HW 10: Sequential Processor for the Term Project

Table of Contents

Initialize

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

stat_od_proj_init
ObsData = load('J3obs_HW11.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;
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 = 1;
if use_joseph
    P_joseph_store = zeros(num_obs,1);
else
    P_trace_store = zeros(num_obs,1);
end

% SNC
use_SNC = 0;
SNC_sigma = 2e-5;
Q = eye(3)*SNC_sigma*SNC_sigma;

STM_accum = eye(consts.state_len);

RMS_accum = 0;
obs_error_store = zeros(2,num_obs);
```

```
inrtl_state_est_store = zeros(6,num_obs);
variance_store = zeros(6,num_obs);
num_RMS_meas = 0;
obs_time_store = zeros(num_obs,1);

% 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

    % Time update
    STM_accum = STM_obs2obs*STM_accum;
    x_ap = STM_obs2obs*x_est;
    P_ap = STM_obs2obs*P*STM_obs2obs';
    delta_t = obs_time - obs_time_last;
    if use_SNC && delta_t < 100 % add process noise if needed
        Gamma = delta_t*[eye(3)*delta_t/2;eye(3)];
        P_ap(1:6,1:6) = P_ap(1:6,1:6) + Gamma*Q*Gamma';
    end
    obs_time_last = obs_time;

    % 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

        obs_error = y - H_tilda*x_est;
        if obs_time >= 100*60
            RMS_accum = RMS_accum + obs_error'*W*obs_error;
            num_RMS_meas = num_RMS_meas + 1;
        end
        obs_error_store(:,ii) = obs_error;
        obs_time_store(ii) = obs_time;
        inrtl_state_est_store(:,ii) = state_at_obs(1:6)'+x_est(1:6);
        variance_store(:,ii) = diag(P(1:6,1:6));
    end

    RMS = sqrt(RMS_accum./num_RMS_meas);
    figure
    subplot(2,1,1)
    plot(obs_time_store,obs_error_store(1,:),'.')
    title('Range Post-Fit Residuals')
    ylabel('m')
    subplot(2,1,2)
    plot(obs_time_store,obs_error_store(2,:),'.')
    title('Range-Rate Post-Fit Residuals')
    ylabel('m/s')
    xlabel('Time (s)')

    truth_data = load('J3truth_HW11.txt');
    estimation_errors = inrtl_state_est_store-truth_data(:,2:7)';
    RMS_errors = sqrt(sum(estimation_errors.*estimation_errors,2)./ ...
        length(estimation_errors));
    pos_plots = figure;
    vel_plots = figure;
    axis_label = {'x', 'y', 'z'};
    for ii = 1:3
        figure(pos_plots);
        subplot(3,1,ii)
        hold on
        plot (obs_time_store,estimation_errors(1,:),'.');
        plot (obs_time_store,sqrt(abs(variance_store(1,:))),'r--');
        plot (obs_time_store,-sqrt(abs(variance_store(1,:))),'r--');
        title(sprintf('%s position error',axis_label{ii}))
        ylabel('m'),xlabel('Time (s)')
```

```
    figure(vel_plots);
    subplot(3,1,ii)
    hold on
    plot (obs_time_store,estimation_errors(1+3,:),'.');
    plot (obs_time_store,sqrt(abs(variance_store(1+3,:))),'r--');
    plot (obs_time_store,-sqrt(abs(variance_store(1+3,:))),'r--');
    title(sprintf('%s velocity error',axis_label{ii}))
    ylabel('m/s'),xlabel('Time (s)')
end
```

Published with MATLAB® R2013b