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

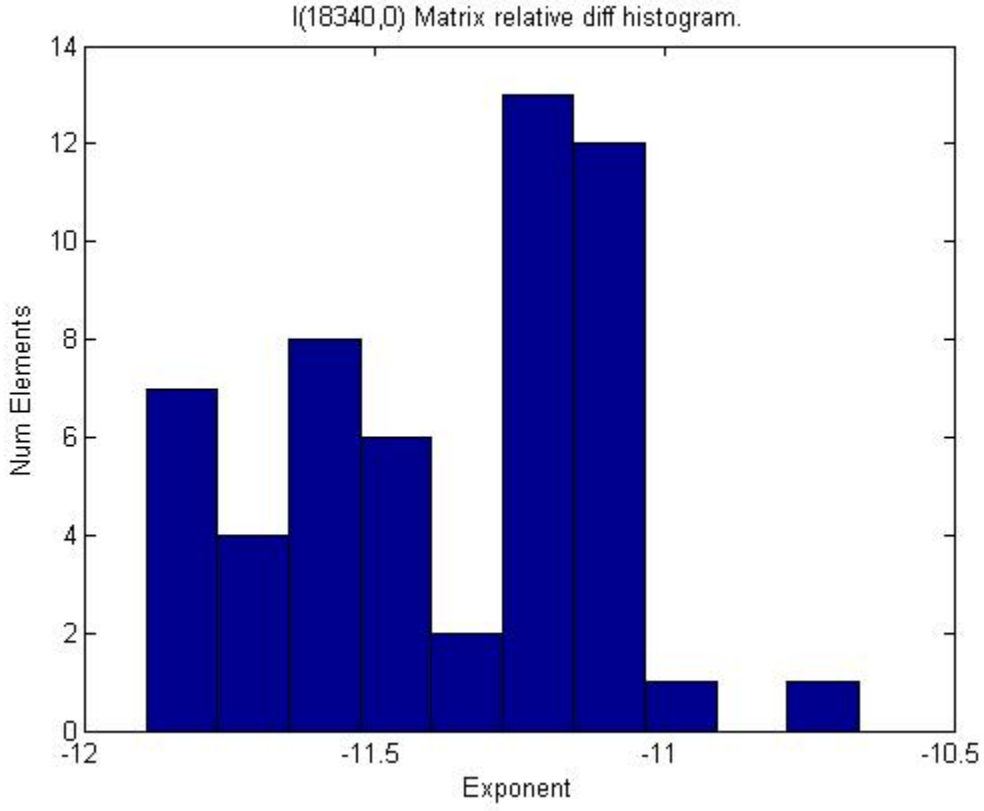
```
dt = 0.1;
times = 0:dt:18340;
ode_opts = odeset('RelTol', 1e-12, 'AbsTol', 1e-20);

% 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 =
```

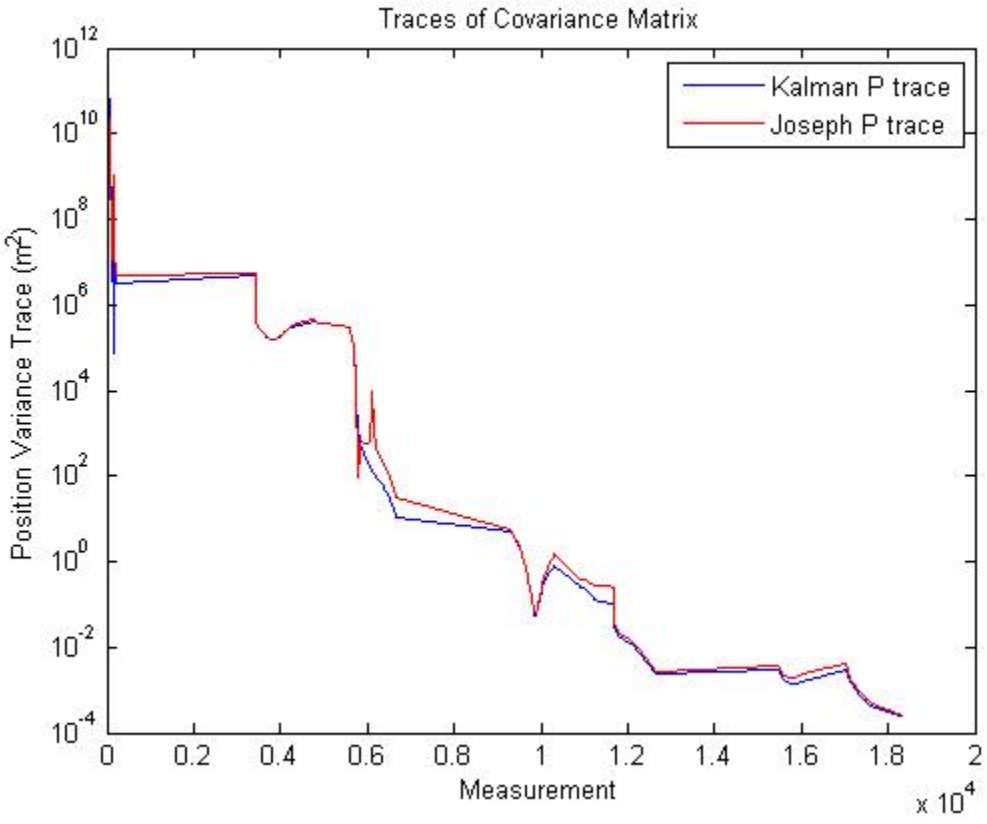
```
   1.0e+06 *

   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.

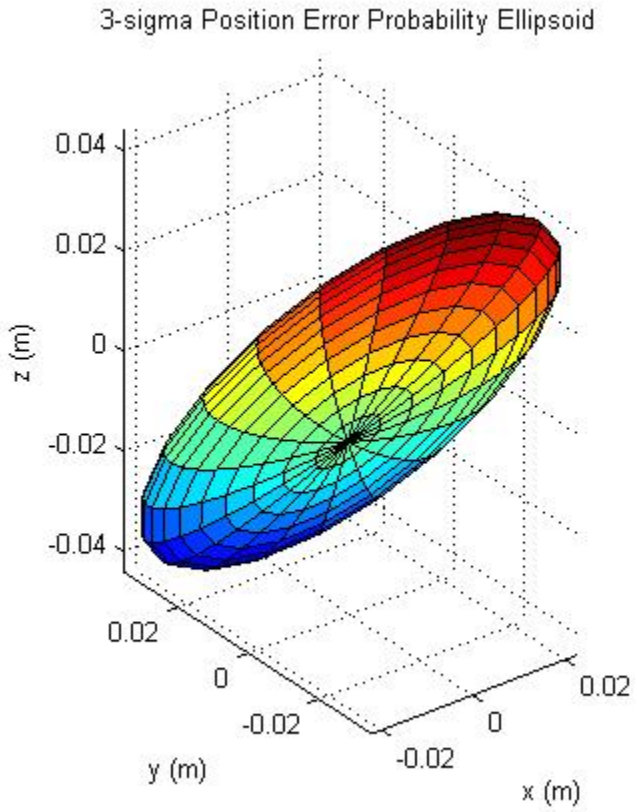
```
figure
semilogy(ObsData(:,1),abs(P_trace_store))
hold on
semilogy(ObsData(:,1),abs(P_joseph_store),'r')
legend('Kalman P trace', 'Joseph P trace')
title('Traces of Covariance Matrix')
xlabel('Measurement'), ylabel('Position Variance Trace (m^2)')
```



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