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

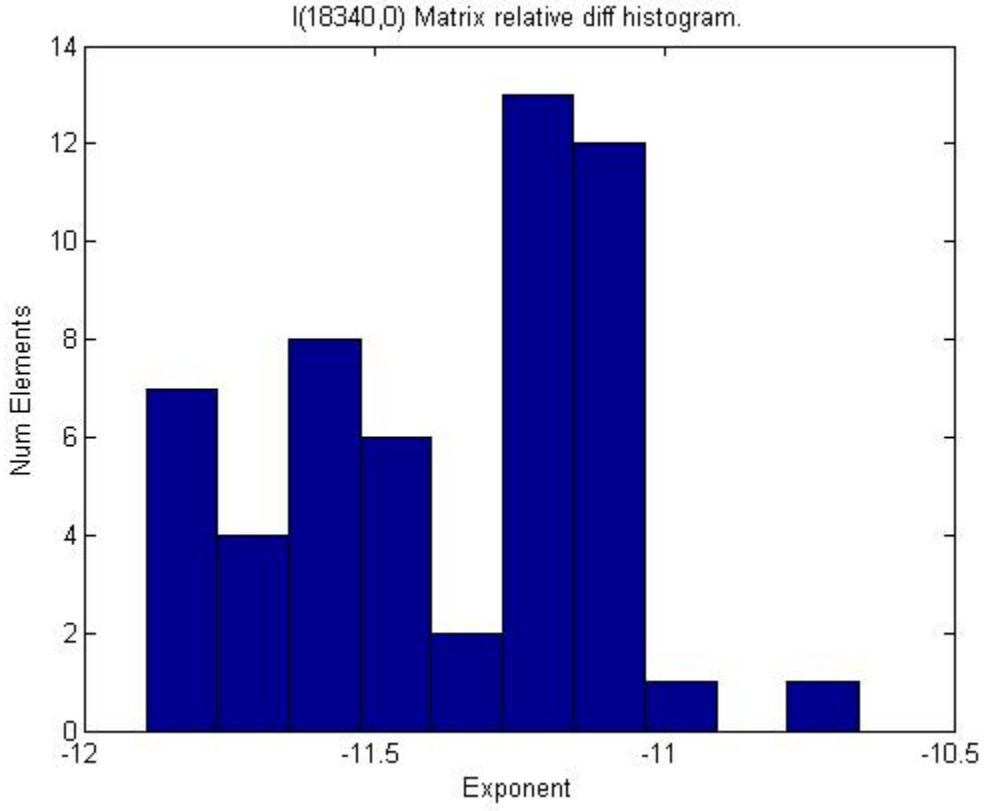
```
dt = 0.1;
times = 0:dt:18340;
ode_opts = odeset('RelTol', 1e-12, 'AbsTol', 1e-20);

% 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;
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

% CKF init
x_est = x0_ap;
P = P0;
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);
% 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(1:3,1:3));
        else
            P = (I-K*H_tilda)*P_ap;
            P_trace_store(ii) = trace(P(1:3,1:3));
        end

    end
```

```
% Estimated state at t=0
STM_18340_0 = eye(consts.state_len);
STM_18340_0(1:important_block(1),1:important_block(2)) = ...
    reshape(X_store(end,consts.state_len+1:end), ...
    important_block(1), important_block(2));
est_x0 = STM_18340_0\x_est;

% Result of accumulated STM
Phi_given = load('BatchPhi.mat');
relDiffSTMend = abs((STM_accum-Phi_given.Phi_t18340)./Phi_given.Phi_t18340);
figure
hist(reshape(log10(relDiffSTMend(1:6, 1:9)),6*9,1))
title('I(18340,0) Matrix relative diff histogram.')
xlabel('Exponent')
ylabel('Num Elements')
```

```
Warning: Matrix is close to singular or badly scaled. Results may be inacc

RCOND =  5.083090e-17.
```



Compare results with Online Solution and Batch Solution

The relative difference between CKF and the online solution is worse than the batch-vs-online difference. I suspect it has to do with the usage and update of the covariance matrix at each step, since it's not well-conditioned.

```
% Covariance matrix rel diffs from batch soln
% relDiffCov = abs((STM_accum\P/(STM_accum')-P0_est_batch)./P0_est_batch);
% figure
% hist(reshape(log10(relDiffCov),18*18,1))
% title('P Matrix relative diff histogram.')
% xlabel('Exponent')
% ylabel('Num Elements')

format long
est_x_18340 = x_est
est_x0
% Compare with online answers
x_hat_given = load('BatchXhat');
x_diff = abs((est_x0 - x_hat_given.xhat_pass1)./x_hat_given.xhat_pass1);
fprintf('Estimated State Deviation Error with online answers\n')
for ii = 1:consts.state_len
    fprintf('%.0e\n',x_diff(ii))
end
fprintf('\n')

% Compare with batch-derived state.
x_diff = abs((est_x0 - x_est_batch)./x_est_batch);
fprintf('Estimated State Deviation Error with batch solution\n')
for ii = 1:consts.state_len
    fprintf('%.0e\n',x_diff(ii))
end
```

```
est_x_18340 =

   1.0e+06 *

  -0.000587587573647
  -0.001075141074919
   0.001887072629805
   0.000000417695298
   0.000001990050205
   0.000001226254719
  -8.942208560827096
  -0.000000000000656
   0.000000155506456
   0.000000000001829
   0.000000000001353
  -0.000000000000249
  -0.000010552784646
   0.000009947713568
   0.000005797860704
  -0.000005750500009
   0.000002291907091
   0.000001485087375


est_x0 =
```

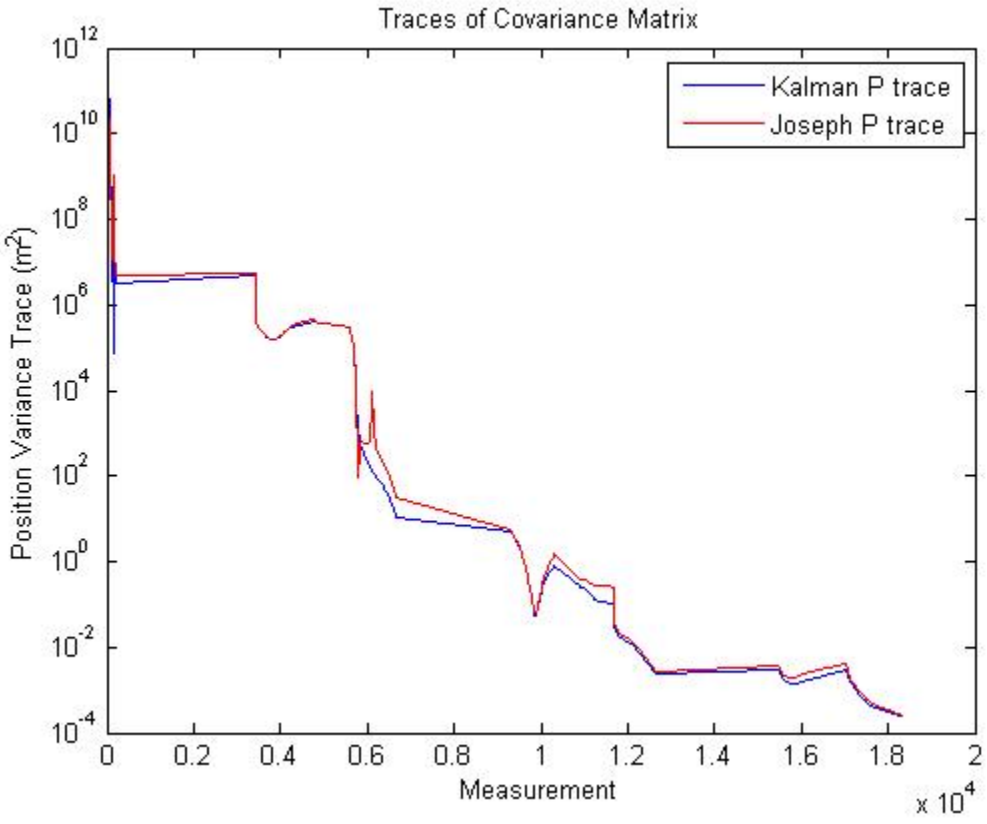
```
   1.0e+06 *

   0.000000003676901
  -0.000000315587998
  -0.000000145958197
   0.000000040901421
   0.000000032792862
  -0.000000014735473
  -8.942208560827096
  -0.000000000000656
   0.000000155506456
   0.000000000001829
   0.000000000001353
  -0.000000000000249
  -0.000010552784646
   0.000009947713568
   0.000005797860704
  -0.000005750500009
   0.000002291907091
   0.000001485087375

Estimated State Deviation Error with online answers
1e+00
2e-01
2e-01
8e-04
1e-03
1e-03
5e-02
2e-03
5e-02
2e-02
2e-02
2e-02
1e-03
4e-03
6e-04
5e-03
2e-02
2e-02

Estimated State Deviation Error with batch solution
1e+00
2e-01
2e-01
8e-04
1e-03
1e-03
6e-02
2e-03
5e-02
2e-02
2e-02
2e-02
```

```
1e-03
4e-03
6e-04
5e-03
2e-02
2e-02
```

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.

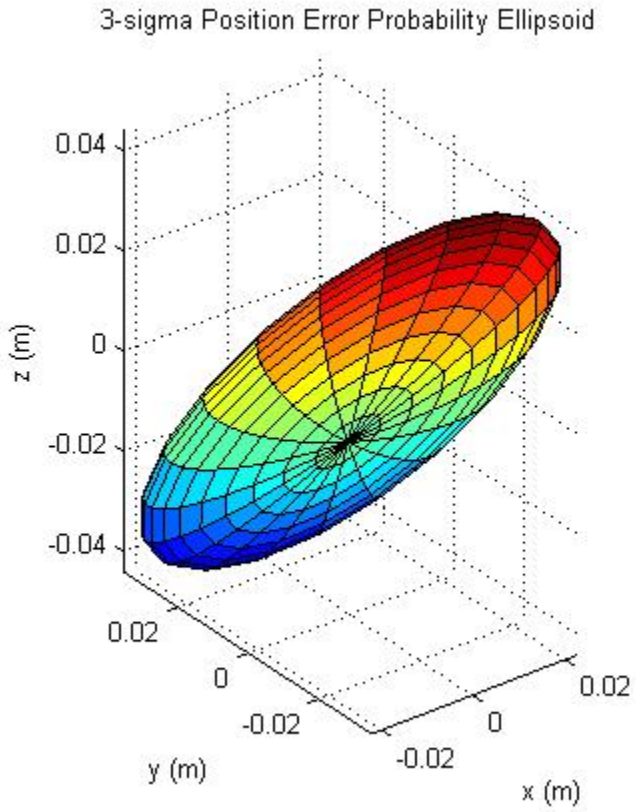
```
figure
semilogy(ObsData(:,1),abs(P_trace_store))
hold on
semilogy(ObsData(:,1),abs(P_joseph_store),'r')
legend('Kalman P trace', 'Joseph P trace')
title('Traces of Covariance Matrix')
xlabel('Measurement'), ylabel('Position Variance Trace (m^2)')
```



Error Ellipsoid

```
[U,Pprime] = eigs(P0_est_batch(1:3,1:3));
Semi = [sqrt(9*Pprime(1,1)) sqrt(9*Pprime(2,2)) sqrt(9*Pprime(3,3))];
figure
```

```
plotEllipsoid(U,Semi);
title('3-sigma Position Error Probability Ellipsoid')
xlabel('x (m)'),ylabel('y (m)'),zlabel('z (m)'),
```



Published with MATLAB® R2013b

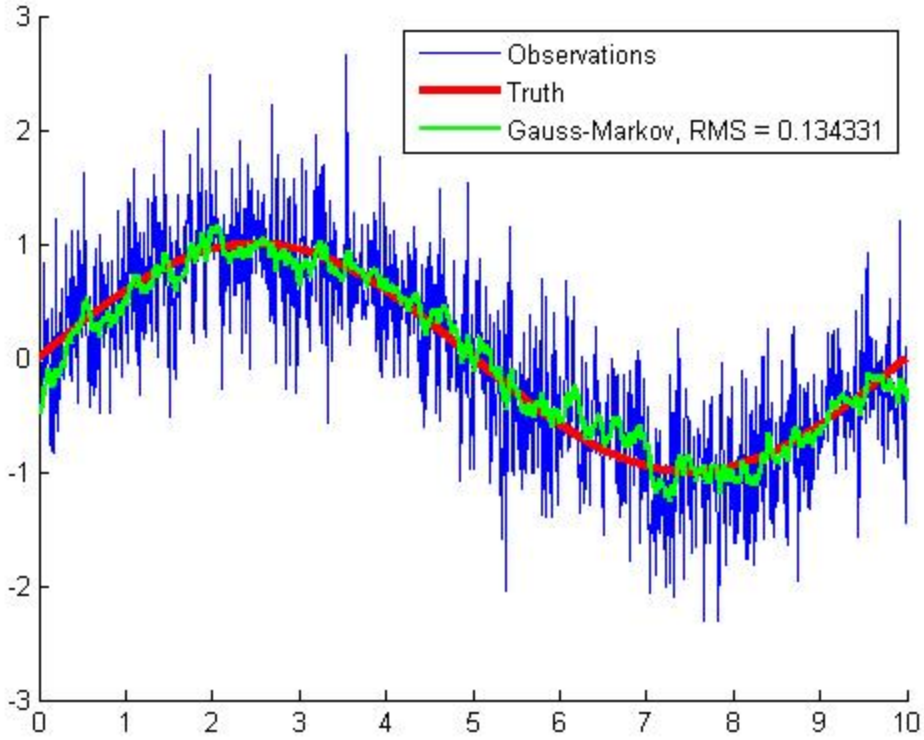
Markov

Initialize

```
clearvars -except function_list pub_opt P_joseph_store
global function_list;
function_list = {};
close all

obs_data = load('hw11.dat');

T = 10;
truth = sin(obs_data(:,1) * 2*pi/T);
figure
hold on
plot(obs_data(:,1),obs_data(:,2))
plot(obs_data(:,1),truth,'r','LineWidth',3)

eta0_ap = 0;
P0_ap = 1;
R = 1;
Q = 1;
num_obs = length(obs_data(:,1));
eta_est_store = zeros(num_obs,1);

eta_est = eta0_ap;
P = P0_ap;

tc_vec = [.1 1 3 5];
sig_vec = [.5 4 2 1];

best_RMS = -1;
for idx = 1:length(tc_vec)
time_const = tc_vec(idx);
beta = 1/time_const;
sigma = sig_vec(idx);
eta_est_store_inner = zeros(num_obs,1);
RMS_accum = 0;
for ii = 1:num_obs
    % STM
    if ii == 1 %measurement at t = 0
        m = 1;
    else
        m = exp(-beta*(obs_data(ii,1)-obs_data(ii-1,1)));
    end
    STM = m;

    % Time Update
    eta_ap = STM*eta_est;
    gamma = sqrt(sigma*sigma/2/beta*(1-m*m));
    P_ap = STM*P*STM + gamma*Q*gamma;
```

```
        % Kalman gain
        K = P_ap/(P_ap+1); % valid for this 1D case

        % Measurement Update
        Y = obs_data(ii,2);
        eta_est = eta_ap +K*(Y-eta_ap); %H~ == 1 in this case.
        P = K;

        eta_est_store_inner(ii) = eta_est;
        RMS_accum = RMS_accum + (truth(ii)-eta_est)*(truth(ii)-eta_est);

    end
    RMS = sqrt(RMS_accum/num_obs);
    fprintf(sprintf('RMS for tau=%f, sigma=%f: %f\n',time_const, sigma, RMS));
    if best_RMS == -1
        best_RMS = RMS;
        eta_est_store = eta_est_store_inner;
    else
        if RMS < best_RMS
            eta_est_store = eta_est_store_inner;
        end
        best_RMS = min(best_RMS,RMS);
    end
    end
    plot(obs_data(:,1),eta_est_store,'g','LineWidth',2)
    legend('Observations', 'Truth', sprintf('Gauss-Markov, RMS = %f',best_RMS))
    % figure
    % hist(obs_data(ii,2)-eta_est_store);
```

```
        RMS for tau=0.100000, sigma=0.500000: 0.630702
        RMS for tau=1.000000, sigma=4.000000: 0.224250
        RMS for tau=3.000000, sigma=2.000000: 0.170042
        RMS for tau=5.000000, sigma=1.000000: 0.134331
```



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 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

```
function accel = drag_accel( state, drag_data )
%drag_accel calculate drag on spacecraft
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

Cd = drag_data.Cd;
A = drag_data.A;
m = drag_data.m;

rho0 = 4e-13; %kg/m3
r0 = 7298.145; %km
H = 200.0; %km
theta_dot = 7.29211585530066e-5; %rad/s

if isfield(drag_data, 'model_params')
    if isfield(drag_data.model_params, 'rho0')
        rho0 = drag_data.model_params.rho0;
    end
    if isfield(drag_data.model_params, 'r0')
        r0 = drag_data.model_params.r0;
    end
    if isfield(drag_data.model_params, 'H')
        H = drag_data.model_params.H;
    end
    if isfield(drag_data.model_params, 'theta_dot')
        theta_dot = drag_data.model_params.theta_dot;
    end
end

r = norm(state(1:3));

rho = rho0*exp(-(r-r0)/H);
rel_wind = [state(4) + theta_dot*state(2);
    state(5) - theta_dot*state(1);
    state(6)];

accel = -0.5*Cd*A/m*rho*rel_wind*norm(rel_wind);

end
```

Published with MATLAB® R2013b

```
function fcnPrintQueue( filename )
global function_list;
if exist('function_list', 'var')
    file_in_list = 0;
    for idx = 1:length(function_list)
        if strcmp(function_list(idx), filename);
            file_in_list = 1;
            break
        end
    end
    if ~file_in_list
%         fprintf('%s\n', filename);
        function_list = [function_list; filename];
    end
end
end
```

Published with MATLAB® R2013b

```
function accel = J2_accel( pos, params )
%J2_accel Acceleration due to J2
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

% Defaults (earth)
J2 = 0.00108248;
mu = 398600.4; %km3/s2
Re = 6378.145; %km

% Use input params if desired
if nargin > 1 % There are params
    if isfield(params, 'J2')
        J2 = params.J2;
    end
    if isfield(params, 'mu')
        mu = params.mu;
    end
    if isfield(params, 'Re')
        Re = params.Re;
    end
end

%Calculate accel
r = norm(pos);
z = pos(3);
const = 1.5*mu*J2*Re*Re/(r*r*r*r*r);
sin_sq_phi = z*z/(r*r);

accel = const*[5*sin_sq_phi - 1;
    5*sin_sq_phi - 1;
    5*sin_sq_phi - 3].*pos;
end
```

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

```
function state_dot = two_body_state_dot(t, state, opts)
%two_body_state_dot   Return state_dot given state. Used for numerical
%integration
% The first 6 elements are assumed to be r and v. If a state transition
% matrix (STM) is to be calculated as well, opts.OD needs to be set up.
% Currently assumes r/v are the only state elements that have non-zero
% derivatives.
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

state_dot = zeros(length(state),1);
state_dot(1:3) = state(4:6);
mu = 3.986e5; % km3/s2
if isfield(opts, 'mu')
    mu = opts.mu;
end

r_vec = (state(1:3));
r = norm(r_vec);
state_dot(4:6) = -mu * r_vec/(r*r*r);

if isfield(opts, 'J2') && isfield(opts.J2, 'use')
    if opts.J2.use == 1
        if isfield(opts.J2, 'params')
            state_dot(4:6) = state_dot(4:6) + J2_accel(state(1:3), opts.J2.params)
        else
            state_dot(4:6) = state_dot(4:6) + J2_accel( state(1:3) );
        end
    end
end
if isfield(opts, 'drag') && isfield(opts.drag, 'use')
    if opts.drag.use == 1
        state_dot(4:6) = state_dot(4:6) + drag_accel( state, opts.drag );
    end
end

if isfield(opts, 'OD')
    if opts.OD.use == 1
        opts.OD.state_len;
        % The OD.state_len is the length of the estimation state. The rest
        % is the STM, numerically propagated with the A-Matrix
        A = opts.OD.A_mat_handle(state(1:opts.OD.state_len),opts.OD.A_params);

        % Block matrix multiplication
%         long_dim = 9;
%         STM = zeros(long_dim);
        STM = reshape(state(opts.OD.state_len+1:end),...
            opts.OD.A_params.important_block(1),...
            opts.OD.A_params.important_block(2));
        STM_dot = ...
            A(1:opts.OD.A_params.important_block(1),1:opts.OD.A_params.important_b
            *STM;
        % Pack up the important stuff
```

```
            state_dot(opts.OD.state_len+1:end) = reshape(STM_dot,...
                opts.OD.A_params.important_block(1)*opts.OD.A_params.important_block(2
        end
    end
```

Published with MATLAB® R2013b