HW 10: Sequential Processor for the Term Project

Table of Contents

Initialize

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

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

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

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

x0_ap = zeros(consts.state_len,1);

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

Sequential Processor

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

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

% Store off every 20 seconds of data
```

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

[num_obs, ~] = size(ObsData);

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

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

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

end
```

CKF init

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

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

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

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

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

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

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

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

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

    % Kalman gain
    K = alpha*W_ap*F;

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

    % sqrt of the covariance:
```

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

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

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

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

    if ii == 1
        x_est_Potter = x_est;
    end
end

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

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

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



%
```

Covariance Matrix Traces

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

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

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


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

Published with MATLAB® R2013b