);
xdot = state(4);
ydot = state(5);
zdot = state(6);
mu = state(7);
J2 = state(8);
Cd = state(9);

Re = consts.Re;
area = consts.area;
rho = consts.rho;
theta_dot = consts.theta_dot;
m = consts.m;

H = 88667; %m

% vars to reduce computations
x2 = x*x;
y2 = y*y;
z2 = z*z;
r = sqrt(x2+y2+z2);
sqrt_r = sqrt(r);
v = sqrt(xdot*xdot+ydot*ydot+zdot*zdot);
rel_wind_x = (xdot + theta_dot*y);
rel_wind_y = (ydot - theta_dot*x);
zdot2 = zdot*zdot;
rel_wind_mag = sqrt(rel_wind_x*rel_wind_x + rel_wind_y*rel_wind_y + zdot2);
Re2 = Re*Re;

rho0 = 3.614e-13; %kg/m3
r0 = 700000+6378136.3; %km
H = 88667.0; %km

% Only a few elements are populated
A(1,4) = 1;
A(2,5) = 1;
A(3,6) = 1;

A(4,1) = (3*mu*x^2)/(r*r*r*r*r) - ...
    mu/(r*r*r) + ...
    (3*J2*Re2*mu*((5*z2)/(r*r) - 1))/(2*(r)^(5)) - ...
    (15*J2*Re2*mu*x2*z2)/(r)^(9) - ...
    (15*J2*Re2*mu*x2*((5*z2)/(r*r) - 1))/(2*(r)^(2)) + ...
    (Cd*area*theta_dot*rho*rel_wind_x*rel_wind_y)/(2*m*rel_wind_mag) + ...
```

```
    (Cd*area*x*rho*(xdot + theta_dot*y)*rel_wind_mag)/(2*H*m*r);
A1 = [
A2 = [
A3 = [
A4 = [            (3*mu*x^2)/(x^2 + y^2 + z^2)^(5/2) - mu/(x^2 + y^2 + z^2)^(3/2)
A5 = [ (3*mu*x*y)/(x^2 + y^2 + z^2)^(5/2) + (Cd*area*rho0*theta_dot*exp((r0 - (x^2
A6 = [
A(1,:) = A1;
A(2,:) = A2;
A(3,:) = A3;
A(4,:) = A4;
A(5,:) = A5;
A(6,:) = A6;
A4(4);
```

Published with MATLAB® R2013b

```
function H_tilda = stat_od_proj_H_tilda(state, consts)
%stat_od_proj_H_tilda Calculate H_tilda matrix for Stat OD project
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

% Init H_tilda, set up local vars
x = state(1);
y = state(2);
z = state(3);
xdot = state(4);
ydot = state(5);
zdot = state(6);
% mu = state(7);
% J2 = state(8);
% Cd = state(9);

theta_dot = consts.theta_dot;
theta = consts.t*consts.theta_dot;

%Identify the site the observation was from:
xs = state(9+(consts.site-1)*3+1);
ys = state(9+(consts.site-1)*3+2);
zs = state(9+(consts.site-1)*3+3);

H_tilda = zeros(2,18);
H_tilda(1,:) = [
H_tilda(2,:) = [ (xdot + theta_dot*ys*cos(theta) + theta_dot*xs*sin(theta))/((ys*c

% Zero out the site terms  where there weren't observations.
for ii = 1:3
    if consts.site ~= ii
        H_tilda(:,9+(ii-1)*3+1:9+(ii-1)*3+3) = zeros(2,3);
    end
end
```

Published with MATLAB® R2013b

```
fcnPrintQueue(mfilename('fullpath'))
```

Initial data for stat OD project

```
mu = 3.986004415e14; %m3/s2
J2 = 1.082626925638815e-3;
Re = 6378136.3; %m

theta_dot = 7.2921158553e-5; %rad/s

site(1).name = 'Pacific Ocean Ship Sensor';
site(1).id = 101;
site(1).r = [-5127510.0 -3794160.0 0.0]'; % m

site(2).name = 'Pirinclik, Turkey';
site(2).id = 337;
site(2).r = [3860910.0 3238490.0 3898094.0]'; % m

site(3).name = 'Thule, Greenland';
site(3).id = 394;
site(3).r = [549505.0 -1380872.0 6182197.0]'; % m

ri = [757700.0 5222607.0 4851500.0]';
vi = [2213.21 4678.34 -5371.30]';

drag.Cd = 2.0;
drag.A = 3.0; % m
drag.m = 970; %kg

% kilometerize everything
% mu = mu*1e-9;
% Re = Re*1e-3;
% site1.r = site1.r*1e-3; %km
% site2.r = site2.r*1e-3; %km
% site3.r = site3.r*1e-3; %km
% ri = ri*1e-3;%km
% vi = vi*1e-3;%km/s

state = [ri; vi; mu; J2; drag.Cd; site(1).r; site(2).r; site(3).r];

% Set up propagator options
propagator_opts.mu = mu;

propagator_opts.drag = drag;
propagator_opts.drag.use = 1;
propagator_opts.drag.model_params.rho0 = 3.614e-13; %kg/m3
propagator_opts.drag.model_params.r0 = 700000+6378136.3;
propagator_opts.drag.model_params.H = 88667;
propagator_opts.drag.model_params.theta_dot = theta_dot;

propagator_opts.J2.use = 1;
propagator_opts.J2.params.J2 = J2;
```

```
propagator_opts.J2.params.mu = mu;
propagator_opts.J2.params.Re = Re;

propagator_opts.OD.use = 1;
propagator_opts.OD.state_len = 18;
propagator_opts.OD.A_mat_handle = @stat_od_proj_A;
propagator_opts.OD.A_params.Re = Re;
propagator_opts.OD.A_params.area = drag.A;
propagator_opts.OD.A_params.rho = propagator_opts.drag.model_params.rho0;
propagator_opts.OD.A_params.theta_dot = theta_dot;
propagator_opts.OD.A_params.m = drag.m;
propagator_opts.OD.A_params.H = propagator_opts.drag.model_params.H;
important_block = [9 9]; %rows, cols
propagator_opts.OD.A_params.important_block = important_block;
propagator_opts.OD.A_params.state_len = propagator_opts.OD.state_len;
STM_i = eye(propagator_opts.OD.state_len);
state = [state; reshape(STM_i(1:important_block(1),1:important_block(2)),...
    important_block(1)*important_block(2),1)];
```

Published with MATLAB® R2013b