HW 2

Least Squares, Weighted Least Squares, Sequential Batch Filtering

Justin Self

AERO557: Advanced Orbital Mechanics

February 20, 2024

Problem 1

Editing Data with Linear Least Squares

Students collect marble data and are wondering if they should remove any of the data to get a better fit. There are 10 student collected samples are x_{oi} = [1 2 3 4 5 6 7 8 9 10] and y_{oi} = [1 2 2 3 4 4 5 7 9 9]. Assume the mathematical model is y = α + βx. Find the best estimate of the state. Determine what detecting and eliminating a bad data point would do to the state. What happens to the errors and confidence values if you remove a bad data point?

Solution.

See Appendix A for results and code for Problem 1

Raw Data Processing

The best estimate of the state was determined using a least-squares technique assuming a linear fit. The best estimate of the state was:

$$\hat{x} = \begin{bmatrix} -0.46667 \\ 0.92121 \end{bmatrix}$$

The best fit model is:

$$y = \alpha + \beta x$$

Where terms with confidence intervals are:

$\alpha = -0.46667 \pm 0.68313$

$B = 0.92121 \pm 0.1101$

Outliers Removed

Computing error between the predicted and observed state to find error and applying a criteria that *any error greater than unity* would constitute "bad" data, observations 6 and 9 were discarded. After processing the shortened dataset, we have:

$$\hat{x} = \begin{bmatrix} -0.21324 \\ 0.86765 \end{bmatrix}$$

The new linear model is: $y = -0.21324 + 0.86765x$

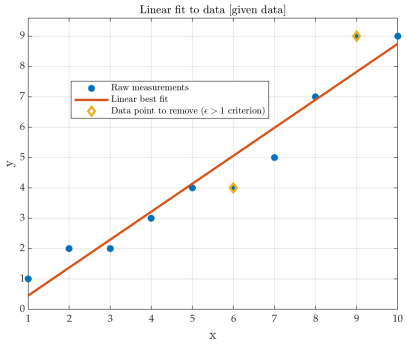
And the new confidence intervals for the coefficients α and β are:

$\alpha = -0.21324 \pm 0.70189$

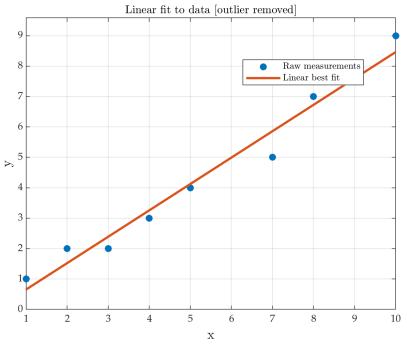
$B = 0.86765 \pm 0.12127$

The plots for this problem are given in Fig. 1 and show the untouched data (1a) and after removing two outliers (1b). The criteria imposed on removing data points was arbitrarily set to errors greater than unity, which was far less restrictive than a realistic criteria. For instance, the criteria was originally set to 2*RMS, but no data exceeded this amount. We tried 1*RMS, but that still yielded no "bad" data. A criteria of 1 was chosen to illustrate data point removal and reprocessing as an exercise.

In this case, removing points 6 and 9 resulted in a *worse* confidence interval, that is, the 95% confidence interval in the first case was ± 0.68 and ± 0.11 for the fit coefficients, respectively, while the intervals widened to ± 0.7 and ± 0.12, respectively. This indicates that removing the two data points did *not* increase confidence in the state estimation. In this case, applying a 1 or 2*RMS error criteria would have been better and, as a result, no data points would have been removed; thus resulting in a higher confidence (tighter intervals) in the estimated state.



(a) Raw data from marble experiment (in blue) with best fit model derived (red) and outliers identified in orange.



(b) Results from processing the data after removing points 6 and 9, identified as outliers by the error criteria defined above.

Figure 1: Least squares results

Problem 2

Weighted Least Squares
Using the data taken from the text file posted for problem 2 (UT date (year, month, day), UT time (hr, min, sec), Az (deg), El (deg), range (km)) taken at a site at latitude (21.5721 deg), longitude (-158.2666 deg) and altitude (300.2 m) with a range error 92.5 m, azimuth error = 0.0224 deg, and elevation error = 0.0139 deg, determine the state vector and co-variance matrix at the first observation as an epoch.

Solution.

See Appendix B for results and code for Problem 2

Most of the details of this problem are best presented in the following charts. The basic batch processing sequence went like this:

- Assume the first data point is the truth model, used for a baseline for all calculations, including Julian Date epoch and observation values.
- Apply weight matrix that accounts for sensor errors, etc.
- Use given radar observations (slant range, azimuth, elevation) to determine best guess for initial state.
- Find the position vector of the observation site at the epoch time (first observation).
- Using Herrick-Gibbs (in this case), find the best estimate of the spacecraft state vector; a 6x1 vector containing position and velocity terms.
- Perform iterative finite differencing to converge on a final state estimate that is more accurate than the initial guess.
- Evaluate guess by computing the covariance matrix, standard deviations and error ellipsoids.

Before Finite Differencing

The initial best guess of the state is defined as:

$$\mathbf{X}_{nom} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$$

After computing the site position, slant range, and spacecraft position vectors at each observation in ECI, the initial state estimation was:

$$\mathbf{X}_{nom} = \begin{bmatrix} 5753.2 \\ 2673.5 \\ 3440.6 \\ 4.3231 \\ -1.927 \\ -5.7256 \end{bmatrix}$$

where the first three elements represent the position in km and the second three elements represent the velocity in km/s.

This initial guess was compared against [1] and the difference was:

```
Difference in position (norm) is: -3.0392 km
Difference in velocity (norm) is: -0.00082403 km/s
```

Since these agree very closely[1], it was determined to be safe to move on to finite differencing.

After Differential Correction with Finite Differencing

After performing the differential correction scheme with finite differencing (and reaching convergence after the fourth iteration) the update to the state became:

$$\mathbf{X}_{nom} = \begin{bmatrix} 5748.9 \\ 2679.8 \\ 3442.9 \\ 4.3287 \\ -1.9204 \\ -5.7262 \end{bmatrix}$$

The final covariance matrix is given in Fig. 2. Final statistical results are shown in Fig. 3. The results are shown graphically in Figs. 4 - 7.

[1] A closer estimate could likely be obtained from performing the more "rigorous" frame conversions as [1] did, but this level of error is acceptable for this problem.

```
Covariance matrix, P, is:
      0.07334    0.0054449   -0.0081869  -0.00086739  -3.5915e-05   4.2618e-05
    0.0054449     0.039581    -0.061185  -1.1602e-05  -0.00044222   0.00065953
   -0.0081869    -0.061185      0.10625   4.8006e-05   0.00069256   -0.0012172
  -0.00086739  -1.1602e-05   4.8006e-05   1.4036e-05  -4.3312e-07  -9.3701e-07
  -3.5915e-05  -0.00044222   0.00069256  -4.3312e-07   6.0007e-06  -9.5418e-06
   4.2618e-05   0.00065953   -0.0012172  -9.3701e-07  -9.5418e-06   1.9981e-05
```

Figure 2: Covariance matrix for batch filtered observation data, $n = 10$.

```
One sigma stdDev for I direction is: 270.8143 m
One sigma stdDev for J direction is: 198.9495 m
One sigma stdDev for K direction is: 325.9605 m
Eigenvalues are:
    0.0032354           0           0
            0    0.071979           0
            0           0     0.14396

Eigenvectors are:
   -0.0072111     0.99044    -0.13773
      0.86001   -0.064134    -0.50623
      0.51023      0.1221     0.85133

Dimensions of the error ellipsoid are:
rI direction: 56.8804 m
rJ direction: 268.2882 m
rK direction: 379.4175 m
```

Figure 3: Statistical results for batch filtering on initial dataset.

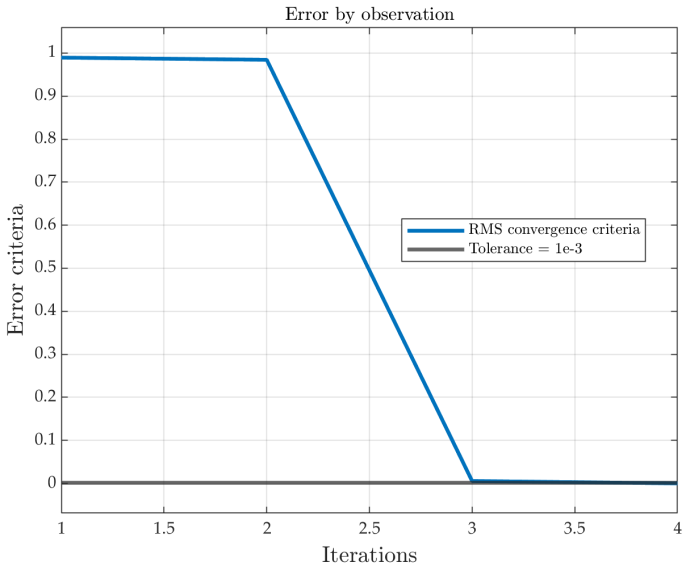


Figure 4: Error criteria vs. iteration showing only four iterations to convergence. See Appendix B for detailed analysis and error criteria.

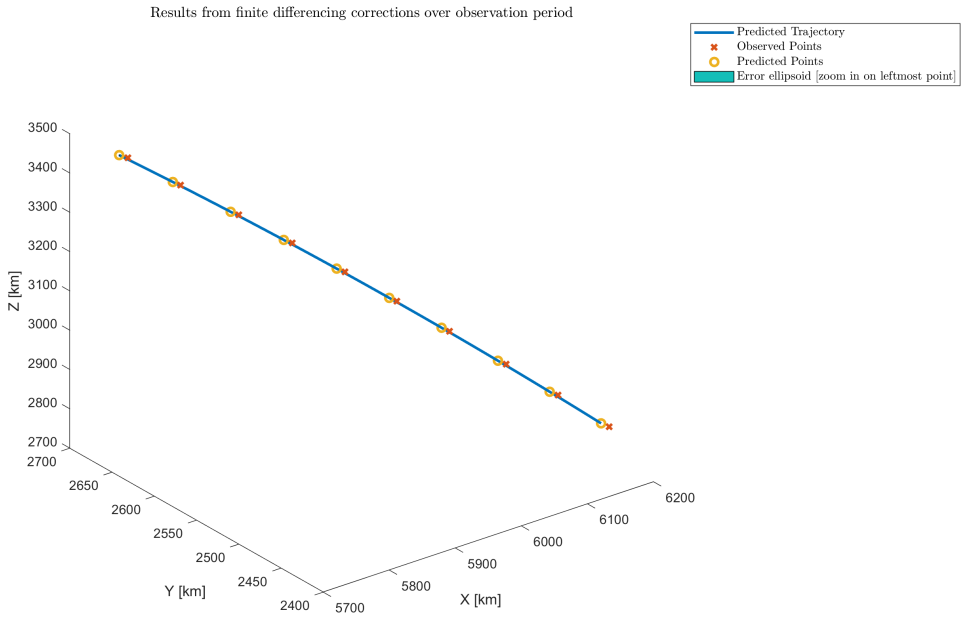


Figure 5: Radar data (x), predicted positions (o), and the predicted trajectory (in blue). The error ellipsoid is visible in Fig. 6.

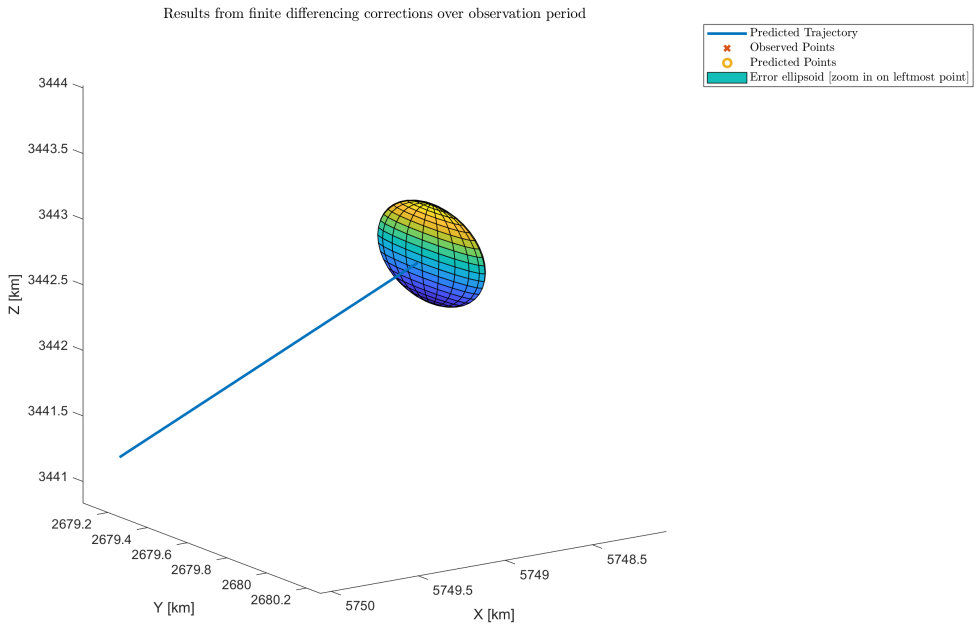


Figure 6: Error ellipsoid surrounding best estimate of the state variable after finite differencing.

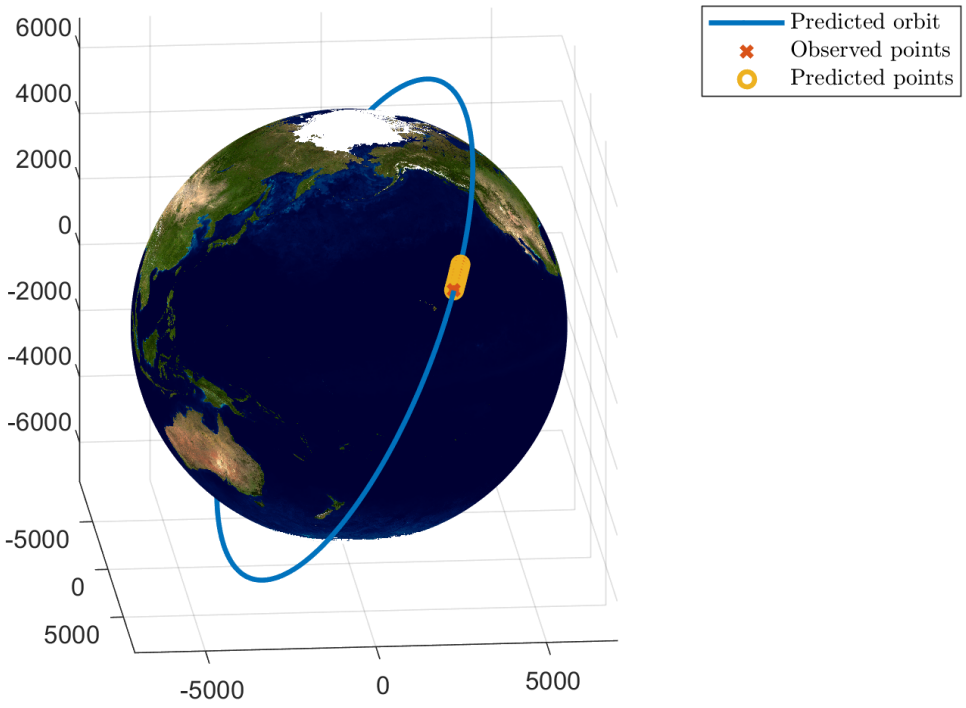


Figure 7: ECI plot showing observations over Kaena Point, Hawaii, and estimated orbital trajectory. Earth graphic courtesy of `planet3D` function (MathWorks).

Problem 3

Sequential Batch Filter

After you finish running the data for problem 2, the site sends you more observations (see problem 3 text file). The data was taken from the same site with the same noise values. Find the new state and P matrix. How much did it change?

Solution.

See Appendix C for results and code for Problem 3

This problem was completed using a Sequential Batch Filtering technique. After processing the initial batch of data (see Problem 2), another group of observations comes in. The general process is:

- Find $H_n^T W_n H_n$ and $H_n^T W_n y_n$ matrices using new observations
- Utilize the $H_k^T W_k H_k$ and $H_k^T W_k y_k$ matrices from the first batch filtering sequence
- Find updated state estimate using the new equations for $\hat{\delta} x_{k+n}$ and $\hat{P}_{k+n}$ using Schur's identity.

Sequential Batch Filtering Results

Before adding the new observational data to the old, the covariance matrix was first checked *before* performing the finite differencing scheme to evaluate the general quality of the new data (see Fig. 8).

```
The first covariance matrix (Pk) was:
      0.07334     0.0054449    -0.0081869   -0.00086739   -3.5915e-05    4.2618e-05
    0.0054449      0.039581     -0.061185   -1.1602e-05   -0.00044222    0.00065953
   -0.0081869     -0.061185       0.10625    4.8006e-05    0.00069256    -0.0012172
  -0.00086739   -1.1602e-05    4.8006e-05    1.4036e-05   -4.3312e-07   -9.3701e-07
  -3.5915e-05   -0.00044222    0.00069256   -4.3312e-07    6.0007e-06   -9.5418e-06
   4.2618e-05    0.00065953    -0.0012172   -9.3701e-07   -9.5418e-06    1.9981e-05

After the initial batch processing of the new data, Pn is:
     0.050639     -0.01263     -0.011531   -0.00079644    0.00020843     0.0001275
     -0.01263     0.010619      -0.01258    0.00023007   -0.00018533    0.00016097
    -0.011531     -0.01258      0.079692    0.00013047    0.00019336    -0.0011991
  -0.00079644    0.00023007    0.00013047    1.7554e-05    -6.123e-06   -2.3662e-06
   0.00020843   -0.00018533    0.00019336    -6.123e-06    4.8563e-06   -1.6435e-06
    0.0001275    0.00016097    -0.0011991   -2.3662e-06   -1.6435e-06    2.6393e-05

Differencing the two (Pk - Pn) gives:
     0.022701     0.018075     0.0033442   -7.0949e-05   -0.00024434   -8.4877e-05
     0.018075     0.028962     -0.048605   -0.00024167   -0.00025688    0.00049856
    0.0033442    -0.048605      0.026559   -8.2467e-05    0.0004992   -1.8064e-05
  -7.0949e-05   -0.00024167   -8.2467e-05   -3.5185e-06    5.6899e-06    1.4292e-06
  -0.00024434   -0.00025688    0.0004992    5.6899e-06    1.1443e-06   -7.8983e-06
  -8.4877e-05    0.00049856   -1.8064e-05    1.4292e-06   -7.8983e-06   -6.4123e-06

~~~~~~~ Covariance matrix looks good. Sequential batch filter next. ~~~~~~~
```

Figure 8: New data was analyzed before implementation into the finite differencing scheme for assimilation into the new data set. In essence, how good is the new data compared to the original set? The new data had a good (low) covariance matrix and was thus included in the state estimation sequential batch filtering process.

The new data was good; evidenced by lower values in the covariance matrix. After the sequential batch filtering process, the overall covariance matrix decreased, 1σ standard deviations (in all three directions) decreased, and the error ellipsoid decreased in all three directions. The new covariance matrix is given in Fig. 9, and final statistical values and error ellipsoid dimensions are given in Fig. 10.

The state estimate from the old data, X_k, and the final state after sequential batch filtering X_{k+n}, are given below:

$$\mathbf{X}_k = \begin{bmatrix} 5748.9 \\ 2679.8 \\ 3442.9 \\ 4.3287 \\ -1.9204 \\ -5.7262 \end{bmatrix} \qquad \mathbf{X}_{k+n} = \begin{bmatrix} 5993.5 \\ 2489 \\ 2960.7 \\ 4.0701 \\ -0.62185 \\ -5.1436 \end{bmatrix}$$

```
Sequential batch filter covariance matrix is:
     0.030773   0.00096577   -0.0036746  -0.00041721  -4.8786e-07   3.0705e-05
   0.00096577     0.016376    -0.023428   6.6367e-06  -0.00020396   0.00028168
   -0.0036746    -0.023428      0.03921   3.4599e-05   0.00029592   -0.0005074
  -0.00041721   6.6367e-06   3.4599e-05   7.9671e-06  -4.8245e-07  -7.5868e-07
  -4.8786e-07  -0.00020396   0.00029592  -4.8245e-07   3.1865e-06  -4.7648e-06
   3.0705e-05   0.00028168   -0.0005074  -7.5868e-07  -4.7648e-06   1.0193e-05

*** Much better than the original set, and better than the new set alone! ****
```

Figure 9: Final covariance matrix, P_n, after sequential batch filtering.

```
CONFIDENCE ELLIPSOID for sequential batch filtered data:
One sigma stdDev for I direction is: 175.4232 m
One sigma stdDev for J direction is: 127.9682 m
One sigma stdDev for K direction is: 198.014 m
Eigenvalues are:
     0.054412            0            0
            0      0.03026            0
            0            0    0.0016866

Eigenvectors are:
      0.15185      0.98763     0.039119
      0.52115     -0.11363      0.84586
     -0.83985      0.10805      0.53196

Dimensions of the error ellipsoid are:
rI direction: 233.2636 m
rJ direction: 173.9545 m
rK direction: 41.0688 m
```

Figure 10: Confidence ellipsoid after sequential batch filtering process. Note that the 1σ standard deviation has decreased with the addition of more high quality data points.

Changes in 1σ standard deviation from sequential batch filtering versus the original data alone (in %) are:

$$\Delta\sigma_x = -46.183$$

$$\Delta\sigma_y = -35.678$$

$$\Delta\sigma_y = -26.882$$

Changes in error ellipsoid (in %) are:

$$\Delta\epsilon_x = -38.521$$

$$\Delta\epsilon_y = -35.161$$

$$\Delta\epsilon_z = -27.798$$

In summary, the new data enhanced the estimation scheme, and the sequential batch filtering process greatly sped up the process.

References

[1] David A Vallado. *Fundamentals of astrodynamics and applications, Third edition.* Microcosm Press, 2007.

Appendix A: Problem 1 Results and Code

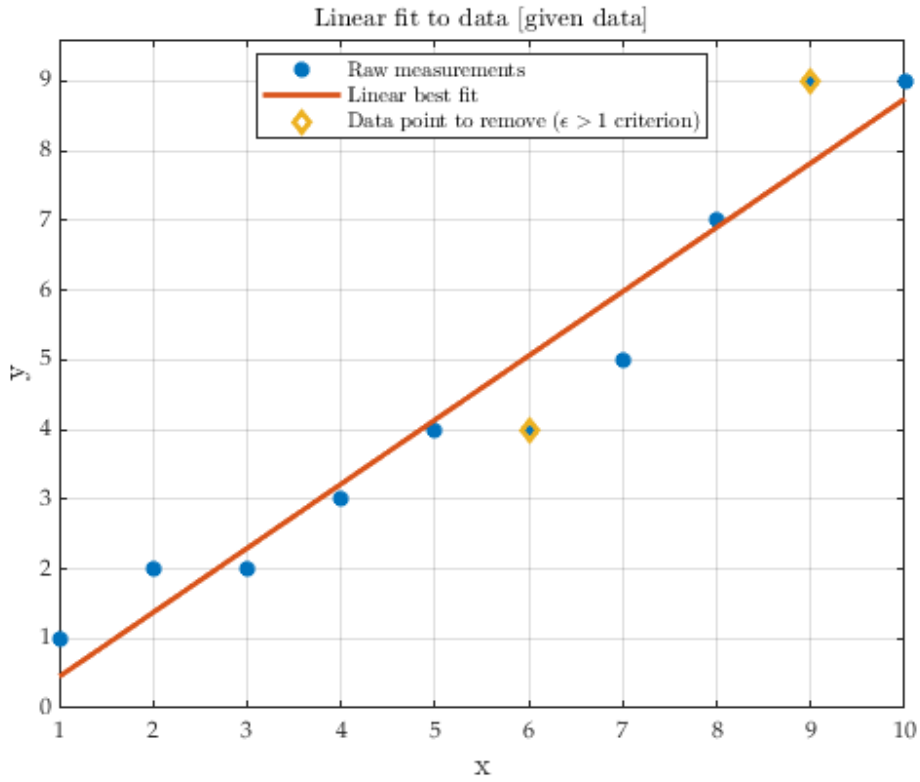
Table of Contents

HW 2.1: Least Squares

```
*** HW 2.1: Least Squares [linear assumption] ***

~~~~~~~~~~~~~~~~ GIVEN DATA ~~~~~~~~~~~~~~~~
Best estimate of the state is:
xhat =
     -0.46667
      0.92121

Linear model (estimated) is: y = -0.46667 + 0.92121*xoi
```

Editing data

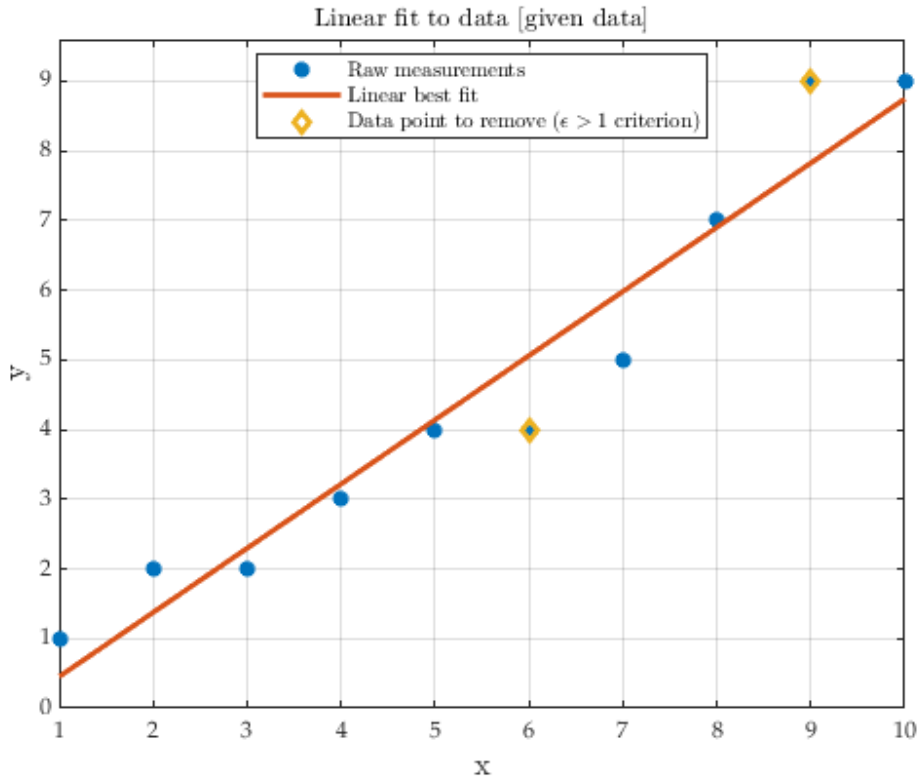
```
Confidence intervals (original dataset):
alpha = -0.46667 +- 0.68313
beta = 0.92121 +- 0.1101
```

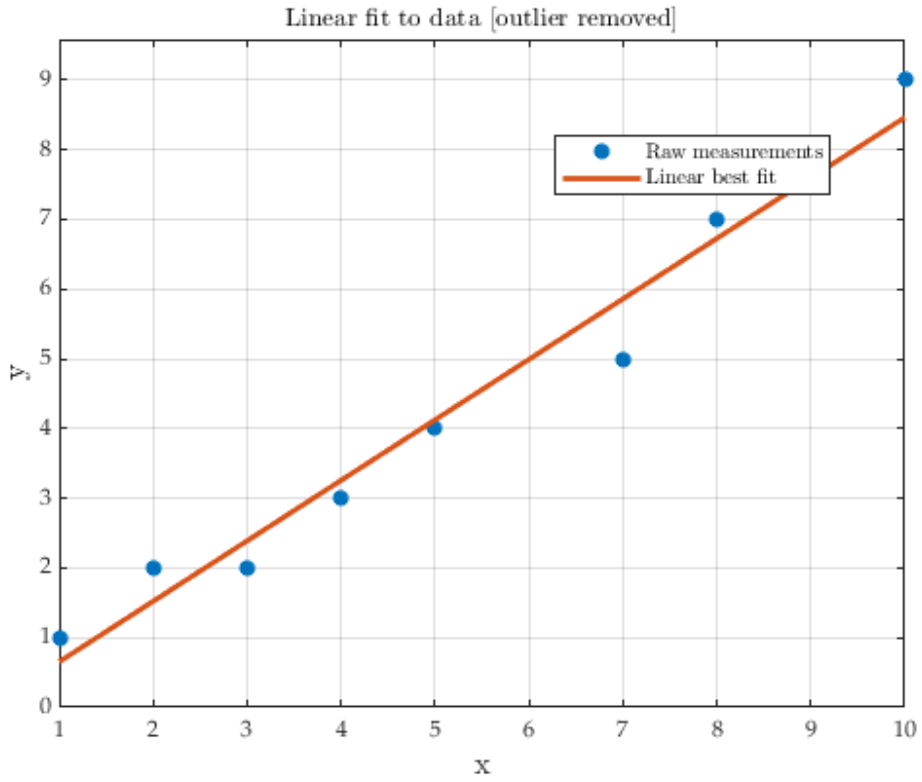


Change of state if data point(s) removed

UPDATED (outliers removed) data

```
~~~~~~~~~~~~~~~~ OUTLIERs (6,9) REMOVED ~~~~~~~~~~~~~~~~
Best estimate of the state [outlier removed] is:
xhat =
     -0.21324
      0.86765

Linear model [outlier removed] is: y = -0.21324 + 0.86765*xoi
```


Linear fit to data [given data]
Raw measurements
Linear best fit
Data point to remove (ϵ > 1 criterion)
y
x
0
1
2
3
4
5
6
7
8
9
10


Linear fit to data [outlier removed]
Raw measurements
Linear best fit
y
x
0
1
2
3
4
5
6
7
8
9
10

Confidence intervals (updated dataset)

```
Confidence intervals (after removing bad data): <----------------
alpha = -0.21324 +/- 0.70189
beta = 0.86765 +/- 0.12127
```

Published with MATLAB® R2023b

Table of Contents


```
%{
Justin Self
California Polytechnic State University
AERO557 | Advanced Orbital Mechanics
Dr. Abercromby, WINTER 2024


HW #2: Least Squares, Weighted Least Squares, Sequential Batch Filtering
%}

clear all; close all; clc;
addpath('C://MATLAB_CODE/Orbits/')
```

HW 2.1: Least Squares

```
disp("*** HW 2.1: Least Squares [linear assumption] ***")
disp(" ")
% Given data
xoi = [1 2 3 4 5 6 7 8 9 10]; %HW data
yoi = [1 2 2 3 4 4 5 7 9 9];

%..... Find best estimate of state assuming a linear fit; y = alpha + Beta*x
% Call homemade function
[y,xhat,alpha,beta,Hbar] = leastsquares_linear(xoi,yoi);

%.... report best state estimate
disp("~~~~~~~~~~~~~~~~ GIVEN DATA ~~~~~~~~~~~~~~~~")
disp("Best estimate of the state is:")
disp("xhat = ")
disp(xhat)
disp("Linear model (estimated) is: y = " + alpha + " + " + beta + "*xoi")
```

Editing data

```
%.... Determine residuals (errors)
err = yoi - y;
n = length(xoi);

%..... RMS
sumSqErr = sum(err.^2);
RMS = sqrt(1/n * sumSqErr);
```

```
% For criteria of Err > 1 (2*RMS or 1*RMS did not yield any residuals),
remove data point(s).
criteria = 1;
z = find(abs(err) > criteria);

% Covariance matrix P
P = inv(Hbar'*Hbar);

sigmax = sqrt(P(1,1));
sigmay = sqrt(P(2,2));

% Print
disp(" ")
disp("Confidence intervals (original dataset):")
disp("alpha = " + alpha + " +- " + sigmax)
disp("beta = " + beta + " +- " + sigmay)

% Plot results
figure()
plot(xoi,yoi,'*',"LineWidth",2)
hold on
plot(xoi,y,'LineWidth',2)
plot(xoi(z),yoi(z),'diamond','LineWidth',2)

% Graph pretty
ylim padded
xlim tight
xLab = xlabel('x','Interpreter','latex');
yLab = ylabel('y','Interpreter','latex');
plotTitle = title('Linear fit to data [given data]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set([xLab, yLab],'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set([xLab, yLab],'FontSize', 12)
grid on
legend('Raw measurements','Linear best fit', 'Data point to remove ($
\epsilon > 1 $ criterion)','interpreter','latex','Location', 'best')
```

Change of state if data point(s) removed

UPDATED (outliers removed) data

```
xoi = [1 2 3 4 5 7 8 10]; % two points removed (hard-coded)
yoi = [1 2 2 3 4 5 7 9];

% Call function again
[y,xhat,alpha,beta,Hbar] = leastsquares_linear(xoi,yoi);

%.... report best state estimate
disp(" ")
disp("~~~~~~~~~~~~~~~~ OUTLIERs (6,9) REMOVED ~~~~~~~~~~~~~~~~")
```

```
disp("Best estimate of the state [outlier removed] is:")
disp("xhat = ")
disp(xhat)
disp("Linear model [outlier removed] is: y = " + alpha + " + " + beta +
"*xoi")

% Plot results
figure()
plot(xoi,yoi,'*',"LineWidth",2)
hold on
plot(xoi,y,'LineWidth',2)

% Graph pretty
ylim padded
xlim tight
xLab = xlabel('x','Interpreter','latex');
yLab = ylabel('y','Interpreter','latex');
plotTitle = title('Linear fit to data [outlier
removed]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set([xLab, yLab],'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set([xLab, yLab],'FontSize', 12)
grid on
legend('Raw measurements','Linear best fit',
'interpreter','latex','Location', 'best')
```

Confidence intervals (updated dataset)

```
%.... Determine residuals (errors)
err = yoi - y;
n = length(xoi);

% Covariance matrix P
P = inv(Hbar'*Hbar);

sigmax = sqrt(P(1,1));
sigmay = sqrt(P(2,2));

% Print
disp(" ")
disp("Confidence intervals (after removing bad data): <----------------")
disp("alpha = " + alpha + " +/- " + sigmax)
disp("beta = " + beta + " +/- " + sigmay)
```

Published with MATLAB® R2023b

Appendix B: Problem 2 Results and Code

Table of Contents

HW 2.2: Weighted Least Squares

```
*** HW 2.2: Weighted Least Squares ***
```

Find Rsite (IJK)

```
Heart checks on State @ 1 from obs. 4, 5, and 6: [should be simliar]
                          r1      r1 (from r4)    r1(from r5)    r1(from r6)
                       _______    ____________    ___________    ___________

    Position [km]      7217.5       7217.3         7216.5         7216.4
    Velocity [km/s]         0       7.4338         7.4337         7.4186

~~~OK
Xnom (before finite differencing) is [km, km/s]:
       5753.2
       2673.5
       3440.6
       4.3231
       -1.927
      -5.7256

Initial guess error between Vallado and Justin
errR is: -3.0392 km
errV is: -0.00082403 km/s
Not too bad. Likely could improve position by using 'rigorous' rho_ECI
formulation.

 ~~~~~ RUN FINITE DIFF LOOP ~~~~~
```

Preallocate for loop

Differential correction loop

```
Xnom (after finite differencing) is [km, km/s]:
```

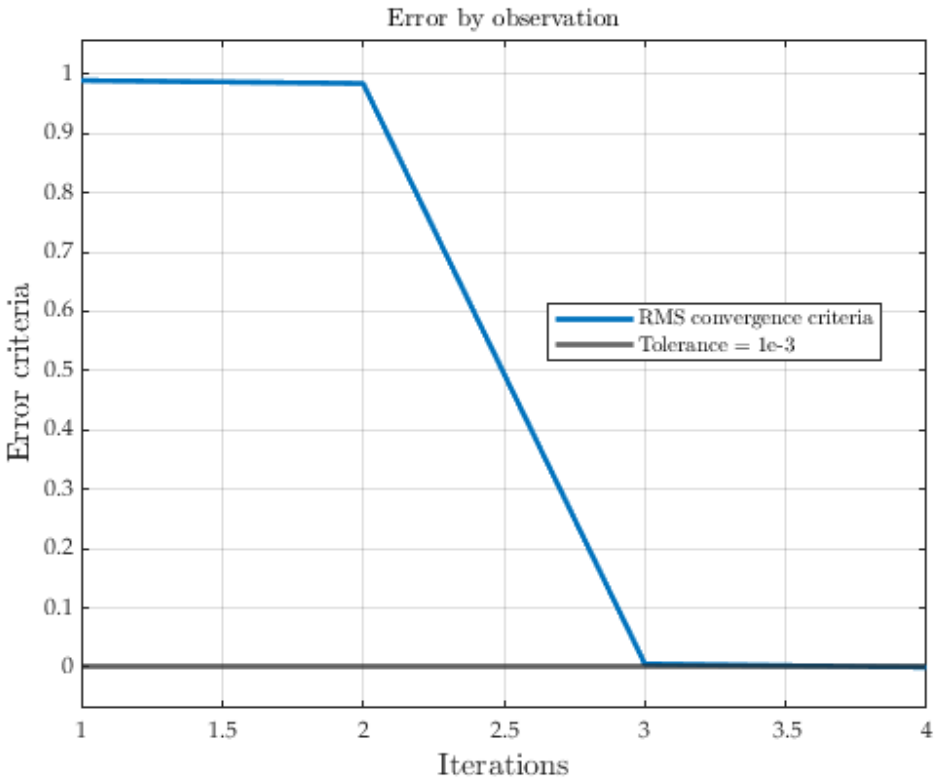
```
       5748.9
       2679.8
       3442.9
       4.3287
      -1.9204
      -5.7262
```

Confidence ellipsoid

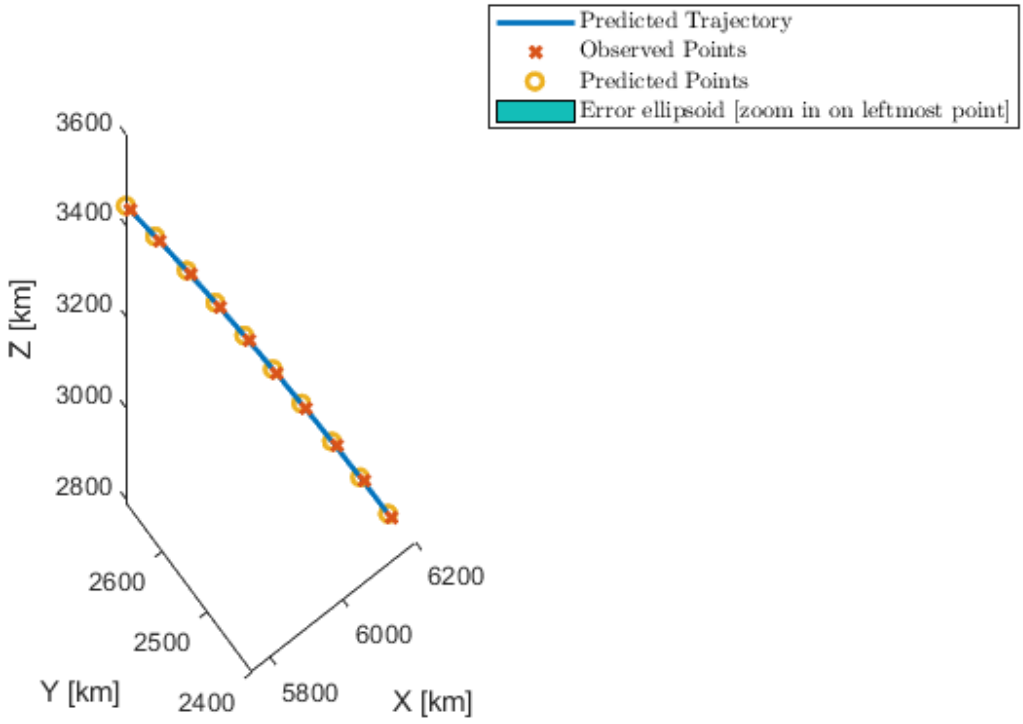
```
Covariance matrix, P, is:
      0.07334    0.0054449   -0.0081869  -0.00086739  -3.5915e-05
4.2618e-05
    0.0054449     0.039581    -0.061185  -1.1602e-05  -0.00044222
0.00065953
   -0.0081869    -0.061185      0.10625   4.8006e-05   0.00069256
-0.0012172
  -0.00086739  -1.1602e-05   4.8006e-05   1.4036e-05  -4.3312e-07
-9.3701e-07
  -3.5915e-05  -0.00044222   0.00069256  -4.3312e-07   6.0007e-06
-9.5418e-06
   4.2618e-05   0.00065953   -0.0012172  -9.3701e-07  -9.5418e-06
1.9981e-05

One sigma stdDev for I direction is: 270.8143 m
One sigma stdDev for J direction is: 198.9495 m
One sigma stdDev for K direction is: 325.9605 m
Eigenvalues are:
    0.0032354            0            0
            0     0.071979            0
            0            0      0.14396

Eigenvectors are:
   -0.0072111      0.99044     -0.13773
      0.86001    -0.064134     -0.50623
      0.51023       0.1221      0.85133

Dimensions of the error ellipsoid are:
rI direction: 56.8804 m
rJ direction: 268.2882 m
rK direction: 379.4175 m
```

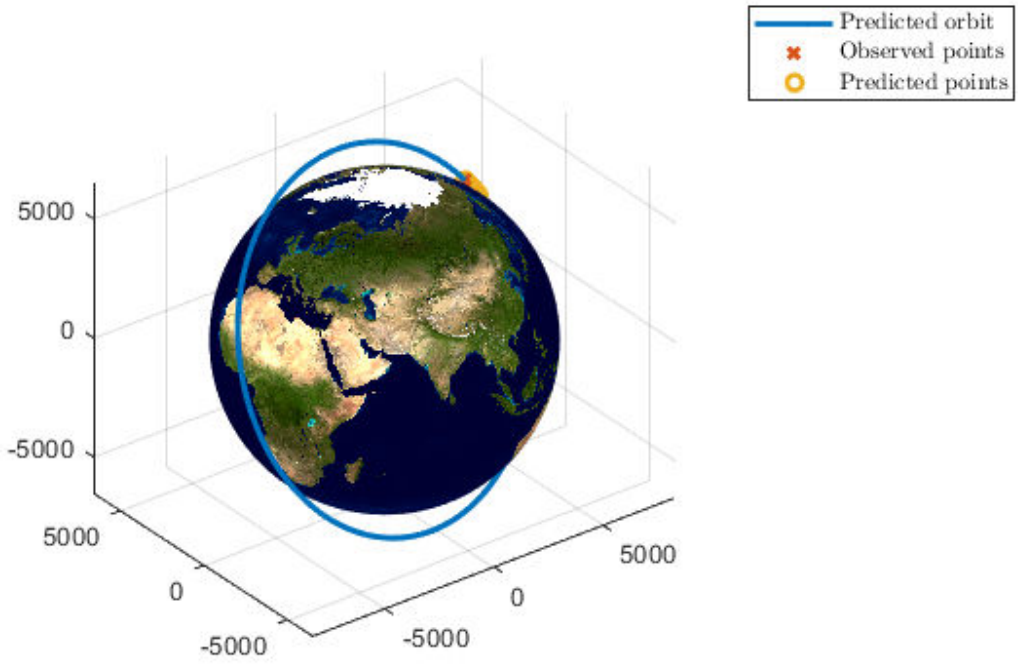
Plotting

ERROR VS. ITERATION PLOT



n finite differencing corrections over observation period





Published with MATLAB® R2023b

```
%{
Justin Self
California Polytechnic State University
AERO557 | Advanced Orbital Mechanics
Dr. Abercromby, WINTER 2024

HW #2 (parts 2 and 3): Weighted Least Squares, Sequential Batch Filtering
%}
clear all; close all; clc;

mu = 398600; % km3/s2

% Add Path
addpath('C://MATLAB_CODE/Orbits/')
%addpath("Functions/")    % Travis'
%addpath("Data/")         % Travis'
```

HW 2.2: Weighted Least Squares

```
disp("*** HW 2.2: Weighted Least Squares ***")
disp(" ")
% Import data
data = readmatrix("Data4Problem2Hw2test2.txt");
year = data(:,1);
m = data(:,2);
d = data(:,3);
UT = data(:,4:6);
az = data(:,7);                    % deg
el = data(:,8);                    % deg
rho = data(:,9);  % range          % km
lat = deg2rad(21.5721);           % degrees
lon = deg2rad(-158.2666);         % degrees
alt = 300.2;                      % m
rhoErr = 92.5/1000;               % km
azErr = deg2rad(0.0224);          % rad
elErr = deg2rad(0.0139);          % rad

UTC = [year,m,d,UT];
UTC0 = UTC;
bias = [80e-3,0.0081,0.0045]; % km, deg [rho, az, el]

%..... W
% Build and normalize weight matrix (km; RADIANS)
W = [1/rhoErr^2   0           0;
     0              1/(azErr)^2   0;
     0              0           1/(elErr)^2];


% Normalize Weight matrix
Wnormalized = W * norm(W)^-1;
```

Find Rsite (IJK)

```
rECI = zeros(3,10);
jd = zeros(10,1);

for i = 1:10
    [RsiteECI, RsiteECEF] = Rsite(rad2deg(lat),rad2deg(lon),alt,UTC(i,:));
    rhoECI =
rhoVectECI(az(i),el(i),rho(i),rad2deg(lat),rad2deg(lon),UTC(i,:));
    rECI(:,i) = RsiteECI + rhoECI;
    jd(i) = juliandate(datetime(UTC(i,:)));
end

%.... Group A
    r1 = rECI(:,1);
    [t1, ~, ~] = juliandateFunction(year(1),m(1),d(1),UT(1,:));
    r4 = rECI(:,4);
    [t4, ~, ~] = juliandateFunction(year(4),m(4),d(4),UT(4,:));
    r7 = rECI(:,7);
    [t7, ~, ~] = juliandateFunction(year(7),m(7),d(7),UT(7,:));

    % Velocity
    [v4, ~,~,~, ~ ] = hgibbs(r1,r4,r7,t1,t4,t7);

    % r4, v4 to v7
    dt = (t7 - t4)*86400;
    [~,v7] = universalVar(mu,dt,r4,v4);

    % Propagate r4 back to r1
        % Time from t1 - t4 (negative)
        dt1 = (t1 - t4)*86400; % days to seconds
        [r1_4,v1_4] = universalVar(mu,dt1,r4,v4);
        v1 = v1_4; % redefine for simplicity later.

%.... Group B
    r2 = rECI(:,2);
    [t2, ~, ~] = juliandateFunction(year(2),m(2),d(2),UT(2,:));
    r5 = rECI(:,5);
    [t5, ~, ~] = juliandateFunction(year(5),m(5),d(5),UT(5,:));
    r8 = rECI(:,8);
    [t8, ~, ~] = juliandateFunction(year(8),m(8),d(8),UT(8,:));
    % Velocity
    [v5, ~,~,~, ~ ] = hgibbs(r2,r5,r8,t2,t5,t8);
    % r5, v5 to v2
    dt = (t2 - t5)*86400; % backward from 5 to 2
    [~,v2] = universalVar(mu,dt,r5,v5);

    % r5, v5 to v8
    dt = (t8 - t5)*86400;
    [~,v8] = universalVar(mu,dt,r5,v5);

    % Propagate r5 back to r1
        % Time from t1 - t5 (negative)
```

```
        dt5 = (t1 - t5)*86400; % days to seconds
        [r1_5,v1_5] = universalVar(mu,dt5,r5,v5);

% Group C
    r3 = rECI(:,3);
    [t3, ~, ~] = juliandateFunction(year(3),m(3),d(3),UT(3,:));
    r6 = rECI(:,6);
    [t6, ~, ~] = juliandateFunction(year(6),m(6),d(6),UT(6,:));
    r9 = rECI(:,9);
    [t9, ~, ~] = juliandateFunction(year(9),m(9),d(9),UT(9,:));
    % Velocity
    [v6, ~,~,~, ~ ] = hgibbs(r3,r6,r9,t3,t6,t9);
    % r6, v6 to v3
    dt = (t3 - t6)*86400; % back from 6 to 3
    [~,v3] = universalVar(mu,dt,r6,v6);
    % r6, v6 to v9
    dt = (t9 - t6)*86400;
    [~,v9] = universalVar(mu,dt,r6,v6);

        % Propagate r6 back to r1
        % Time from t1 - t6 (negative)
        dt6 = (t1 - t6)*86400; % days to seconds
        [r1_6,v1_6] = universalVar(mu,dt6,r6,v6);

disp("Heart checks on State @ 1 from obs. 4, 5, and 6: [should be simliar]")
viewtable = table(...
    [norm(r1);0],...
    [norm(r1_4);norm(v1_4)],...
    [norm(r1_5);norm(v1_5)],...
    [norm(r1_6);norm(v1_6)],...
    'VariableNames',{'r1','r1 (from r4)','r1(from r5)','r1(from
r6)'},'RowName',{'Position [km]','Velocity [km/s]'});
disp(viewtable)
disp("~~~OK")

%.... If all looks good, average each all values to determind X_nom.
Rnom = (r1 + r1_4 + r1_5 + r1_6) / 4;
Vnom = (v1_4 + v1_5 + v1_6) / 3;

format short g
Xnom = [Rnom; Vnom]; % based on time 1.
disp("Xnom (before finite differencing) is [km, km/s]: ")
disp(Xnom)

% Validate against Vallado
RnomVallado = [5975.2904; 2568.6400; 3120.5845];
VnomVallado = [3.983846; -2.071159; -5.917095];

disp("Initial guess error between Vallado and Justin")
errR = norm(RnomVallado) - norm(Rnom);
errV = norm(VnomVallado) - norm(Vnom);
disp("errR is: " + errR + " km")
disp("errV is: " + errV + " km/s")
disp("Not too bad. Likely could improve position by using 'rigorous' rho_ECI
```

```
formulation.")
disp(" ")
disp(" ~~~~~ RUN FINITE DIFF LOOP ~~~~~")

%}
RnomVallado = [5975.2904; 2568.6400; 3120.5845];
VnomVallado = [3.983846; -2.071159; -5.917095];

% ** For testing use Vallado's Xnom **
%Xnom = [RnomVallado;VnomVallado];
```

Preallocate for loop

```
maxiteration = 10; % expect <10

% Get into loop and preallocate
RMSprev = 2;
RMScurrent = 1;
criteria = abs((RMSprev - RMScurrent)/RMSprev);
ii = 1; % iteration counter
lla0 = [rad2deg(lat),rad2deg(lon),alt];
HtWH_accumulated = zeros(6,6);
HtWy_accumulated = zeros(6,1);
RMSterm_accumulated = 0;
Xnom_init = Xnom;
```

Differential correction loop

```
while criteria > 1e-3 && (ii < 10)

    for i = 1:length(data)
        % reset H and y matrices
        H = zeros(6,3);
        y = zeros(1,3);

        % Propagate nominal state foward to time (i)
        dt = 86400*(jd(i) - jd(1));
        [rCalc,vCalc] = universalVar(mu,dt,Xnom(1:3),Xnom(4:6));

        % Calculate observations (rho, az, el)
        UTC = [data(i,1);data(i,2);data(i,3);data(i,4);data(i,5);data(i,6)];
        aer = eci2aer(1000*rCalc',UTC',lla0);
        Az = deg2rad(aer(1));
        El = deg2rad(aer(2));
        Rho = aer(3)/1000;
        calc = [Rho;Az;El];
        obs = [data(i,9);deg2rad(data(i,7));deg2rad(data(i,8))];
        % Take difference
        y = (obs - calc);
        state = [rCalc;vCalc];

        % Finite differencing
        for j = 1:length(state)
```

```
            % Create modified state, perturbed elementwise
            del = Xnom(j)*0.001;
            Xmod = Xnom;
            Xmod(j) = Xnom(j) + del;

            % Propagate both states forward to obs time
            dt = 86400*(jd(i) - jd(1));
            [r_mod,~] = universalVar(mu,dt,Xmod(1:3),Xmod(4:6));
            % Propagate Xnom
            [r_nom,~] = universalVar(mu,dt,Xnom(1:3),Xnom(4:6));

            % Calculate observations, perturbed state
            aer = eci2aer(1000*r_mod',UTC0(i,:),lla0);
            Az = deg2rad(aer(1));
            El = deg2rad(aer(2));
            Rho = aer(3)/1000;
            mod = [Rho;Az;El];

            % Calculate observations, nominal state
            aer = eci2aer(1000*r_nom',UTC0(i,:),lla0);
            Az = deg2rad(aer(1));
            El = deg2rad(aer(2));
            Rho = aer(3)/1000;
            nom = [Rho;Az;El];

            % Build H matrix
            H(j,1:3) = (mod - nom) / del;

        end
        H = H';

        % Compute matrices
        HtWH = H'*W*H; % 6x6
        HtWy = H'*W*y; % 6x1

        % Store
        HtWH_accumulated = HtWH + HtWH_accumulated;
        HtWy_accumulated = HtWy + HtWy_accumulated;
        RMSterm_accumulated = y'*Wnormalized*y + RMSterm_accumulated;

    end

    % Find P
    P = HtWH_accumulated^-1;

    % Calc xhat
    xhat = P*HtWy_accumulated;

    % Update nominal state
    Xnom = Xnom + xhat;

    % Find error
    obs = 9; % number of observations (9)
    N = 3; % number of observation elements (rho, az, el)
```

```
    RMSprev = RMScurrent;
    RMScurrent = sqrt(RMSterm_accumulated / (obs*N));
    criteria = abs((RMSprev - RMScurrent) / RMSprev);
    err(ii) = criteria;

    % Reset storage matrices
    HtWH_accumulated = zeros(6,6);
    HtWy_accumulated = zeros(6,1);
    RMSterm_accumulated = 0;

    % update count
    ii = ii + 1;

end % (ii) big while loop

disp(" ")
disp("Xnom (after finite differencing) is [km, km/s]: ")
disp(Xnom)
```

Confidence ellipsoid

```
disp("Covariance matrix, P, is: ")
disp(P)

% Extract sigmas from top 3x3 of P
onesigI = sqrt(P(1,1)) * 1000; % km to meters
onesigJ = sqrt(P(2,2)) * 1000;
onesigK = sqrt(P(3,3)) * 1000;
disp("One sigma stdDev for I direction is: " + onesigI + " m")
disp("One sigma stdDev for J direction is: " + onesigJ + " m")
disp("One sigma stdDev for K direction is: " + onesigK + " m")

onesigTot = onesigI + onesigJ + onesigK;

%disp("Total standard deviation is: " + onesigTot)
% Vallado has very similar values, but says "for a total of about 470 m"
% HOW? -- per dr A; don't mess with it.

% Calculate error ellipsoid using eigenvalues and eigenvectors
P_modified = P([1,2,3],[1,2,3]); % extract the upper 3x3 from P
[V,D] = eig(P_modified);

disp("Eigenvalues are: ")
disp(D)

disp("Eigenvectors are: ")
disp(V)

% Determine error ellipsoid
errorEI = sqrt(D(1,1))*1000; % km to meters
errorEJ = sqrt(D(2,2))*1000;
errorEK = sqrt(D(3,3))*1000;
```

```
disp("Dimensions of the error ellipsoid are: ")
disp("rI direction: " + errorEI + " m")
disp("rJ direction: " + errorEJ + " m")
disp("rK direction: " + errorEK + " m")

% See below for ellipsoid plots.
```

Plotting

ERROR VS. ITERATION PLOT

```
figure()
plot(err,'LineWidth',2)
hold on
yline(1e-3,'LineWidth',2)

% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Iterations','Interpreter','latex');
yLab = ylabel('Error criteria','Interpreter','latex');
plotTitle = title('Error by observation','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set([xLab, yLab],'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set([xLab, yLab],'FontSize', 12)
grid on
legend('RMS convergence criteria','Tolerance = 1e-3',
'interpreter','latex','Location', 'best')

% Plot ECI observation locations vs. model locations
options = odeset('RelTol', 1e-8, 'AbsTol',1e-8);

% Get r vector for predictions
Rpredict = zeros(3,10);
for i = 1:10
dt = (jd(i)-jd(1))*86400;
[Rpredict(:,i),~] = universalVar(mu,dt,Xnom(1:3),Xnom(4:6));
end

% Set up ode and propagate orbit
tspan = [86400*jd(1) 86400*jd(end)]; % over observation period
state = Xnom;
[~, statenew] = ode45(@pig, tspan, state, options, mu);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% OBSERVATIONS vs MODEL PLOT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure()
plot3(statenew(:,1),statenew(:,2),statenew(:,3),'LineWidth',2)
hold on
plot3(rECI(1,:),rECI(2,:),rECI(3,:),'x','LineWidth',2)
```

```
plot3(Rpredict(1,:),Rpredict(2,:),Rpredict(3,:),'o','LineWidth',2)

% Plot error ellipsoids
%{
[X,Y,Z] = ellipsoid(xc,yc,zc,xr,yr,zr) returns the x-, y-, and z-coordinates
of an ellipsoid without drawing it. The returned ellipsoid has center
 coordinates at (xc,yc,zc), semiaxis lengths (xr,yr,zr), and consists of
 20-by-20 faces.
%}

% FIRST OBSERVATION
% Center
xc = Rpredict(1,1);
yc = Rpredict(2,1);
zc = Rpredict(3,1);

% Radii (in km)
xr = errorEI/1000;
yr = errorEJ/1000;
zr = errorEK/1000;

% Plot it
ellipsoid(xc,yc,zc,xr,yr,zr);

legend("Predicted Trajectory","Observed Points","Predicted Points","Error
ellipsoid [zoom in on leftmost point]",'interpreter','latex')
title("Results from finite differencing corrections over observation
period",'interpreter','latex')
xlabel("X [km]")
ylabel("Y [km]")
zlabel("Z [km]")

% FULL TRAJECTORY PLOT
tspan = [0 1e4]; % full traj
init_coast = Xnom;
[~, statenew] = ode45(@pig, tspan, init_coast, options, mu);

figure()
% Earth
   hold on
opts.Units = 'km';
opts.RotAngle = 175; % get hawaii under observations for reality
units = opts.Units;
planet3D('Earth',opts);
grid on
    hold on
    plot3(statenew(:,1),statenew(:,2),statenew(:,3),'LineWidth',2)
    plot3(rECI(1,:),rECI(2,:),rECI(3,:),'x','LineWidth',2)
    plot3(Rpredict(1,:),Rpredict(2,:),Rpredict(3,:),'o','LineWidth',2)
    legend("","Predicted orbit","Observed points","Predicted
points",'interpreter','latex')
```

OBSERVATION DATA

```
%{
1995 01 29  02 38 37.00   60.4991  16.1932  2047.50200
1995 01 29  02 38 49.00   62.1435  17.2761  1984.67700
1995 01 29  02 39 02.00   64.0566  18.5515  1918.48900
1995 01 29  02 39 14.00   65.8882  19.7261  1859.32000
1995 01 29  02 39 26.00   67.9320  20.9351  1802.18600
1995 01 29  02 39 38.00   70.1187  22.1319  1747.29000
1995 01 29  02 39 50.00   72.5159  23.3891  1694.89100
1995 01 29  02 40 03.00   75.3066  24.7484  1641.20100
1995 01 29  02 40 15.00   78.1000  25.9799  1594.77000
1995 01 29  02 40 27.00   81.1197  27.1896  1551.64000

%}
```

Published with MATLAB® R2023b

Appendix C: Problem 3 Results and Code

Table of Contents

HW 2.3: Sequential Batch Filtering

**** HW 2.3: Sequential Batch Filtering ****

Find Rsite (IJK)

Differential correction loop (initial batch processing)

Differential correction loop

```
The first covariance matrix (Pk) was:
      0.07334    0.0054449   -0.0081869  -0.00086739  -3.5915e-05
4.2618e-05
    0.0054449     0.039581    -0.061185  -1.1602e-05  -0.00044222
0.00065953
   -0.0081869    -0.061185      0.10625   4.8006e-05   0.00069256
-0.0012172
  -0.00086739  -1.1602e-05   4.8006e-05   1.4036e-05  -4.3312e-07
-9.3701e-07
  -3.5915e-05  -0.00044222   0.00069256  -4.3312e-07   6.0007e-06
-9.5418e-06
   4.2618e-05   0.00065953   -0.0012172  -9.3701e-07  -9.5418e-06
1.9981e-05

After the initial batch processing of the new data, Pn is:
     0.050639     -0.01263    -0.011531  -0.00079644   0.00020843
0.0001275
     -0.01263     0.010619     -0.01258   0.00023007  -0.00018533
0.00016097
    -0.011531     -0.01258     0.079692   0.00013047   0.00019336
-0.0011991
  -0.00079644   0.00023007   0.00013047   1.7554e-05   -6.123e-06
-2.3662e-06
```

```
   0.00020843  -0.00018533   0.00019336   -6.123e-06   4.8563e-06
-1.6435e-06
    0.0001275   0.00016097   -0.0011991  -2.3662e-06  -1.6435e-06
2.6393e-05

Differencing the two (Pk - Pn) gives:
     0.022701     0.018075    0.0033442  -7.0949e-05  -0.00024434
-8.4877e-05
     0.018075     0.028962    -0.048605  -0.00024167  -0.00025688
0.00049856
    0.0033442    -0.048605     0.026559  -8.2467e-05    0.0004992
-1.8064e-05
  -7.0949e-05  -0.00024167  -8.2467e-05  -3.5185e-06   5.6899e-06
1.4292e-06
  -0.00024434  -0.00025688    0.0004992   5.6899e-06   1.1443e-06
-7.8983e-06
  -8.4877e-05   0.00049856  -1.8064e-05   1.4292e-06  -7.8983e-06
-6.4123e-06

~~~~~~~ Covariance matrix looks good. Sequential batch filter next. ~~~~~~~
```

Sequential batch filtering: Differential Correction II

```
Xnom from previous observations is: [km, km/s]
       5748.9
       2679.8
       3442.9
       4.3287
      -1.9204
      -5.7262

Xnom after sequential batch filtering is: [km, km/s]
       5993.5
         2489
       2960.7
       4.0701
     -0.62185
      -5.1436

Sequential batch filter covariance matrix is:
     0.030773   0.00096577   -0.0036746  -0.00041721  -4.8786e-07
3.0705e-05
   0.00096577     0.016376    -0.023428   6.6367e-06  -0.00020396
0.00028168
   -0.0036746    -0.023428      0.03921   3.4599e-05   0.00029592
-0.0005074
  -0.00041721   6.6367e-06   3.4599e-05   7.9671e-06  -4.8245e-07
-7.5868e-07
  -4.8786e-07  -0.00020396   0.00029592  -4.8245e-07   3.1865e-06
-4.7648e-06
   3.0705e-05   0.00028168   -0.0005074  -7.5868e-07  -4.7648e-06
```

```
1.0193e-05

*** Much better than the original set, and better than the new set alone!
****
```

Confidence ellipsoid

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CONFIDENCE ELLIPSOID for sequential batch filtered data:
One sigma stdDev for I direction is: 175.4232 m
One sigma stdDev for J direction is: 127.9682 m
One sigma stdDev for K direction is: 198.014 m
Eigenvalues are:
     0.054412            0            0
            0      0.03026            0
            0            0    0.0016866

Eigenvectors are:
      0.15185      0.98763     0.039119
      0.52115     -0.11363      0.84586
     -0.83985      0.10805      0.53196

Dimensions of the error ellipsoid are:
rI direction: 233.2636 m
rJ direction: 173.9545 m
rK direction: 41.0688 m
```

Conclusion

```
The new data was good. Covariance matrix decreased, one sigma standard
deviations (in all three directions) decreased,
and the error ellipsoid decreased in all three directions.

Percent change in one sigma stDev from original dataset is:
      -35.238
      -35.675
      -39.293

Percent change in error ellipsoid dimensions [m] from original dataset is:
      -38.554
      -35.167
       -27.78
```

Published with MATLAB® R2023b

```
%{
Justin Self
California Polytechnic State University
AERO557 | Advanced Orbital Mechanics
Dr. Abercromby, WINTER 2024

HW #2 Part 3: Sequential Batch Filtering
%}
clear all; close all; clc;

mu = 398600; % km3/s2

% Add Path
addpath('C://MATLAB_CODE/Orbits/')
```

HW 2.3: Sequential Batch Filtering

```
disp("*** HW 2.3: Sequential Batch Filtering ***")
disp(" ")
% Site data
lat = 21.5721; % deg
lon = -158.2666; % deg
alt = 300.2; % m

% Run problem 2.2 again; this time as a function.
% Initial data
data = readmatrix("Data4Problem2Hw2test2.txt");

% Call function (k == original data)
[Xnom_k,P_k,xhat_k, ~,HtWH_k,HtWy_k] =
weightedLeastSquares_HW2_2(data,lat,lon,alt); % confirmed it works

% New data arrives!
data2 = readmatrix("DataforP3H2test.txt");

% Extract values
year = data2(:,1);
m = data2(:,2);
d = data2(:,3);
UT = data2(:,4:6);
az = data2(:,7);                % deg
el = data2(:,8);                % deg
rho = data2(:,9);               % range, km
rhoErr = 92.5/1000;             % km
azErr = deg2rad(0.0224);        % rad
elErr = deg2rad(0.0139);        % rad
UTC = [year,m,d,UT];
UTC0 = UTC;

        % *** Same noise values as last time; same site. ****

% Build and normalize weight matrix (km; RADIANS)
```

```
W = [1/rhoErr^2    0            0;
     0               1/(azErr)^2    0;
     0               0            1/(elErr)^2];

% Normalize Weight matrix
Wnormalized = W * norm(W)^-1;
```

Find Rsite (IJK)

```
rECI = zeros(3,length(rho));
jd = zeros(length(rho),1);

for i = 1:length(rho)
    [RsiteECI, RsiteECEF] = Rsite(lat,lon,alt,UTC(i,:));
    rhoECI = rhoVectECI(az(i),el(i),rho(i),lat,lon,UTC(i,:));
    rECI(:,i) = RsiteECI + rhoECI;
    jd(i) = juliandate(datetime(UTC(i,:)));
end
```

Differential correction loop (initial batch processing)

```
maxiteration = 10; % expect <10

% Get into loop and preallocate
RMSprev = 2;
RMScurrent = 1;
criteria = abs((RMSprev - RMScurrent)/RMSprev);
ii = 1; % iteration counter
lla0 = [lat,lon,alt]; % degrees, m
HtWH_accumulated_n = zeros(6,6);
HtWy_accumulated_n = zeros(6,1);
RMSterm_accumulated_n = 0;

% Rename for simplicity
data = data2;

% Follow same protocol as last time to find H_n.
% Predicted value: known from last time; it is Xnom_k.
Xnom = Xnom_k;
```

Differential correction loop

```
while criteria > 1e-3 && (ii < 10)

    for i = 1:length(rho)
        % reset H and y matrices
        H = zeros(6,3);
        y = zeros(1,3);

        % Propagate nominal state foward to time (i)
```

```
dt = 86400*(jd(i) - jd(1));
[rCalc,vCalc] = universalVar(mu,dt,Xnom(1:3),Xnom(4:6));

% Calculate observations (rho, az, el)
UTC = [data(i,1);data(i,2);data(i,3);data(i,4);data(i,5);data(i,6)];
aer = eci2aer(1000*rCalc',UTC',lla0);
Az = deg2rad(aer(1));
El = deg2rad(aer(2));
Rho = aer(3)/1000;
calc = [Rho;Az;El];
obs = [data(i,9);deg2rad(data(i,7));deg2rad(data(i,8))];
% Take difference
y = (obs - calc);
state = [rCalc;vCalc];

% Finite differencing
for j = 1:length(state)
    % Create modified state, perturbed elementwise
    del = Xnom(j)*0.001;
    Xmod = Xnom;
    Xmod(j) = Xnom(j) + del;

    % Propagate both states forward to obs time
    dt = 86400*(jd(i) - jd(1));
    [r_mod,~] = universalVar(mu,dt,Xmod(1:3),Xmod(4:6));
    % Propagate Xnom
    [r_nom,~] = universalVar(mu,dt,Xnom(1:3),Xnom(4:6));

    % Calculate observations, perturbed state
    aer = eci2aer(1000*r_mod',UTC0(i,:),lla0);
    Az = deg2rad(aer(1));
    El = deg2rad(aer(2));
    Rho = aer(3)/1000;
    mod = [Rho;Az;El];

    % Calculate observations, nominal state
    aer = eci2aer(1000*r_nom',UTC0(i,:),lla0);
    Az = deg2rad(aer(1));
    El = deg2rad(aer(2));
    Rho = aer(3)/1000;
    nom = [Rho;Az;El];

    % Build H matrix
    H(j,1:3) = (mod - nom) / del;

end
H = H';

% Compute matrices
HtWH = H'*W*H; % 6x6
HtWy = H'*W*y; % 6x1

% Store
HtWH_accumulated_n = HtWH + HtWH_accumulated_n;
```

```
        HtWy_accumulated_n = HtWy + HtWy_accumulated_n;
        RMSterm_accumulated_n = y'*Wnormalized*y + RMSterm_accumulated_n;

    end

    % Find P
    P_n = HtWH_accumulated_n^-1;

    % Calc xhat
    xhat_n = P_n*HtWy_accumulated_n;

    % Update nominal state
    Xnom = Xnom + xhat_n;

    % Find error
    obs = 9; % number of observations (9)
    N = 3; % number of observation elements (rho, az, el)
    RMSprev = RMScurrent;
    RMScurrent = sqrt(RMSterm_accumulated_n / (obs*N));
    criteria = abs((RMSprev - RMScurrent) / RMSprev);
    err(ii) = criteria;

    % Reset storage matrices
    HtWH_accumulated_n = zeros(6,6);
    HtWy_accumulated_n = zeros(6,1);
    RMSterm_accumulated_n = 0;

    % update count
    ii = ii + 1;

end % (ii) big while loop

% This was for the initial batch processing. Now look at P_n
disp("The first covariance matrix (Pk) was: ")
disp(P_k)

disp("After the initial batch processing of the new data, Pn is: ")
disp(P_n)

disp("Differencing the two (Pk - Pn) gives: ")
disp(P_k - P_n)

disp("~~~~~~~ Covariance matrix looks good. Sequential batch filter next.
~~~~~~~")
```

Sequential batch filtering: Differential Correction II

```
% Run the exact calculation again; yes its expensive. But I want to display
% the P matrices above.

clear criteria
```

```
RMSprev = 2;
RMScurrent = 1;
criteria = abs((RMSprev - RMScurrent)/RMSprev);
Xnom = Xnom_k;

% This is a copy of the work above, EXCEPT with the BATCH FILTERING.
while criteria > 1e-3 && (ii < 10)

    for i = 1:length(rho)
        % reset H and y matrices
        H = zeros(6,3);
        y = zeros(1,3);

        % Propagate nominal state foward to time (i)
        dt = 86400*(jd(i) - jd(1));
        [rCalc,vCalc] = universalVar(mu,dt,Xnom(1:3),Xnom(4:6));

        % Calculate observations (rho, az, el)
        UTC = [data(i,1);data(i,2);data(i,3);data(i,4);data(i,5);data(i,6)];
        aer = eci2aer(1000*rCalc',UTC',lla0);
        Az = deg2rad(aer(1));
        El = deg2rad(aer(2));
        Rho = aer(3)/1000;
        calc = [Rho;Az;El];
        obs = [data(i,9);deg2rad(data(i,7));deg2rad(data(i,8))];
        % Take difference
        y = (obs - calc);
        state = [rCalc;vCalc];

        % Finite differencing
        for j = 1:length(state)
            % Create modified state, perturbed elementwise
            del = Xnom(j)*0.001;
            Xmod = Xnom;
            Xmod(j) = Xnom(j) + del;

            % Propagate both states forward to obs time
            dt = 86400*(jd(i) - jd(1));
            [r_mod,~] = universalVar(mu,dt,Xmod(1:3),Xmod(4:6));
            % Propagate Xnom
            [r_nom,~] = universalVar(mu,dt,Xnom(1:3),Xnom(4:6));

            % Calculate observations, perturbed state
            aer = eci2aer(1000*r_mod',UTC0(i,:),lla0);
            Az = deg2rad(aer(1));
            El = deg2rad(aer(2));
            Rho = aer(3)/1000;
            mod = [Rho;Az;El];

            % Calculate observations, nominal state
            aer = eci2aer(1000*r_nom',UTC0(i,:),lla0);
            Az = deg2rad(aer(1));
            El = deg2rad(aer(2));
            Rho = aer(3)/1000;
```

```
            nom = [Rho;Az;El];

            % Build H matrix
            H(j,1:3) = (mod - nom) / del;

        end % (j) for element

        H = H';

        % Compute matrices
        HtWH = H'*W*H; % 6x6
        HtWy = H'*W*y; % 6x1

        % Store
        HtWH_accumulated_n = HtWH + HtWH_accumulated_n;
        HtWy_accumulated_n = HtWy + HtWy_accumulated_n;
        RMSterm_accumulated_n = y'*Wnormalized*y + RMSterm_accumulated_n;

    end % (i) for loop BY OBSERVATION

    % Find P
    %P_n = HtWH_accumulated_n^-1;
    P_n_seq = (HtWH_accumulated_n + P_k^-1)^-1;

    % Calc xhat
    %xhat_n = P_n_seq*HtWy_accumulated_n;
    xhat_n_seq = P_n_seq * (HtWy_accumulated_n + HtWy_k);

    % Update nominal state
    Xnom = Xnom + xhat_n_seq;

    % Find error
    obs = 9; % number of observations (9)
    N = 3; % number of observation elements (rho, az, el)
    RMSprev = RMScurrent;
    RMScurrent = sqrt(RMSterm_accumulated_n / (obs*N));
    criteria = abs((RMSprev - RMScurrent) / RMSprev);
    err(ii) = criteria;

    % Reset storage matrices
    HtWH_accumulated_n = zeros(6,6);
    HtWy_accumulated_n = zeros(6,1);
    RMSterm_accumulated_n = 0;

    % update count
    ii = ii + 1;

end % (ii) big while loop

% Rename final state vector
Xnom_n = Xnom;

% Compare sequential batch filter results with initial batch filter
% (New + old vs old obs)
```

```
disp("Xnom from previous observations is: [km, km/s]")
disp(Xnom_k)
disp("Xnom after sequential batch filtering is: [km, km/s]")
disp(Xnom_n)

disp("Sequential batch filter covariance matrix is: ")
disp(P_n_seq)

disp("*** Much better than the original set, and better than the new set
alone! **** ")
```

Confidence ellipsoid

```
disp(" ")
disp("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
disp("CONFIDENCE ELLIPSOID for sequential batch filtered data:")
% Extract sigmas from top 3x3 of Pn
onesigI = sqrt(P_n_seq(1,1)) * 1000; % km to meters
onesigJ = sqrt(P_n_seq(2,2)) * 1000;
onesigK = sqrt(P_n_seq(3,3)) * 1000;
disp("One sigma stdDev for I direction is: " + onesigI + " m")
disp("One sigma stdDev for J direction is: " + onesigJ + " m")
disp("One sigma stdDev for K direction is: " + onesigK + " m")

% Calculate error ellipsoid using eigenvalues and eigenvectors
P_modified_seq = P_n_seq([1,2,3],[1,2,3]); % extract the upper 3x3 from P
[V,D] = eig(P_modified_seq);

disp("Eigenvalues are: ")
disp(D)

disp("Eigenvectors are: ")
disp(V)

% Determine error ellipsoid
errorEI = sqrt(D(1,1))*1000; % km to meters
errorEJ = sqrt(D(2,2))*1000;
errorEK = sqrt(D(3,3))*1000;

disp("Dimensions of the error ellipsoid are: ")
disp("rI direction: " + errorEI + " m")
disp("rJ direction: " + errorEJ + " m")
disp("rK direction: " + errorEK + " m")
```

Conclusion

```
disp("The new data was good. Covariance matrix decreased, one sigma standard
deviations (in all three directions) decreased,")
disp("and the error ellipsoid decreased in all three directions.")
disp(" ")

% How much better?
% --- onesig stdev
```

```
v1 = [270.8719; 198.9401; 326.1777];
v2 = [onesigI; onesigJ; onesigK];
percentChange = 100 .* (v2 - v1) ./ v1;

disp("Percent change in one sigma stDev from original dataset is: ")
disp(percentChange)

%- --- error ellipsoid dims
v1 = [379.6245; 268.3135; 56.8661];
v2 = [errorEI; errorEJ; errorEK];
percentChange = 100 .* (v2 - v1) ./ v1;

disp("Percent change in error ellipsoid dimensions [m] from original dataset
is: ")
disp(percentChange)
```

OBSERVATION DATA (new)

```
%{
1995 01 29  02 40 39.00   84.3708  28.3560  1512.08500
1995 01 29  02 40 51.00   87.8618  29.4884  1476.41500
1995 01 29  02 41 03.00   91.5955  30.5167  1444.91500
1995 01 29  02 41 15.00   95.5524  31.4474  1417.88000
1995 01 29  02 41 27.00   99.7329  32.2425  1395.56300
1995 01 29  02 41 39.00  104.0882  32.8791  1378.20200
1995 01 29  02 41 51.00  108.6635  33.3788  1366.01000
1995 01 29  02 42 03.00  113.2254  33.5998  1359.10000
%}
```

Published with MATLAB® R2023b

Functions Used (all problems)

```
function [y,xhat,alpha,beta,Hbar] = leastsquares_linear(xoi,yoi)

%..... Find best estimate of state assuming a linear fit; y = alpha + Beta*x
n = length(xoi);

% Error = yoi - (alpha + beta*x);
% Hbar = [partial(error) of delE/delalpha , partial(error) of delE/delBeta]
partial_eps_alpha = -1 * ones(length(xoi),1);
partial_eps_beta = -xoi';
Hbar = [partial_eps_alpha, partial_eps_beta];

% Determine xhat == best guess;
% xhat = (H' * H) ^-1 * H' * yoi

% Check for singularity
closetozero = 1e-12;
A = Hbar'*Hbar;

if det(A) < closetozero
    % this matrix is probably singular
    a = pinv(A);
    else
    a = inv((A));
end

b = Hbar'*-1* yoi'; % vallado multiplies by (-1) here; fixes issues with
upside-down fit line; okay; see derivation.
xhat = a * b;

% Extract coefficients from linear fit
alpha = xhat(1);
beta = xhat(2);

% Line of best fit: y = alpha + beta*x;
y = alpha + beta.*xoi;

end % funct
```

Published with MATLAB® R2023b

```
function [RsiteECI, RsiteECEF] = Rsite(lat,lon,alt,UTC)
%{
This function solves for the ECI vector for the site given inputs:
lat = latitude, degrees
lon = longitude, degrees
alt = altitude, meters
UTC = [yyyy,mm,dd,HH,MM,SS];

Author: Justin Self
2/17/2024

%}

addpath('C://MATLAB_CODE/Orbits/')

Re = 6378; % equatorial radius of earth; km
Rp = 6357; % polar radius of the earth; km
f = (Re - Rp) / Re; % oblateness factor
hellp = alt/1000; % altitude of site in km

yyyy = UTC(1);
mm   = UTC(2);
dd   = UTC(3);
HH   = UTC(4);
MM   = UTC(5);
SS   = UTC(6);

UT = [HH,MM,SS];

re = 6378;
rpole = 6357;

LST = local_sidereal_time_justin(yyyy,mm,dd,UT,lon);

% Convert to radians
LST = deg2rad(LST);
lat = deg2rad(lat);
lon = deg2rad(lon);
H = alt/1000;

% Courtesy of Travis Bouck for validation
f = (re-rpole)/re;
c1 = ((re/(sqrt(1-(2*f - f^2)*(sin(lat)^2)))) + H)*cos(lat);
c2 = ((re*(1-f)^2)/(sqrt(1-(2*f - f^2)*(sin(lat)^2))) + H)*sin(lat);

RsiteECI = [c1*cos(LST);c1*sin(LST);c2];

% .... ECEF
Ce = Re / sqrt(1 - (( 2* f - f^2 ) * sin(lat)*sin(lat)));
Se = (1 - (2*f - f^2)) * Ce;
RsiteECEF = [  (Ce + hellp) * cos(lat) * cos(lon);
            (Ce + hellp) * cos(lat) * sin(lon);
```

```
            (Se + hellp) * sin(lat)];
end % Rsite
```

Published with MATLAB® R2023b

```
function rhoVectECI = rhoVectECI(az,el,rho,lat,lon,UTC)

%{
INPUTS:     az: degrees, topo
            el: degrees, topo
            rho = magnitude of slant range vector; km, topo
            lat = degrees
            long = degrees
            UTC: [yyyy,mm,dd,HH,MM,SS]

OUTPUT:
            slant range vector in ECI

Author: Justin Self
2/17/2024

%}

rhosez(:,1) = -rho.*cosd(el).*cosd(az);
rhosez(:,2) = rho.*cosd(el).*sind(az);
rhosez(:,3) = rho.*sind(el);

yyyy = UTC(1);
mm   = UTC(2);
dd   = UTC(3);
HH   = UTC(4);
MM   = UTC(5);
SS   = UTC(6);
UT = [HH,MM,SS];

LST = local_sidereal_time_justin(yyyy,mm,dd,UT,lon); % lon in degrees

%rhoVectECI = ((rotz(-LST)')*((roty(rad2deg(-((pi/2)-
latitude))))')*rhoVectSEZ);
rhoVectECI = rz(-LST,'deg')*ry((-(90 - lat)),'deg')*rhosez'; % LST and lat
in degrees; rho in km

end % function
```

Published with MATLAB® R2023b

```
function aer = eci2aer(POSITION,UTC,LLA0,varargin)

%   ECI2AER Convert Earth-centered Inertial (ECI) to local azimuth,
%   elevation, and slant range coordinates.
%   AER =
ECI2AER( POSITION,UTC,LLA0,REDUCTION,DELTAAT,DELTAUT1,POLARMOTION,'ADDPARAMNA
ME',ADDPARAMVALUE )
%   converts a set of ECI Cartesian coordinates, POSITION, to local azimuth,
%   elevation and slant range coordinates.
%
%   Inputs arguments for ECI2AER are:
%   POSITION:       M-by-3 array of ECI coordinates in meters.
%   UTC:            Array of Universal Coordinated Time (UTC) in year,
%                   month, day, hour, minutes, and seconds for which the
%                   function calculates the coordinate conversion. Define
%                   the array as one of the following: Array with 1 row and
%                   6 columns, or M-by-6 array for M transformation
%                   matrices, one for each UTC date. Values for year,
%                   month, day, hour, and minutes should be whole numbers.
%   LLA0:           M-by-3 array with the geodetic coordinates of the local
%                   reference with latitude, longitude and ellipsoidal
%                   altitude in degrees, degrees and meters respectively.
%   REDUCTION:      String indicating the reduction process through which
%                   the function calculates the coordinate conversion.
%                   It can either be IAU-76/FK5 (which uses the IAU 1976
%                   Precession Model and the IAU 1980 Theory of Nutation,
%                   which is no longer current but some programs still use
%                   this reduction) or IAU-2000/2006 (which uses the P03
%                   precession model). The reduction method you select
%                   determines the ADDPARAMNAME parameter pair
%                   characteristics. The IAU-76/FK5 method returns a
%                   coordinate conversion that is not orthogonal due to the
%                   polar motion approximation. The default value is
%                   IAU-2000/2006.
%   DELTAAT:        Difference, in seconds, between the International Atomic
%                   Time (TAI) and UTC. It can be defined as either a
%                   scalar or a one-dimensional array with M elements (if M
%                   UTC dates are defined), for equal number ECI
%                   coordinates. The default value is an M-by-1 null array.
%   DELTAUT1:       Difference, in seconds, between UTC and Universal Time
%                   (UT1). It can be defined as either a scalar or a one-
%                   dimensional array with M elements (if M UTC dates
%                   defined) for equal number ECI coordinates. The default
%                   value is an M-by-1 null array.
%   POLARMOTION:    Polar displacement due to the motion of the Earth's
%                   crust, in radians, along the x- and y-axis. It can be
%                   defined as either a 1-by-2 array or an M-by-2 (if M UTC
%                   dates defined) for equal number of ECI coordinates. The
%                   default value is an M-by-2 null array.
%
%   Input parameter Name/Value pairs for 'ADDPARAMNAME' and ADDPARAMVALUE
%   are:
%   'DNUTATION':    (IAU-76/FK5 reduction only) M-by-2 array for the
```

```
%                   adjustment in radians to the longitude (dDeltaPsi) and
%                   obliquity (dDeltaEpsilon). The default value is an
%                   M-by-2 null array.
%   'DCIP':         (IAU-2000/2006 reduction only) M by 2 array for the
%                   adjustment in radians to the location of the Celestial
%                   Intermediate Pole (CIP) along the x (dDeltaX) and y
%                   (dDeltaY) axis. The default value is an M-by-2 null
%                   array.
%   'FLATTENING':   Earth's flattening. The default value is the one
%                   defined by WGS84 (1/298.257223563).
%   'RE':           Earth's equatorial radius. The default value is the one
%                   defined by WGS84 (6378137 meters).
%   'PSI0':         Angular direction of the local reference system (degrees
%                   clockwise from north). The default value is an M-by-1
%                   null array.
%
%   For historical values for DNUTATION and DCIP, see the International
%   Earth Rotation and Reference Systems Service (IERS) website
%   (http://www.iers.org) under the 'Earth Orientation Data' product.
%
%   Output calculated by ECI2AER is:
%   AER:    M-by-3 array with the local reference coordinates azimuth,
%           elevation, and slant range in degrees, degrees, and meters,
%           respectively. Azimuth is defined as the angle measured
%           clockwise from true north and varies between 0 and 360 deg.
%           Elevation is defined as the angle between a plane perpendicular
%           to the ellipsoid's surface at LL0 and the line that goes from
%           the local reference to the object's position and varies between
%           -90 and 90 degrees. Slant range is defined as the straight line
%           distance between the local reference and the object.
%
%   This method has higher accuracy over small distances from the local
%   geodetic reference frame (LLA0) in latitude and longitude, and nearer to
%   the equator.
%
%   Example:
%   Estimate the position of the azimuth, elevation and range for an object
%   with Earth Centered Inertial position 1e08*[-3.8454 -0.5099 -0.3255]
%   meters for the date 1969/7/20 21:17:40 UTC at 28.4 deg North, 80.5 deg
%   West and 2.7 meters altitude.
%
%   AER = eci2aer(1e08*[-3.8454,-0.5099,-0.3255],[1969,7,20,21,17,40], ...
%                 [28.4,-80.5,2.7])
%
%   See also LLA2ECI, ECI2LLA, DCMECI2ECEF, FLAT2LLA, LLA2FLAT, DELTAUT1,
%   DELTACIP, POLARMOTION.
```

Validate i/o

```
% Validate outputs
nargoutchk(0,1)

% Validate date
```

```
validateattributes(UTC,{'numeric'},{'ncols',6,'real','finite','nonnan'})

% Validate latitude longitude altitude
validateattributes(POSITION,{'numeric'},{'ncols',3,'real','finite','nonnan'})

% Assign date vectors
year = UTC(:,1);
month = UTC(:,2);
day = UTC(:,3);
hour = UTC(:,4);
min = UTC(:,5);
sec = UTC(:,6);

% Validate vectors
if any(year<1)
    error(message('aero:dcmeci2ecef:invalidYear'));
end
if any(month<1) || any(month>12)
    error(message('aero:dcmeci2ecef:invalidMonth'));
end
if any(day<1) || any(day>31)
    error(message('aero:dcmeci2ecef:invalidDay'));
end
if any(hour<0) || any(hour>24)
    error(message('aero:dcmeci2ecef:invalidHour'));
end
if any(min<0) || any(min>60)
    error(message('aero:dcmeci2ecef:invalidMin'));
end
if any(sec<0) || any(sec>60)
    error(message('aero:dcmeci2ecef:invalidSec'));
end
len = length(year);

% Parse and validate the inputs
ob = inputParser;
validReduction = {'IAU-2000/2006','IAU-76/FK5'};
addRequired(ob,'POSITION',@(x) validateattributes(x,{'numeric'},
{'ncols',3,'real',...
    'finite','nonnan','size',[len,3]}));
addRequired(ob,'UTC',@(x) validateattributes(x,{'numeric'},
{'ncols',6,'real',...
    'finite','nonnan'}));
addRequired(ob,'LLA0',@(x) validateattributes(x,{'numeric'},
{'ncols',3,'real',...
    'finite','nonnan','size',[len,3]}));
addOptional(ob,'reduction','iau-2000/2006',@(x)
validateReduction(x,validReduction));
addOptional(ob,'deltaAT',zeros(len,1),@(x) validateattributes(x,
{'numeric'},...
    {'real','finite','nonnan','size',[len,1]}));
addOptional(ob,'deltaUT1',zeros(len,1),@(x) validateattributes(x,
{'numeric'},...
    {'real','finite','nonnan','size',[len,1]}));
```

```
addOptional(ob,'polarMotion',zeros(len,2),@(x) validateattributes(x,
{'numeric'},...
    {'real','finite','nonnan','size',[len,2]}));
addParameter(ob,'dNutation',zeros(len,2),@(x) validateattributes(x,
{'numeric'},...
    {'real','finite','nonnan','size',[len,2]}));
addParameter(ob,'dCIP',zeros(len,2),@(x) validateattributes(x,{'numeric'},...
    {'real','finite','nonnan','size',[len,2]}));
addParameter(ob,'flattening',1/298.257223563,@(x) validateattributes(x,
{'numeric'},...
    {'real','finite','nonnan','size',[1 1]}));
addParameter(ob,'RE',6378137,@(x) validateattributes(x,{'numeric'},...
    {'real','finite','nonnan','size',[1 1]}));
addOptional(ob,'PSI0',zeros(len,1),@(x) validateattributes(x,{'numeric'},...
    {'real','finite','nonnan','size',[len,1]}));

% Parse input object
parse(ob,POSITION,UTC,LLA0,varargin{:});

% Validate reduction
reduction = ob.Results.reduction;
reduction = lower(validatestring(reduction,validReduction));

%Validate that the additional parameter matches the reduction method
if any(strcmp(ob.UsingDefaults,'dNutation')) &&
~any(strcmp(ob.UsingDefaults,'dCIP')) &&...
        ~strcmp(reduction,'iau-2000/2006')
    error(message('aero:dcmeci2ecef:invalidDNutation'));
elseif ~any(strcmp(ob.UsingDefaults,'dNutation')) &&
any(strcmp(ob.UsingDefaults,'dCIP')) && ...
        ~strcmp(reduction,'iau-76/fk5')
    error(message('aero:dcmeci2ecef:invalidDCIP'));
end

%Validate that both reduction methods are not defined (just one should be
%defined)
if ~any(strcmp(ob.UsingDefaults,'dNutation')) &&
~any(strcmp(ob.UsingDefaults,'dCIP'))
    error(message('aero:dcmeci2ecef:invalidDNutationDCIP'))
end
```

Calculate DCM ECI to ECEF

```
switch reduction
    case 'iau-76/fk5'
        dcm =
dcmeci2ecef(ob.Results.reduction,ob.Results.UTC,ob.Results.deltaAT,...

ob.Results.deltaUT1,ob.Results.polarMotion,'dNutation',ob.Results.dNutation);
    case 'iau-2000/2006'
        dcm =
dcmeci2ecef(ob.Results.reduction,ob.Results.UTC,ob.Results.deltaAT,...
```

```
ob.Results.deltaUT1,ob.Results.polarMotion,'dCIP',ob.Results.dCIP);
end
```

Calculate position in ECEF coordinates

```
tmp = arrayfun(@(k) (dcm(:,:,k)*POSITION(k,:)'),1:len,'UniformOutput',false);
ecefObject = cell2mat(tmp)';
```

Calculate position in NED coordinates

Wrap latitude and longitude for the local geodetic reference system

```
LLA0 = ob.Results.LLA0;
[~, LLA0(:,1), LLA0(:,2)] = wraplatitude( ob.Results.LLA0(:,1),
ob.Results.LLA0(:,2),180);
[~, LLA0(:,2)] = wraplongitude( LLA0(:,2), 180 );
% Use local function for NED transform
tmp2 = arrayfun(@(k)
(nedCalc(ecefObject(k,:),LLA0(k,:),ob.Results.flattening,ob.Results.RE)),1:le
n,'UniformOutput',false);
nedObject = cell2mat(tmp2');
```

Calculate Azimuth, Elevation, Range

```
hypotxy = hypot(nedObject(:,1),nedObject(:,2));
r = hypot(hypotxy,-nedObject(:,3));
elev = atan2d(-nedObject(:,3),hypotxy);
az = atan2d(nedObject(:,2),nedObject(:,1))-ob.Results.PSI0;
az = mod(az,360);
aer = [az elev r];

end

function validateReduction(reduction,validReduction)
validatestring(reduction,validReduction);
end

function ned = nedCalc(posEcef,LLA0,f,Re)
% This helper function calculates the NED coordinates for the object given
% in ECEF coordinates given a local geodetic position LLA0. It requires the
% flattening and equatorial radius. It calculates the ENU coordinates and
% finally rotates it for NED.

refPosEcef = lla2ecef(LLA0,f,Re);
dPos = posEcef - refPosEcef;

% ENU position
enu = [ -sind(LLA0(2))                  cosd(LLA0(2))                    0;...
        -cosd(LLA0(2))*sind(LLA0(1))    -sind(LLA0(1))*sind(LLA0(2))
cosd(LLA0(1)); ...
        cosd(LLA0(1))*cosd(LLA0(2))     cosd(LLA0(1))*sind(LLA0(2))
sind(LLA0(1))]*...
```

```
        dPos';

% NED position
ned = [enu(2) enu(1) -enu(3)];

end
```


Published with MATLAB® R2023b

```
function [Xnom,P,xhat,jd,HtWH,HtWy] =
weightedLeastSquares_HW2_2(data,lat,lon,alt)

%{
This function takes raw observational data as an input and performs finite
differencing (at hard coded values) to determine best state vector for an
orbit. THIS IS SPECIFICALLY PARAMETERIZED FOR AERO557, Winter 2024, HW 2,
Problem 2.
Author: Justin Self
2/17/2024 (Denver, CO)

INPUTS:
    dataset of observations (VERY SPECIFIC TO THE HW PROBLEM
    lat (degrees)
    lon (degrees)
    altitude (meters)
%}

% HW 2.2: Weighted Least Squares
mu = 398600; % km3/s2

% Extract inputs
year = data(:,1);
m = data(:,2);
d = data(:,3);
UT = data(:,4:6);
az = data(:,7);                  % deg
el = data(:,8);                  % deg
rho = data(:,9);  % range        % km
lat = deg2rad(lat);          % degrees to rad
lon = deg2rad(lon);        % degrees to rad
rhoErr = 92.5/1000;              % km
azErr = deg2rad(0.0224);         % rad
elErr = deg2rad(0.0139);         % rad

UTC = [year,m,d,UT];
UTC0 = UTC;
bias = [80e-3,0.0081,0.0045]; % km, deg [rho, az, el]

%..... W
% Build and normalize weight matrix (km; RADIANS)
W = [1/rhoErr^2   0           0;
     0              1/(azErr)^2   0;
     0              0           1/(elErr)^2];


% Normalize Weight matrix
Wnormalized = W * norm(W)^-1;
```

Find Rsite (IJK)

```
rECI = zeros(3,10);
jd = zeros(10,1);

for i = 1:10
    [RsiteECI, ~] = Rsite(rad2deg(lat),rad2deg(lon),alt,UTC(i,:));
    rhoECI =
rhoVectECI(az(i),el(i),rho(i),rad2deg(lat),rad2deg(lon),UTC(i,:));
    rECI(:,i) = RsiteECI + rhoECI;
    jd(i) = juliandate(datetime(UTC(i,:)));
end

%.... Group A
    r1 = rECI(:,1);
    [t1, ~, ~] = juliandateFunction(year(1),m(1),d(1),UT(1,:));
    r4 = rECI(:,4);
    [t4, ~, ~] = juliandateFunction(year(4),m(4),d(4),UT(4,:));
    r7 = rECI(:,7);
    [t7, ~, ~] = juliandateFunction(year(7),m(7),d(7),UT(7,:));

    % Velocity
    [v4, ~,~,~, ~ ] = hgibbs(r1,r4,r7,t1,t4,t7);

    % r4, v4 to v7
    dt = (t7 - t4)*86400;
    [~,v7] = universalVar(mu,dt,r4,v4);

    % Propagate r4 back to r1
        % Time from t1 - t4 (negative)
        dt1 = (t1 - t4)*86400; % days to seconds
        [r1_4,v1_4] = universalVar(mu,dt1,r4,v4);
        v1 = v1_4; % redefine for simplicity later.

%.... Group B
    r2 = rECI(:,2);
    [t2, ~, ~] = juliandateFunction(year(2),m(2),d(2),UT(2,:));
    r5 = rECI(:,5);
    [t5, ~, ~] = juliandateFunction(year(5),m(5),d(5),UT(5,:));
    r8 = rECI(:,8);
    [t8, ~, ~] = juliandateFunction(year(8),m(8),d(8),UT(8,:));
    % Velocity
    [v5, ~,~,~, ~ ] = hgibbs(r2,r5,r8,t2,t5,t8);
    % r5, v5 to v2
    dt = (t2 - t5)*86400; % backward from 5 to 2
    [~,v2] = universalVar(mu,dt,r5,v5);

    % r5, v5 to v8
    dt = (t8 - t5)*86400;
    [~,v8] = universalVar(mu,dt,r5,v5);

    % Propagate r5 back to r1
        % Time from t1 - t5 (negative)
```

```
        dt5 = (t1 - t5)*86400; % days to seconds
        [r1_5,v1_5] = universalVar(mu,dt5,r5,v5);

% Group C
    r3 = rECI(:,3);
    [t3, ~, ~] = juliandateFunction(year(3),m(3),d(3),UT(3,:));
    r6 = rECI(:,6);
    [t6, ~, ~] = juliandateFunction(year(6),m(6),d(6),UT(6,:));
    r9 = rECI(:,9);
    [t9, ~, ~] = juliandateFunction(year(9),m(9),d(9),UT(9,:));
    % Velocity
    [v6, ~,~,~, ~ ] = hgibbs(r3,r6,r9,t3,t6,t9);
    % r6, v6 to v3
    dt = (t3 - t6)*86400; % back from 6 to 3
    [~,v3] = universalVar(mu,dt,r6,v6);
    % r6, v6 to v9
    dt = (t9 - t6)*86400;
    [~,v9] = universalVar(mu,dt,r6,v6);

        % Propagate r6 back to r1
        % Time from t1 - t6 (negative)
        dt6 = (t1 - t6)*86400; % days to seconds
        [r1_6,v1_6] = universalVar(mu,dt6,r6,v6);

%.... If all looks good, average each all values to determind X_nom.
Rnom = (r1 + r1_4 + r1_5 + r1_6) / 4;
Vnom = (v1_4 + v1_5 + v1_6) / 3;

format short g
Xnom = [Rnom; Vnom]; % based on time 1.

% Validate against Vallado
RnomVallado = [5975.2904; 2568.6400; 3120.5845];
VnomVallado = [3.983846; -2.071159; -5.917095];

% ** For testing use Vallado's Xnom **
%Xnom = [RnomVallado;VnomVallado];
```

Preallocate for loop

```
maxiteration = 10; % expect <10

% Get into loop and preallocate
RMSprev = 2;
RMScurrent = 1;
criteria = abs((RMSprev - RMScurrent)/RMSprev);
ii = 1; % iteration counter
lla0 = [rad2deg(lat),rad2deg(lon),alt];
HtWH_accumulated = zeros(6,6);
HtWy_accumulated = zeros(6,1);
RMSterm_accumulated = 0;
```

Differential correction loop

```
while criteria > 1e-3 && (ii < 10)

    for i = 1:length(data)
        % reset H and y matrices
        H = zeros(6,3);
        y = zeros(1,3);

        % Propagate nominal state foward to time (i)
        dt = 86400*(jd(i) - jd(1));
        [rCalc,vCalc] = universalVar(mu,dt,Xnom(1:3),Xnom(4:6));

        % Calculate observations (rho, az, el)
        UTC = [data(i,1);data(i,2);data(i,3);data(i,4);data(i,5);data(i,6)];
        aer = eci2aer(1000*rCalc',UTC',lla0);
        Az = deg2rad(aer(1));
        El = deg2rad(aer(2));
        Rho = aer(3)/1000;
        calc = [Rho;Az;El];
        obs = [data(i,9);deg2rad(data(i,7));deg2rad(data(i,8))];
        % Take difference
        y = (obs - calc);
        state = [rCalc;vCalc];

        % Finite differencing
        for j = 1:length(state)
            % Create modified state, perturbed elementwise
            del = Xnom(j)*0.001;
            Xmod = Xnom;
            Xmod(j) = Xnom(j) + del;

            % Propagate both states forward to obs time
            dt = 86400*(jd(i) - jd(1));
            [r_mod,~] = universalVar(mu,dt,Xmod(1:3),Xmod(4:6));
            % Propagate Xnom
            [r_nom,~] = universalVar(mu,dt,Xnom(1:3),Xnom(4:6));

            % Calculate observations, perturbed state
            aer = eci2aer(1000*r_mod',UTC0(i,:),lla0);
            Az = deg2rad(aer(1));
            El = deg2rad(aer(2));
            Rho = aer(3)/1000;
            mod = [Rho;Az;El];

            % Calculate observations, nominal state
            aer = eci2aer(1000*r_nom',UTC0(i,:),lla0);
            Az = deg2rad(aer(1));
            El = deg2rad(aer(2));
            Rho = aer(3)/1000;
            nom = [Rho;Az;El];

            % Build H matrix
```

```
            H(j,1:3) = (mod - nom) / del;

        end
        H = H';

        % Compute matrices
        HtWH = H'*W*H; % 6x6
        HtWy = H'*W*y; % 6x1

        % Store
        HtWH_accumulated = HtWH + HtWH_accumulated;
        HtWy_accumulated = HtWy + HtWy_accumulated;
        RMSterm_accumulated = y'*Wnormalized*y + RMSterm_accumulated;

    end

    % Find P
    P = HtWH_accumulated^-1;

    % Calc xhat
    xhat = P*HtWy_accumulated;

    % Update nominal state
    Xnom = Xnom + xhat;

    % Find error
    obs = 9; % number of observations (9)
    N = 3; % number of observation elements (rho, az, el)
    RMSprev = RMScurrent;
    RMScurrent = sqrt(RMSterm_accumulated / (obs*N));
    criteria = abs((RMSprev - RMScurrent) / RMSprev);
    err(ii) = criteria;

    % Reset storage matrices
    HtWH_accumulated = zeros(6,6);
    HtWy_accumulated = zeros(6,1);
    RMSterm_accumulated = 0;

    % update count
    ii = ii + 1;

end % (ii) big while loop
```

Published with MATLAB® R2023b