ASEN 6080 HW 4

– Ian Gober, 108577813

1. a. Make sure you have a working CKF with SNC.

It works! See PDF

b. Implement the sequential filter smoothing algorithm, including the covariance matrix.

See PDF!

c. Compare processing a set of data without process noise to processing with a batch after 1 iteration.

See PDF!

My smoothed LKF results without process noise ~~look~~ look almost identical to a Batch filter after one iteration.

This makes sense, as the smoothed covariance without process noise is essentially just the final covariance mapped back in time, while the batch maps the estimated initial covariance forward in time. Since theoretically $P_{LKF,f} = P_{Batch,0}$, the results will be identical.

However, the smoother exhibits some numerical issues. In particular, the reported covariance matrices become non-positive definite near the start of the orbit. This is likely due to saturation of the LKF, which results in a covariance estimate that's too smug to reflect reality. This issue is avoided when we introduce SNC, and hence some semblance of a noise floor to avoid saturation.

d. Take the results of the CKF with SNC using the optimal σ from HW 3 and smooth them. Generate plots of the inertial state errors over time with covariance envelope + provide the RMS state errors.

I used a σ of 1e-8 km/s^2.

See PDF for plots

Component-wise smoothed RMS:

$$[7.015\times10^{-3},\ 4.555\times10^{-3},\ 8.598\times10^{-3},\ 6.668\times10^{-6},\ 3.434\times10^{-6},\ 4.274\times10^{-6}]^T$$

3D smoothed RMS: 1.199×10^{-2}

e. Compare the RMS values from part d to those from HW3 and discuss.

Component-wise SNC RMS:

$$\left[1.722\times10^{-2},\ 1.550\times10^{-2},\ 2.282\times10^{-2},\ 2.684\times10^{-5},\ 1.492\times10^{-5},\ 2.152\times10^{-5}\right]^T$$

3D SNC RMS: 3.232×10^{-2}

The smoothed state error RMS is much less than with just SNC, with most components and the 3D RMS being more than halved.

This makes sense, as the point of a smoother is to generate a better state estimate at each measurement time, i.e. reduce the state error. Further, smoothing removes the initial spike in LKF state uncertainty, which contributes heavily to the LKF state error. Removing that spike then intuitively reduces the state error RMS over the whole dataset.

ASEN 6080 HW 3 Problem 1 Main Script

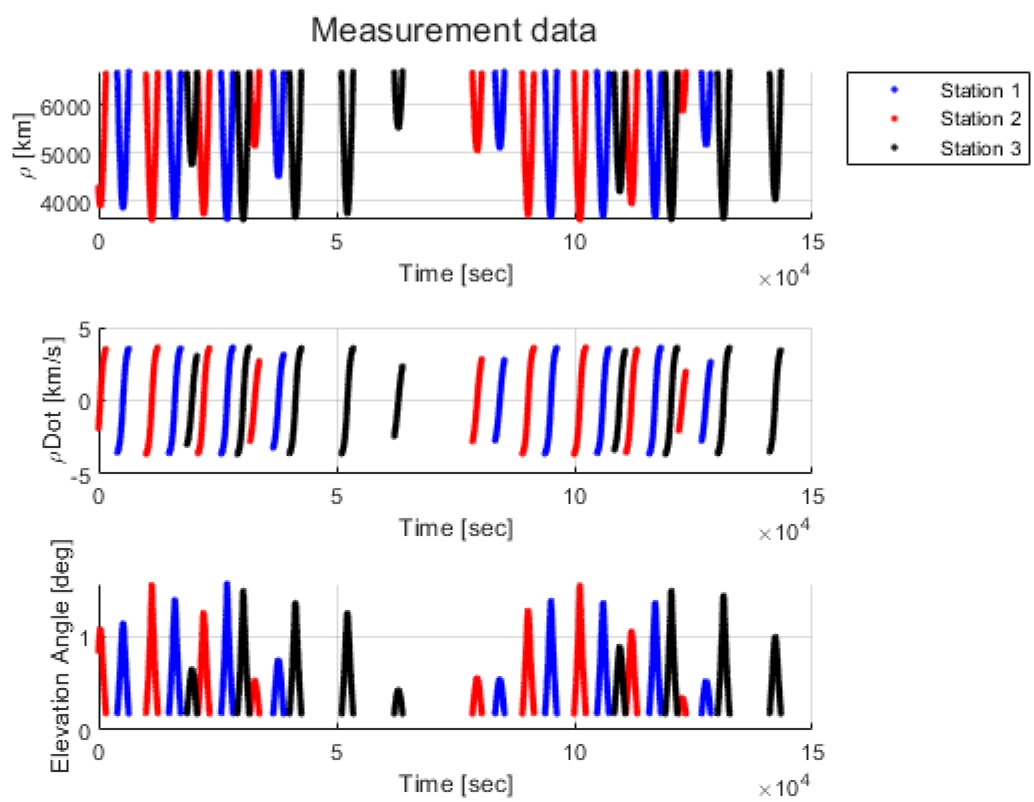
Table of Contents

By: Ian Faber

Housekeeping

Setup

Make Truth Data



Problem 1a: Filter setup

Problem 1a: Prove SNC works with optimal sigma

Based on plots, sigma = 1e-8 balances both postfit and 3D RMS

```
1a. Running LKF and Smoother with SNC for sigma = 1.000e-08 km/s^2

    Running LKF:
Prefit RMS: 242.0616, Postfit RMS: 1.0070. Hit max LKF iterations. Runs so
far: 1
Final prefit RMS: 242.0616. Hit maximum number of 1 runs
Final postfit RMS: 1.0070. Hit maximum number of 1 runs
```

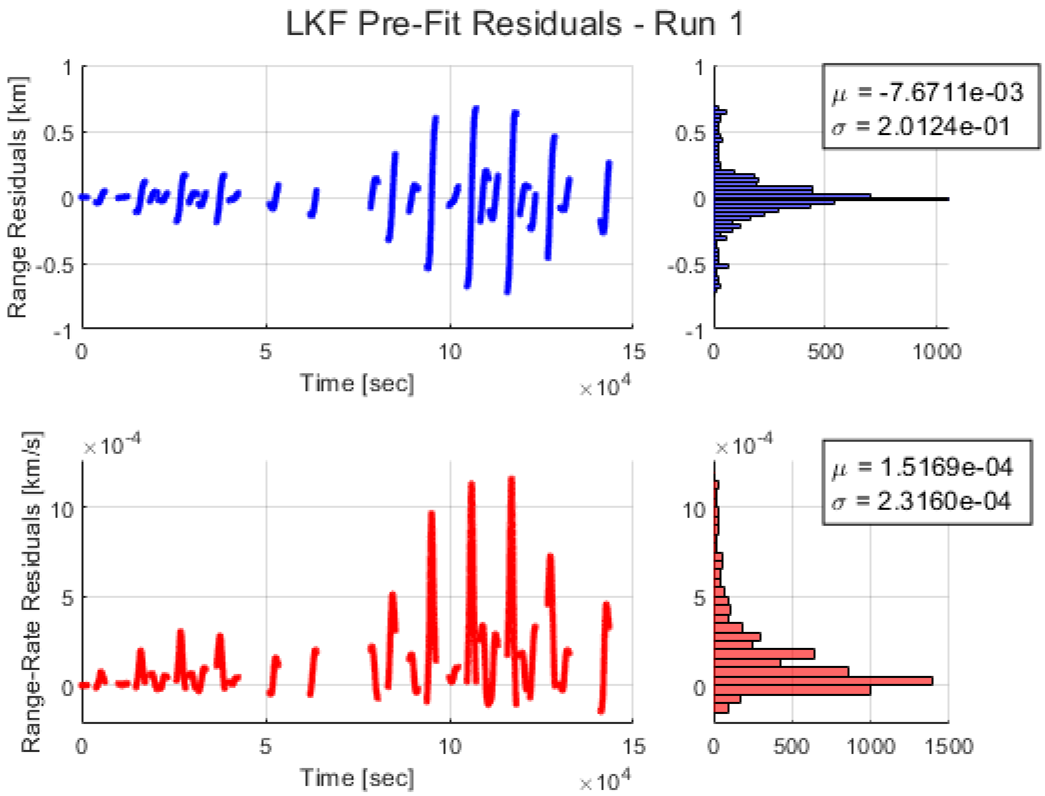
Problem 1b/d+e: Implement sequential filter smoothing algorithm and compare RMS values

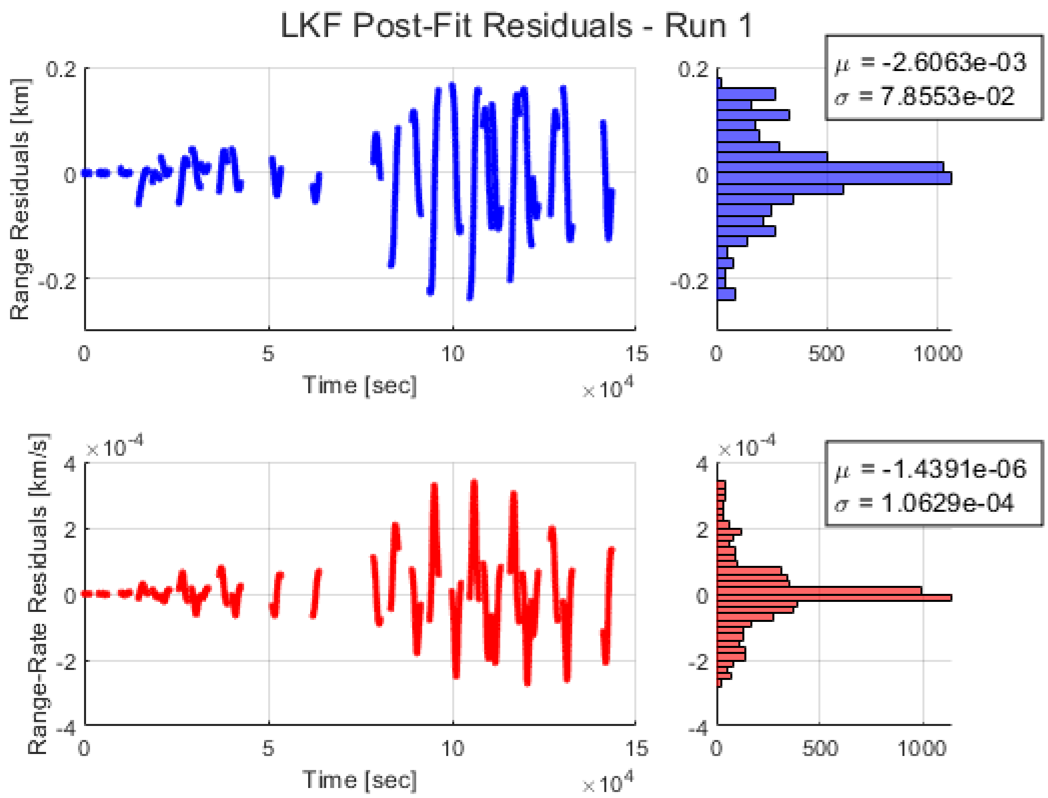
```
1b/d/e. Smoothing LKF SNC results

    Running Smoother...

    LKF w/ SNC State RMS Errors:
        Component-wise: [1.722e-02, 1.550e-02, 2.282e-02, 2.684e-05,
1.492e-05, 2.152e-05]
        3D: 3.252e-02

    Smoother w/ SNC State RMS Errors:
        Component-wise: [7.015e-03, 4.555e-03, 8.598e-03, 6.668e-06,
5.434e-06, 4.274e-06]
        3D: 1.199e-02
```

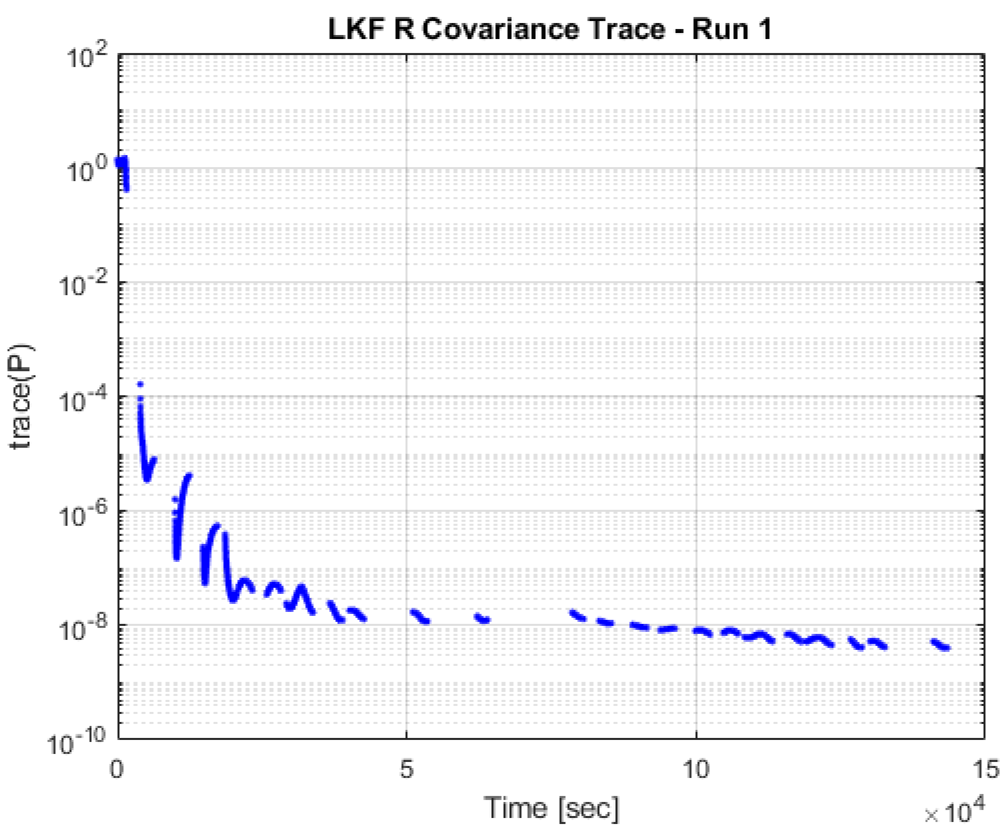
Problem 1c: Smoothing without process noise

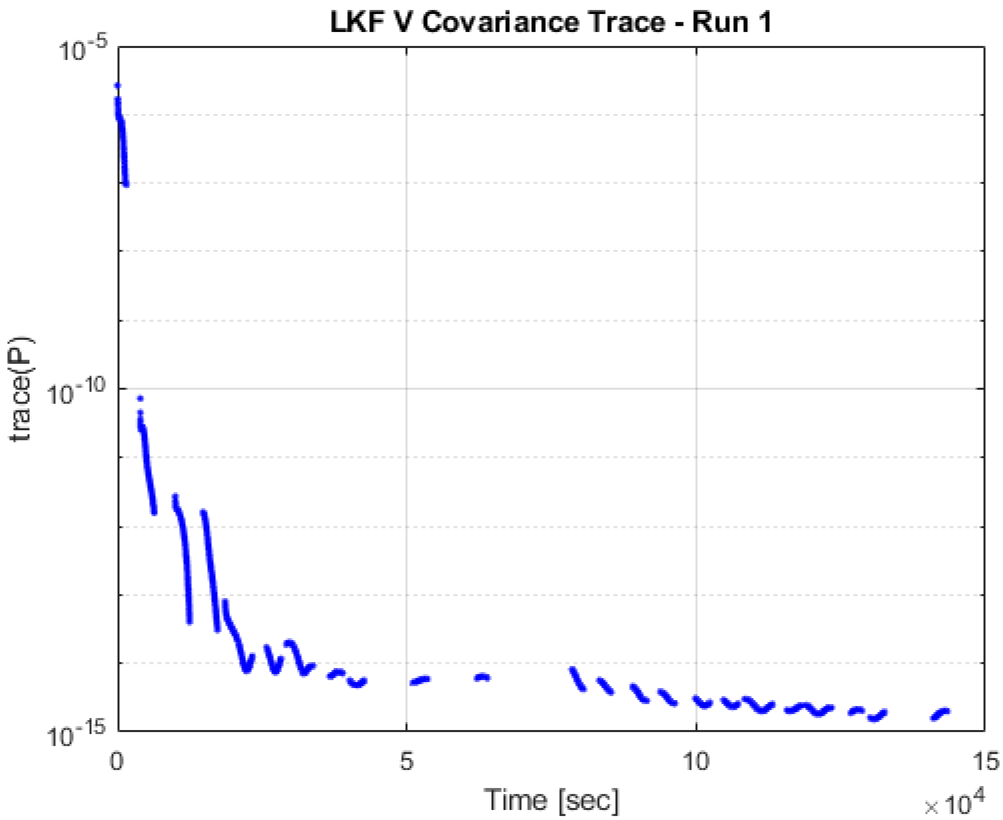
```
1c. Running LKF w/ Smoother and Batch, both without Process noise

    Running LKF:
Prefit RMS: 242.0616, Postfit RMS: 93.4748. Hit max LKF iterations. Runs so
far: 1
Final prefit RMS: 242.0616. Hit maximum number of 1 runs
Final postfit RMS: 93.4748. Hit maximum number of 1 runs

    Running Smoother...

    Running Batch Filter:
Prefit RMS: 242.0619, Postfit RMS: 93.5517. Hit max Batch iterations. Runs
```

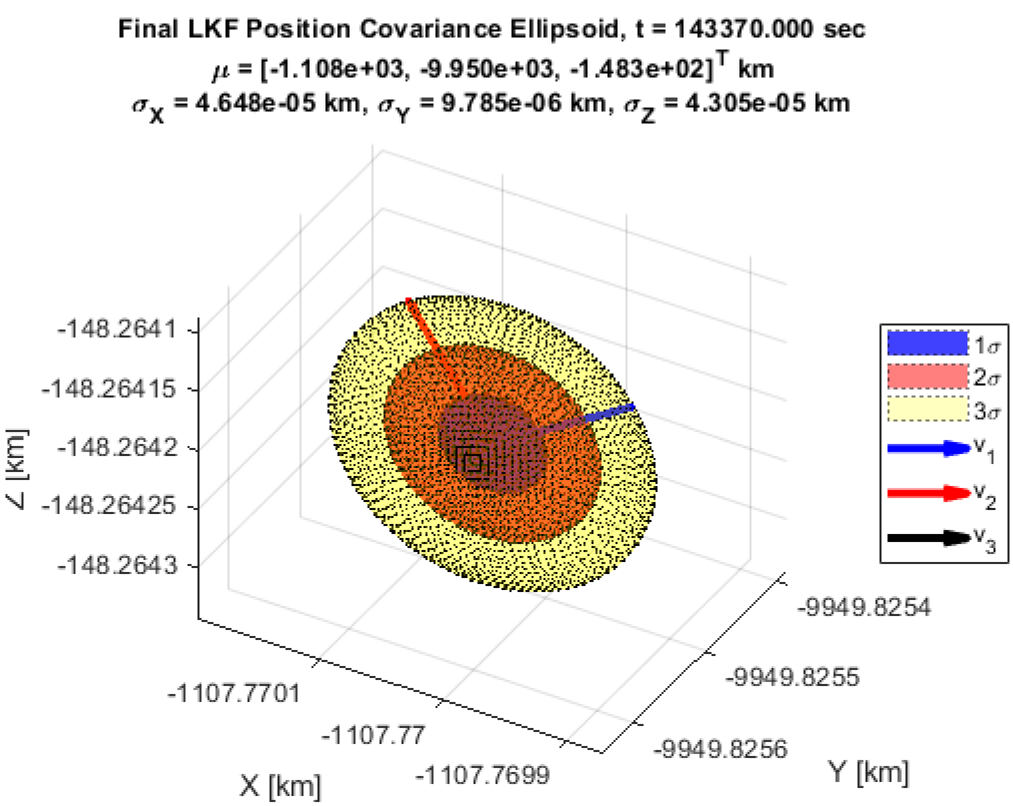
```
so far: 1
Final prefit RMS: 242.0619. Hit maximum number of 1 runs
Final postfit RMS: 93.5517. Hit maximum number of 1 runs
```

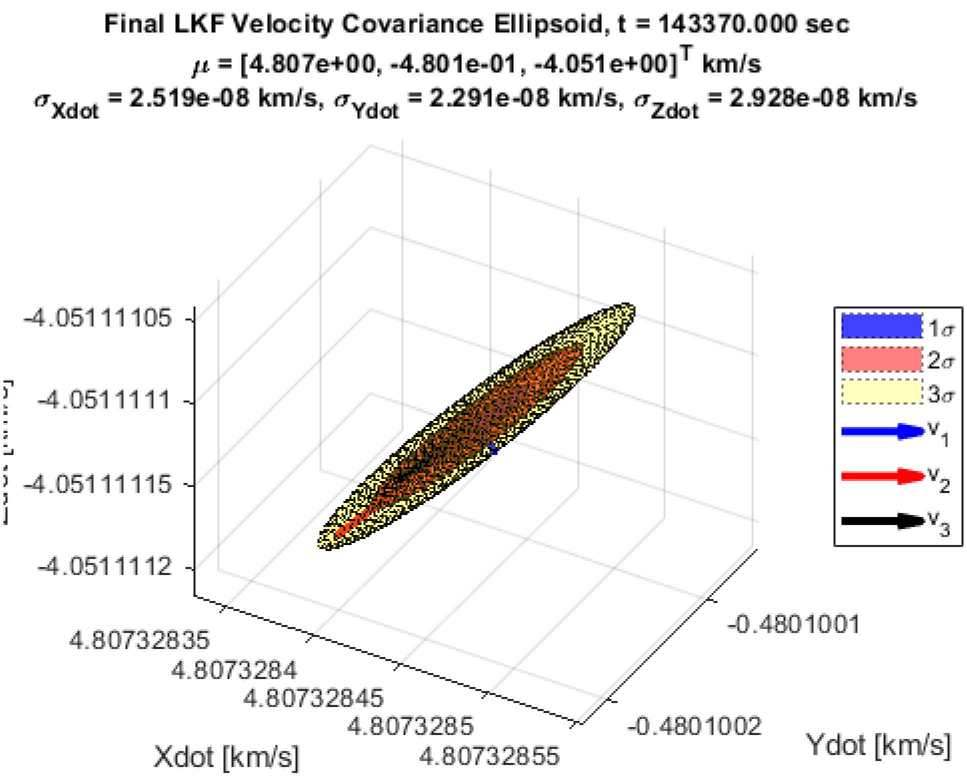


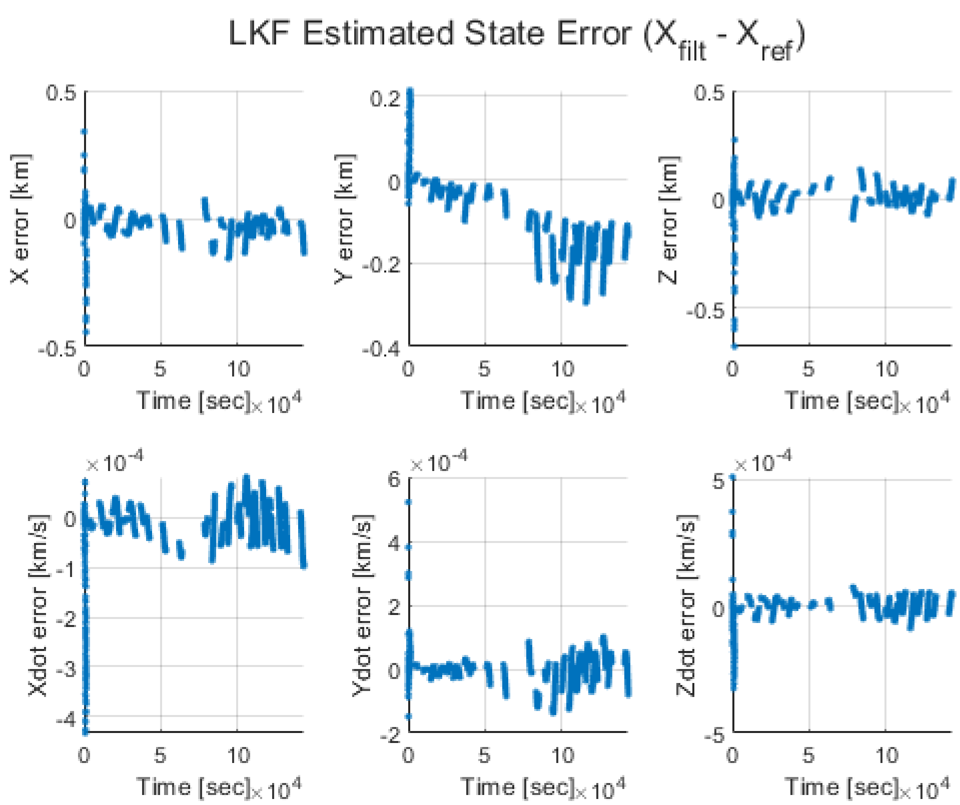

LKF Post-Fit Residuals - Run 1
Range Residuals [km]
Time [sec]
×10^4
μ = -2.6063e-03
σ = 7.8553e-02
Range-Rate Residuals [km/s]
×10^-4
μ = -1.4391e-06
σ = 1.0629e-04

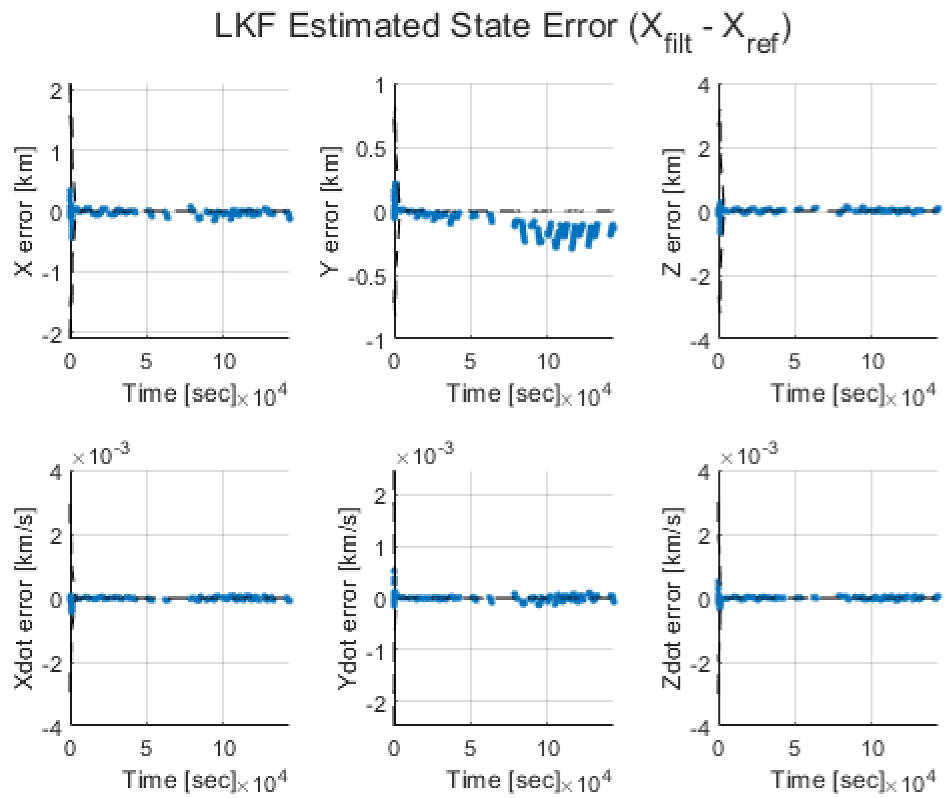

LKF R Covariance Trace - Run 1
trace(P)
Time [sec]
×10^4

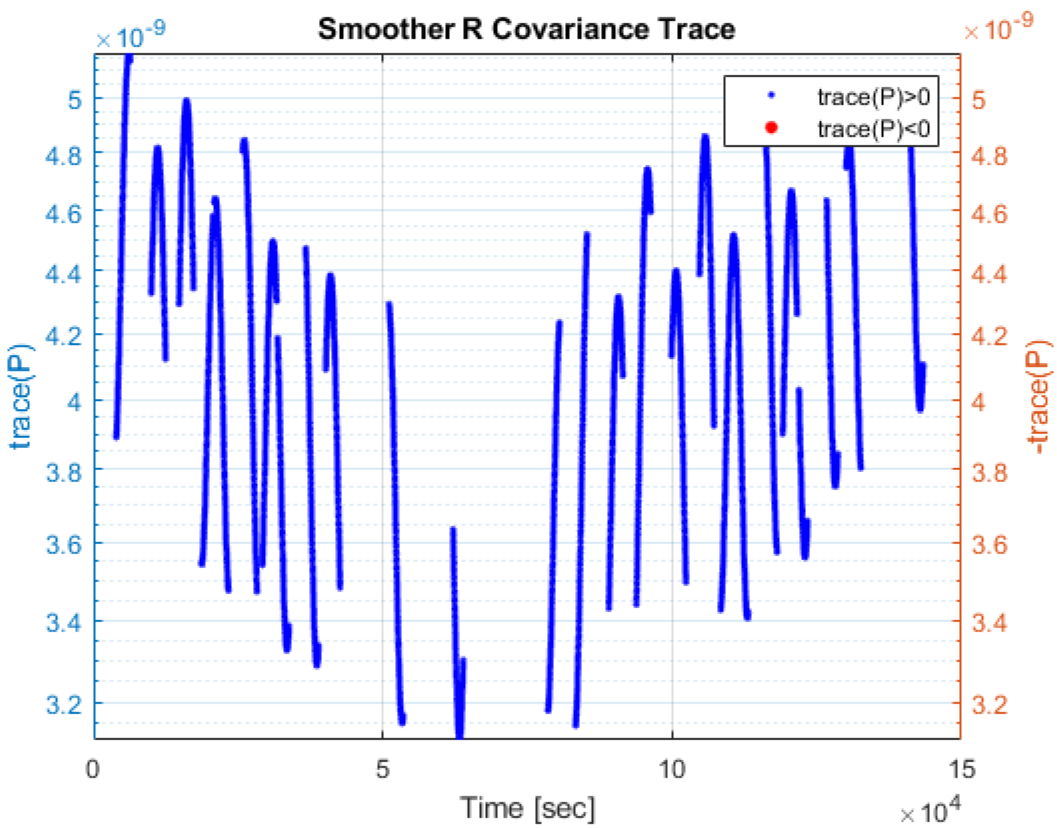

LKF V Covariance Trace - Run 1
10^-5
10^-10
10^-15
trace(P)
0
5
10
15
Time [sec]
×10^4

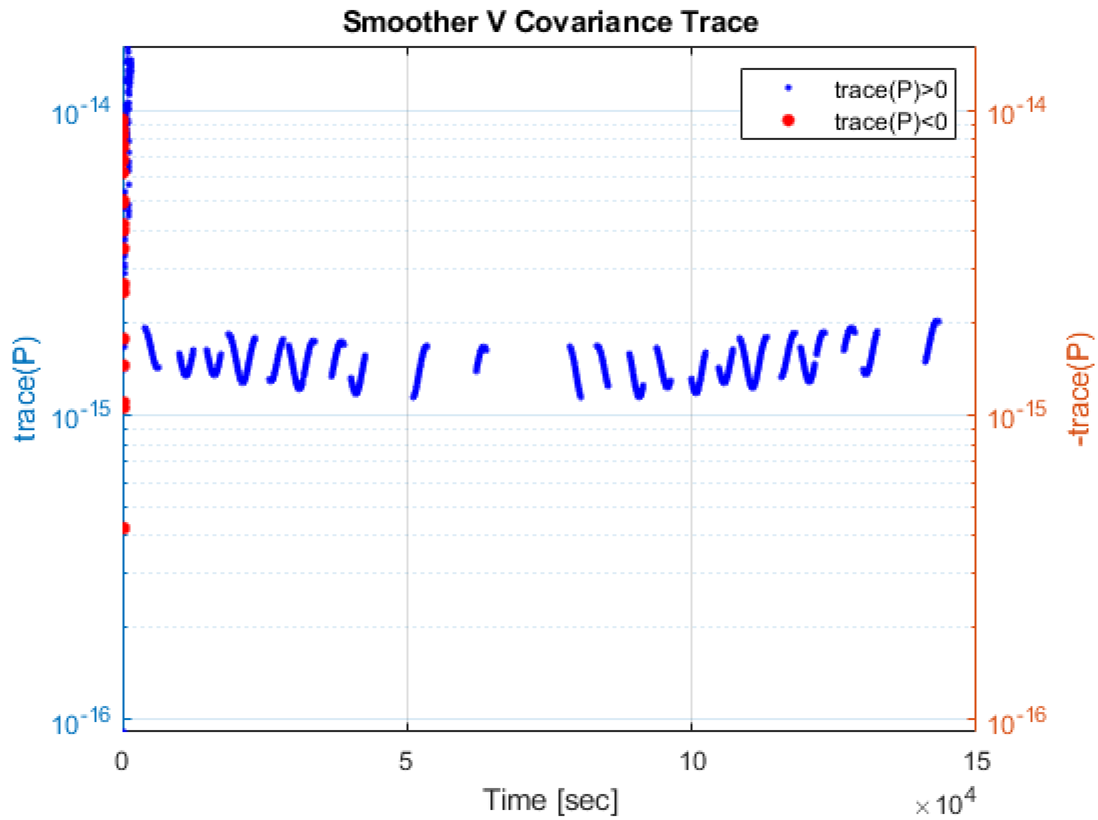

Final LKF Position Covariance Ellipsoid, t = 143370.000 sec
μ = [-1.108e+03, -9.950e+03, -1.483e+02]^T km
σX = 4.648e-05 km, σY = 9.785e-06 km, σZ = 4.305e-05 km
-148.2641
-148.26415
-148.2642
-148.26425
-148.2643
Z [km]
-1107.7701
-1107.77
-1107.7699
X [km]
-9949.8254
-9949.8255
-9949.8256
Y [km]
1σ
2σ
3σ
v1
v2
v3

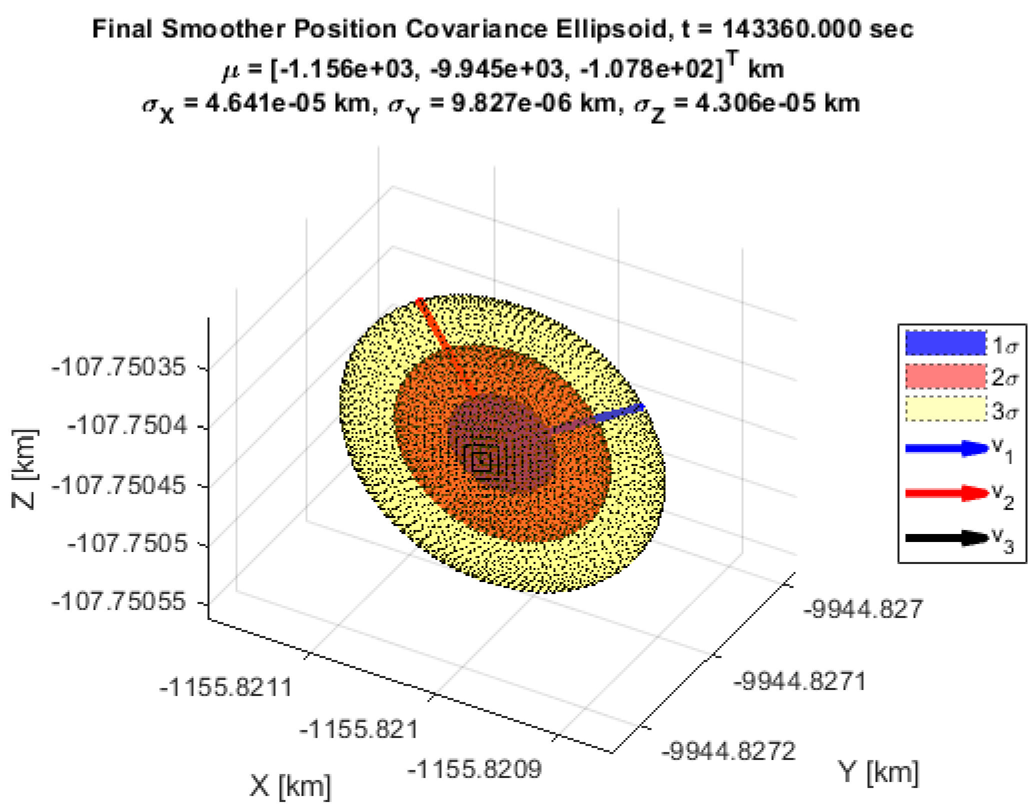

Final LKF Velocity Covariance Ellipsoid, t = 143370.000 sec
μ = [4.807e+00, -4.801e-01, -4.051e+00]^T km/s
σ_{Xdot} = 2.519e-08 km/s, σ_{Ydot} = 2.291e-08 km/s, σ_{Zdot} = 2.928e-08 km/s
-4.05111105
-4.0511111
-4.05111115
-4.0511112
4.80732835
4.8073284
4.80732845
4.8073285
4.80732855
Xdot [km/s]
-0.4801001
-0.4801002
Ydot [km/s]
1σ
2σ
3σ
v_1
v_2
v_3

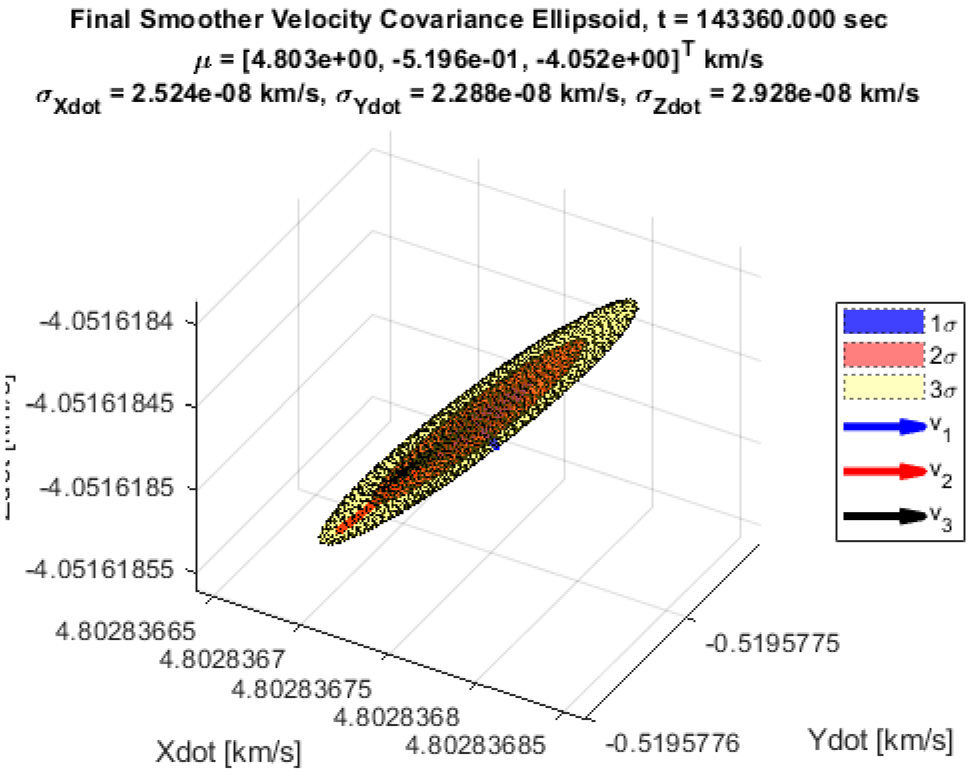

LKF Estimated State Error (X_{filt} - X_{ref})
X error [km]
Y error [km]
Z error [km]
Xdot error [km/s]
Ydot error [km/s]
Zdot error [km/s]
Time [sec] $\times 10^4$
$\times 10^{-4}$

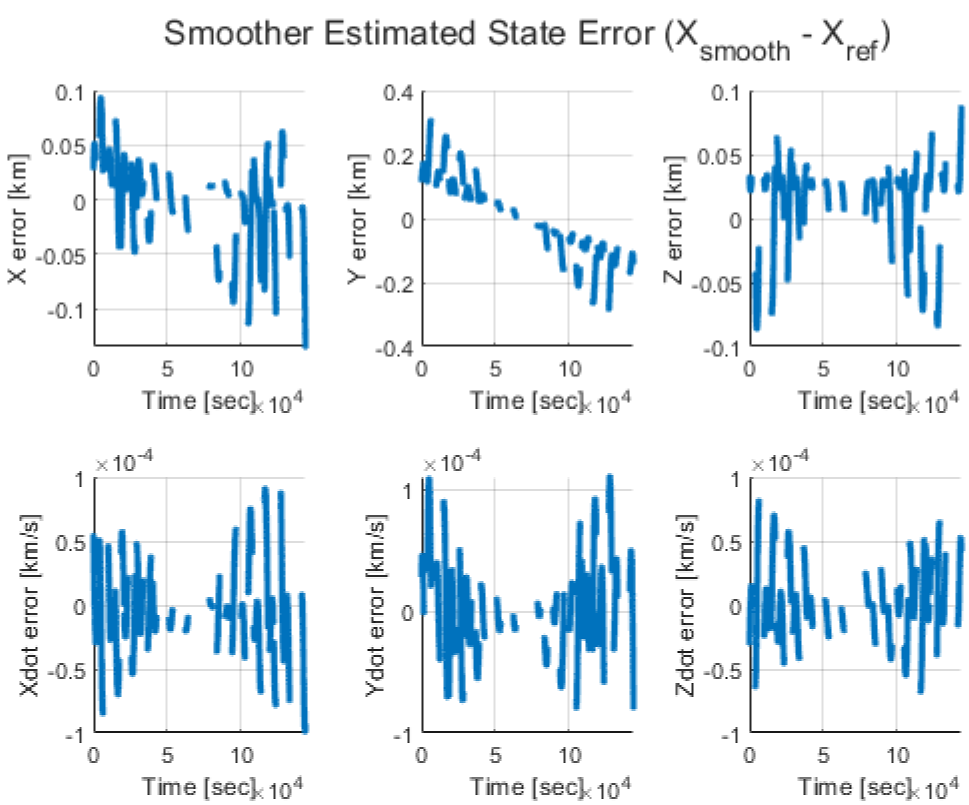

LKF Estimated State Error (X_{filt} - X_{ref})
X error [km]
Y error [km]
Z error [km]
Xdot error [km/s]
Ydot error [km/s]
Zdot error [km/s]
Time [sec]
$\times 10^4$
$\times 10^{-3}$

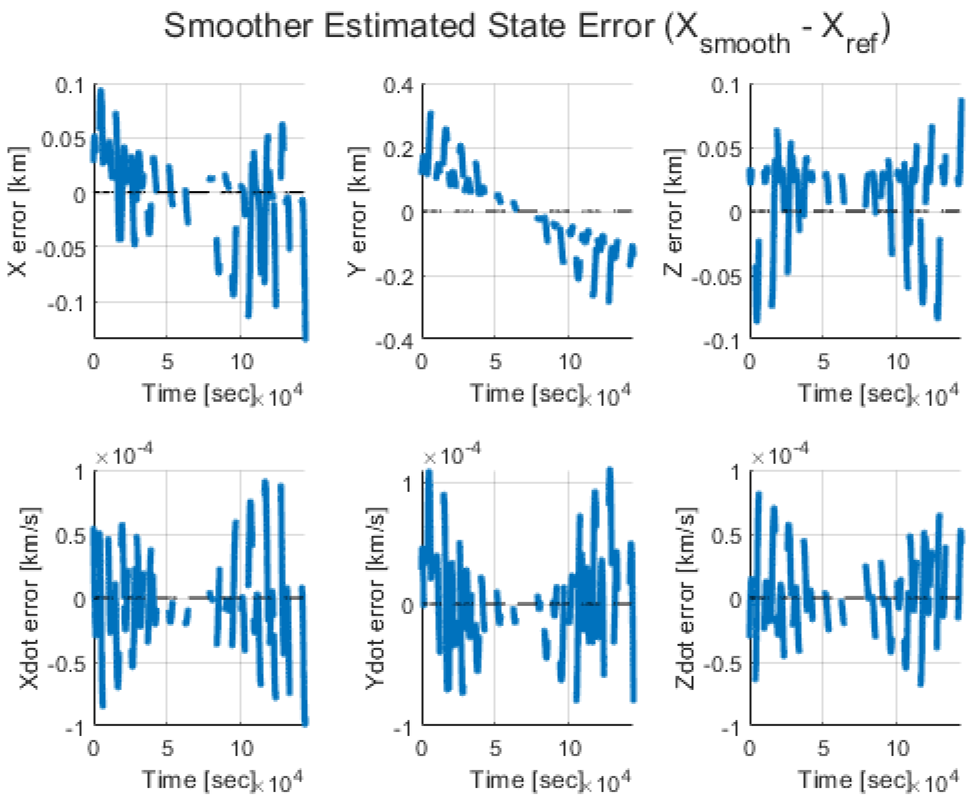

Smoother R Covariance Trace
trace(P)>0
trace(P)<0
trace(P)
-trace(P)
Time [sec]
$\times 10^{-9}$
$\times 10^4$

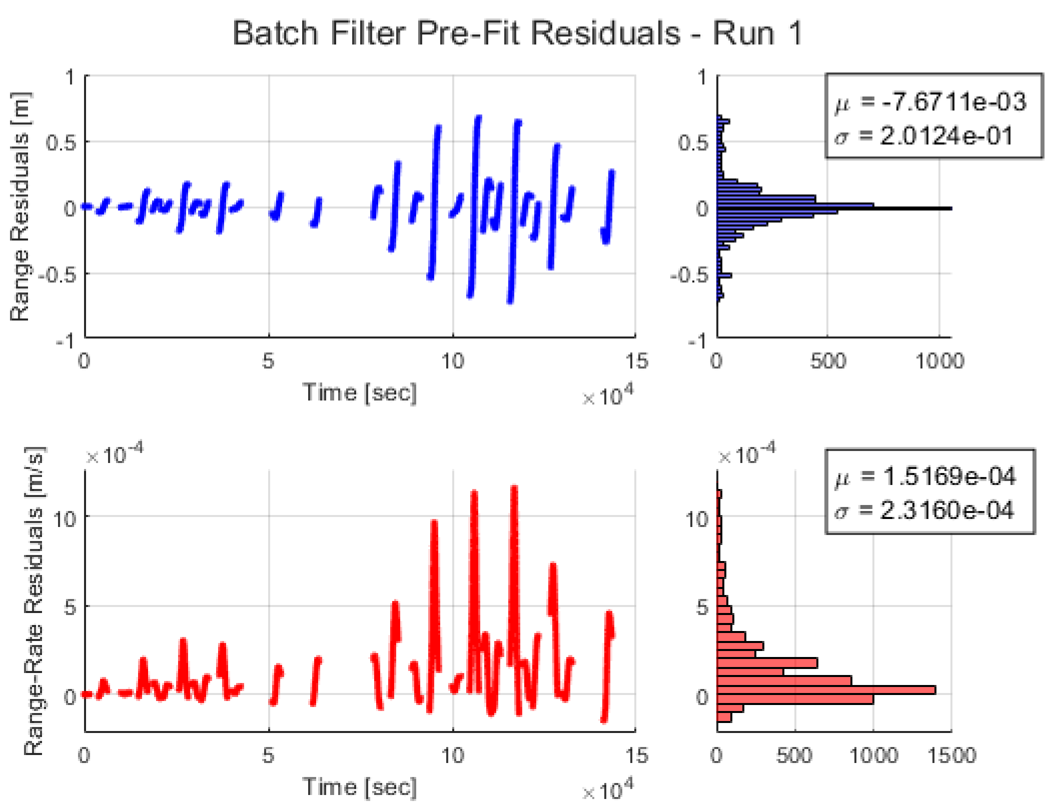

Smoother V Covariance Trace
trace(P)>0
trace(P)<0
trace(P)
-trace(P)
Time [sec]

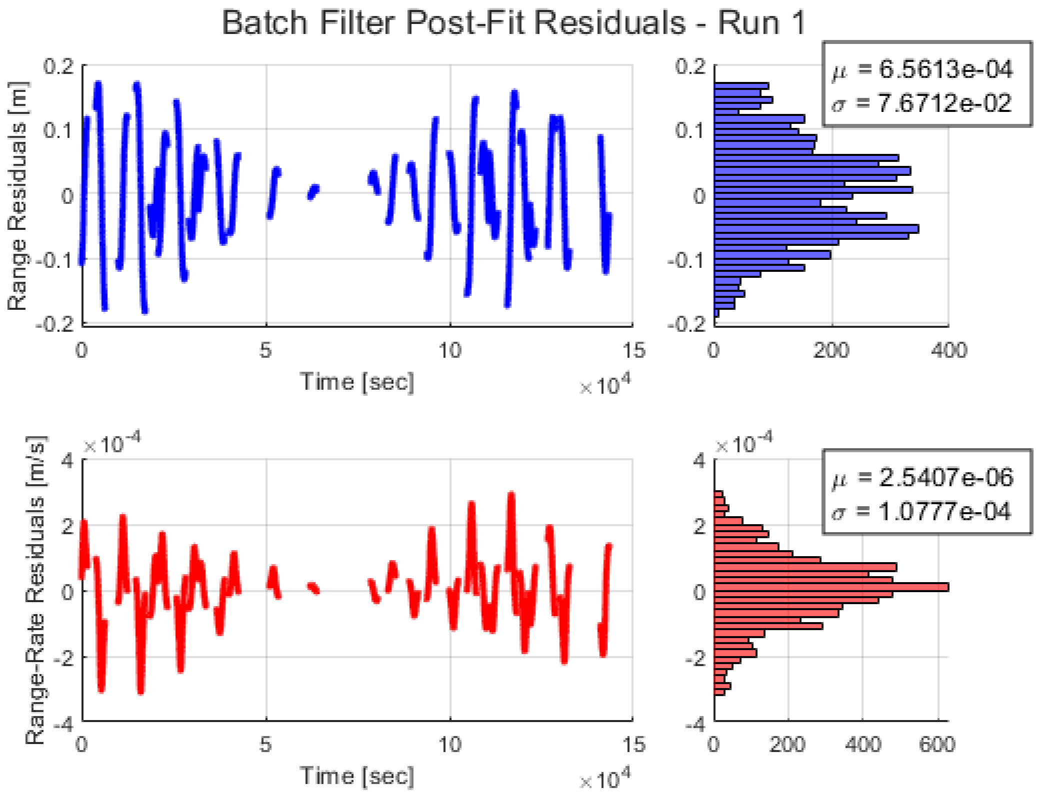

Final Smoother Position Covariance Ellipsoid, t = 143360.000 sec
μ = [-1.156e+03, -9.945e+03, -1.078e+02]T km
σX = 4.641e-05 km, σY = 9.827e-06 km, σZ = 4.306e-05 km
Z [km]
X [km]
Y [km]

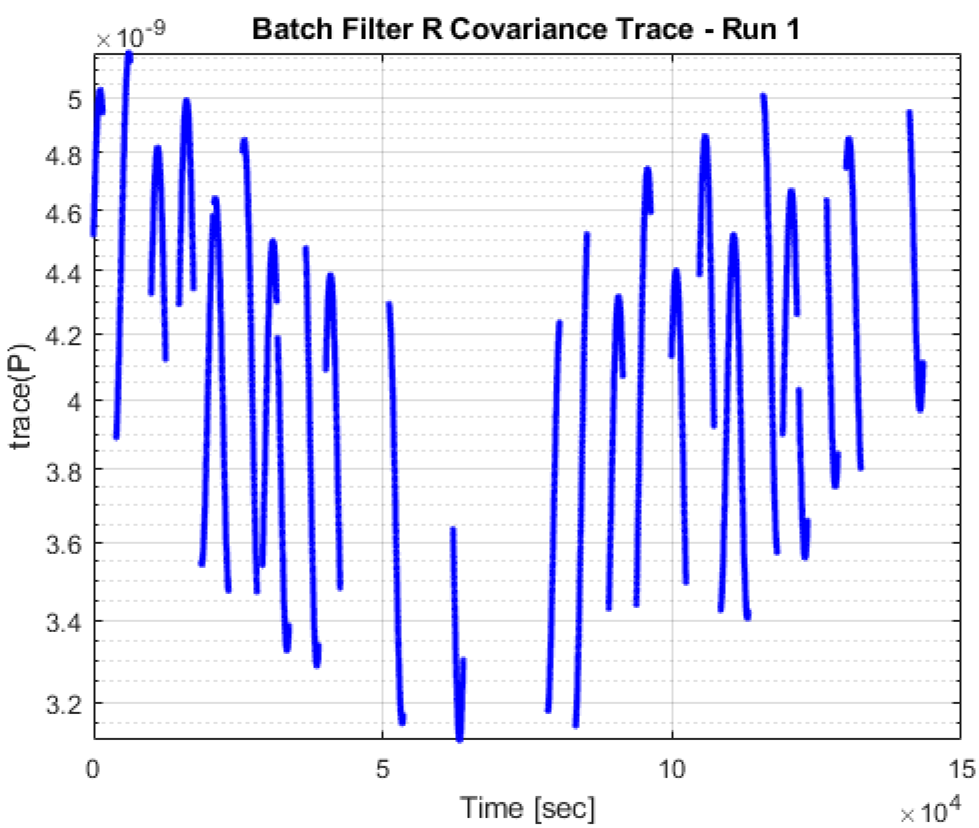

Final Smoother Velocity Covariance Ellipsoid, t = 143360.000 sec
μ = [4.803e+00, -5.196e-01, -4.052e+00]^T km/s
σXdot = 2.524e-08 km/s, σYdot = 2.288e-08 km/s, σZdot = 2.928e-08 km/s
1σ
2σ
3σ
v1
v2
v3
Xdot [km/s]
Ydot [km/s]

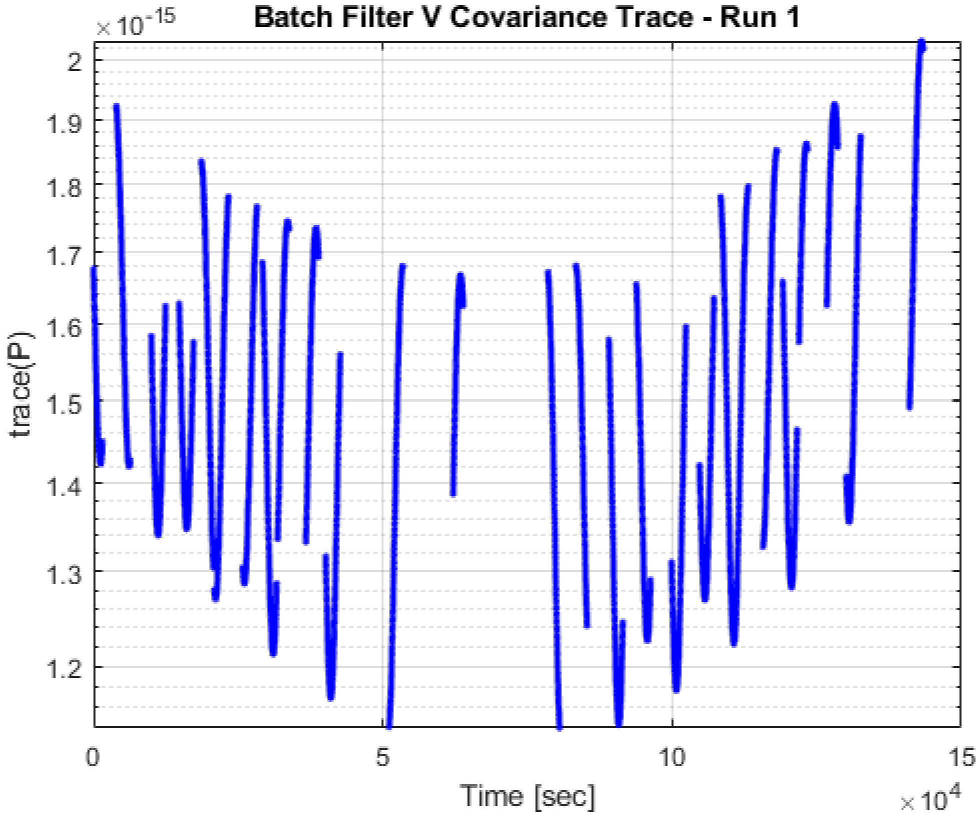

Smoother Estimated State Error (Xsmooth - Xref)
X error [km]
Y error [km]
Z error [km]
Xdot error [km/s]
Ydot error [km/s]
Zdot error [km/s]
Time [sec]
×10^4

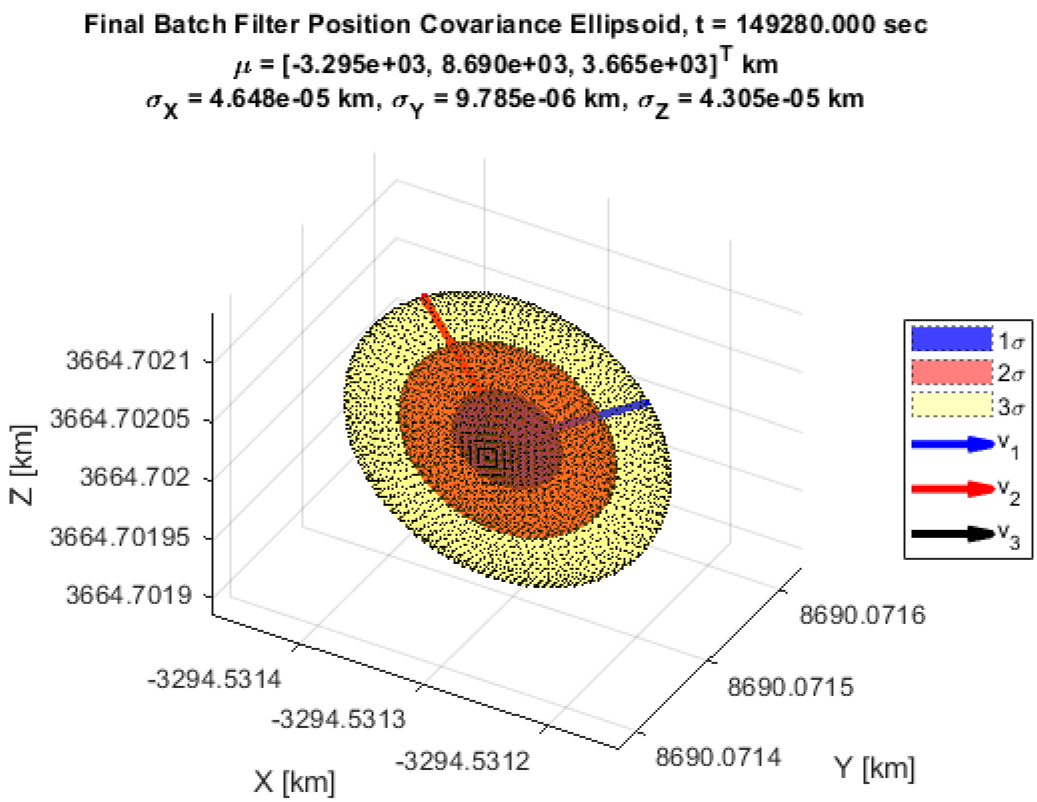

Smoother Estimated State Error (X_smooth - X_ref)
X error [km]
Y error [km]
Z error [km]
Xdot error [km/s]
Ydot error [km/s]
Zdot error [km/s]
Time [sec]
×10^4
×10^-4

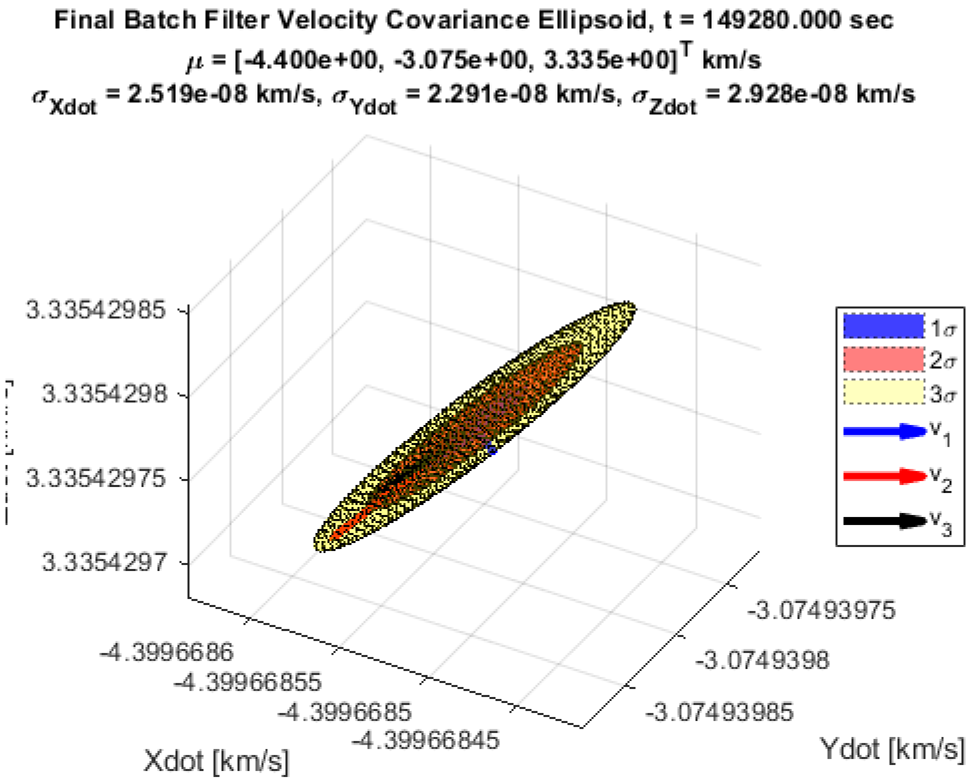

Batch Filter Pre-Fit Residuals - Run 1
Range Residuals [m]
Time [sec]
×10^4
μ = -7.6711e-03
σ = 2.0124e-01
Range-Rate Residuals [m/s]
×10^-4
μ = 1.5169e-04
σ = 2.3160e-04

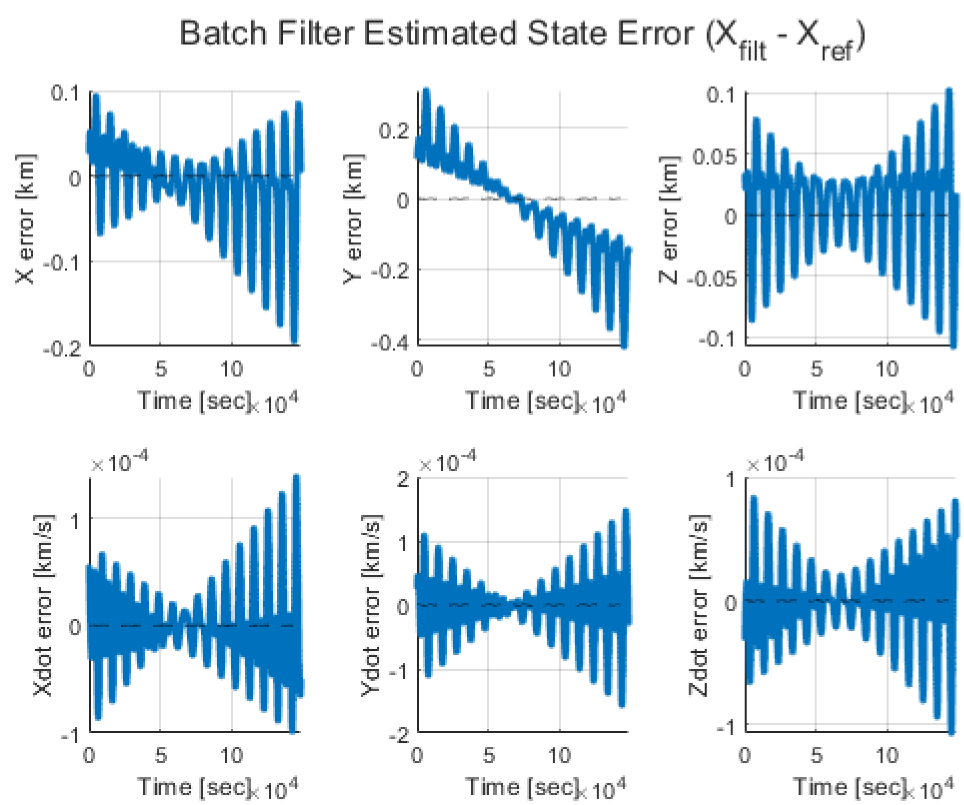

Batch Filter Post-Fit Residuals - Run 1
Range Residuals [m]
Time [sec]
$\times 10^4$
μ = 6.5613e-04
σ = 7.6712e-02
Range-Rate Residuals [m/s]
$\times 10^{-4}$
μ = 2.5407e-06
σ = 1.0777e-04


Batch Filter R Covariance Trace - Run 1
$\times 10^{-9}$
trace(P)
Time [sec]
$\times 10^4$


Batch Filter V Covariance Trace - Run 1
$\times 10^{-15}$
2
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
trace(P)
0
5
10
15
Time [sec]
$\times 10^{4}$


Final Batch Filter Position Covariance Ellipsoid, t = 149280.000 sec
μ = [-3.295e+03, 8.690e+03, 3.665e+03]^T km
σ_X = 4.648e-05 km, σ_Y = 9.785e-06 km, σ_Z = 4.305e-05 km
3664.7021
3664.70205
3664.702
3664.70195
3664.7019
Z [km]
-3294.5314
-3294.5313
-3294.5312
X [km]
8690.0716
8690.0715
8690.0714
Y [km]
1σ
2σ
3σ
v_1
v_2
v_3


Final Batch Filter Velocity Covariance Ellipsoid, t = 149280.000 sec
μ = [-4.400e+00, -3.075e+00, 3.335e+00]^T km/s
σ_Xdot = 2.519e-08 km/s, σ_Ydot = 2.291e-08 km/s, σ_Zdot = 2.928e-08 km/s
3.33542985
3.3354298
3.33542975
3.3354297
-4.3996686
-4.39966855
-4.3996685
-4.39966845
Xdot [km/s]
-3.07493975
-3.0749398
-3.07493985
Ydot [km/s]
1σ
2σ
3σ
v1
v2
v3


Batch Filter Estimated State Error (X_filt - X_ref)
X error [km]
Y error [km]
Z error [km]
Xdot error [km/s]
Ydot error [km/s]
Zdot error [km/s]
Time [sec]
× 10^4
× 10^-4

Published with MATLAB® R2023b

Table of Contents


```
function smoothOut = Smoother(filterOut)
% Function that implements a sequential smoothing algorithm for Stat OD
% problems
%   - Inputs:
%       - filterOut: Output structure from an LKF run, as defined in
%                    LKF_SNC.m
%   - Outputs:
%       - smoothOut: Smoother output structure with the following fields:
%           - tSmoothed: Smoothed estimate time
%           - xSmoothed: Smoothed deviation estimates:
%                        [xSmoothed_1, xSmoothed_2, ..., xSmoothed_k]
%           - PSmoothed: Smoothed state covariance estimates:
%                        [{PSmoothed_1}, {PSmoothed_2}, ..., {PSmoothed_k}]
%           - XSmoothed: Smoothed full state estimate, calculated as
%                        XSmoothed = Xstar + xSmoothed:
%                        [XSmoothed_1, XSmoothed_2, ..., XSmoothed_k]
%
%   By: Ian Faber, 03/03/2025
%
    % Process filterOut structure
t = filterOut.t;
xEst = filterOut.xEst;
PBarEst = filterOut.PBarEst;
PEst = filterOut.PEst;
XStar = filterOut.XStar;
Phi = filterOut.Phi;

    % Initialize outputs
tSmoothed = [];
xSmoothed = [];
PSmoothed = [];
XSmoothed = [];

    % Find value of l
l = size(xEst, 2);

    % Run smoother
x_kp1l = xEst(:,l); % Deviation at t_kp1 (t_l) given measurements at t_l
P_kp1l = PEst{l}; % Covariance at t_kp1 (t_l) given measurements at t_l
for k = (l-1):-1:1
        % Pull out filter estimates
    x_kk = xEst(:,k); % Deviation at t_k given measurements at t_k
    P_kp1k = PBarEst{k+1}; % Covariance at t_kp1 given measurements at t_k
    P_kk = PEst{k}; % Covariance at t_k given measurements at t_k
    tSmoothed = [tSmoothed; t(k)];

        % Pull out Phi(t_kp1, t_k)
```

```
    Phi_kp1 = Phi{k+1};

        % Smooth estimate
    % S_k = P_kk*Phi_kp1'*(Phi_kp1*P_kk*Phi_kp1' + Q_kp1)^-1;
    S_k = P_kk*Phi_kp1'*(P_kp1k)^-1;
    x_kl = x_kk + S_k*(x_kp1l - Phi_kp1*x_kk);

        % Smooth covariance
    P_kl = P_kk + S_k*(P_kp1l - P_kp1k)*S_k';

        % Save outputs
    xSmoothed = [xSmoothed, x_kl];
    PSmoothed = [PSmoothed; {P_kl}];
    XSmoothed = [XSmoothed, XStar(:,k) + x_kl];

        % Update for next run
    x_kp1l = x_kl;
    P_kp1l = P_kl;


end

    % Assign smoother outputs - need to flip them to be time ascending!
smoothOut.tSmoothed = flipud(tSmoothed);
smoothOut.xSmoothed = fliplr(xSmoothed);
smoothOut.PSmoothed = flipud(PSmoothed);
smoothOut.XSmoothed = fliplr(XSmoothed);

end
```

Published with MATLAB® R2023b