AE4890-11 Planetary Sciences I - Assignment 1

Authors:
Jake Smith - 5093775
Akke Toeter - 1507958

19-10-2019

1 Introduction

A

B

C

D

E

F

G

References

Appendix 1: Code for extended Kalman filter

```
%% Iterative least squares

%Dimensionalising appended matrices used for each satellite
x_j = zeros(6,1);
x_k = zeros(6,1);
x_kalman = zeros(6,1);
x_final = zeros(6,1);
A = zeros(8,6);
y = zeros(8,1);
K_k = zeros(6,6);
y_k = zeros(8,10);
H = zeros(8,6);
mu = 3.98600000e14; %m^3/s^-2
rho_cc = zeros(11,8);
rho_without_cc = zeros(11,8);
deltay = zeros(8,11);
phi_kj = zeros(6);
phi_kj_0 = eye(6);
dx = zeros(6,1);
phi = zeros(6,6);
I = eye(6);

%Initial conditions
x_j = [x_precise(1,1);
      y_precise(1,1);
      z_precise(1,1);
      (x_precise(1,2) - x_precise(1,1))/dt;
      (y_precise(1,2) - y_precise(1,1))/dt;
      (z_precise(1,2) - z_precise(1,1 ))/dt;];
 x_j = x_j*1000*1.1;

% Use calculated receiver clock error - use values on Brightspace (to be uploaded)
CCSA = 1.0e+05 *  [-3.521560447372325;
                   -3.526008605949202;
                   -3.530126526071388;
                   -3.534638014464830;
                   -3.539314654653520;
                   -3.543740183357683;
                   -3.547870435889991;
                   -3.552313411363543;
                   -3.556403241716776;
                   -3.560824802120515;
                   -3.564958307332973]; % In metres

% Adjustable parameter
Pj_init = 30.03;
P_j = eye(6)*Pj_init;

% Change the weighting of the observations against the states
R_init = 0.01;
R_j = eye(8)*R_init;

% Adjustable parameter
Gamma_init = 2;
Gamma = eye(6)*Gamma_init;

% Adjustable parameter
q = 10e-8;
Q = eye(6)*q;
dt = 10;
```

```
% Frid search parameter ranges
Gamma_param = [0.001     0.01  0.1  0.5  2  5  10];
P_param = [0.01  0.1  1  10  30  100  500  1000];
Q_param = 10e-8 * [  0.01          0.1  1    10        100  1000];
R_param = [      0.0005   0.001  0.05  0.1  1  10  100  1000  10e8];
X_param = [      0.5  0.95  0.99  1.01  1.05  1.5  2];

Radii = zeros(length(Gamma_param),length(P_param),length(Q_param),length(R_param),
    length(X_param),10);

% Iterate through gridsearch configurations
for it_Gamma = 1:length(Gamma_param)
    Gamma = eye(6)*Gamma_param(it_Gamma);

    for it_P = 1:length(P_param)
        % Create covariance matrix
        P_j = eye(6)* P_param(it_P);

        for it_Q = 1:length(Q_param)
            Q = eye(6)*Q_param(it_Q);

            for it_R = 1:length(R_param)
                % Change the weighting of the observations against the states
                R_j = eye(8)* R_param(it_R);

                for it_X = 1:length(X_param)
                    % Initialize parameters: dependent parameters
                    x_j = [x_precise(1,1);
                         y_precise(1,1);
                         z_precise(1,1);
                         (x_precise(1,2) - x_precise(1,1))/dt;
                         (y_precise(1,2) - y_precise(1,1))/dt;
                         (z_precise(1,2) - z_precise(1,1))/dt;];
                    x_j=x_j*1000*X_param(it_X);

                    % Reset phi_kj for every param setting
                    phi_kj = zeros(6);

                    for epochno = 1:10

                        r_j = sqrt(x_j(1)^2 + x_j(2)^2 + x_j(3)^2);

                        x_dot_jk = [x_j(4);
                             x_j(5);
                             x_j(6);
                            -x_j(1)*mu/r_j^3;
                            -x_j(2)*mu/r_j^3;
                            -x_j(3)*mu/r_j^3];

                        % Compute transition matrix
                        dFdx = [0  0  0  1  0  0;
                        0  0  0  0  1  0;
                        0  0  0  0  0  1;
                        (3*mu*r_j^(-5)*x_j(1)^2-mu*r_j^(-3))  (3*mu*x_j(1)*x_j(2)*
                            r_j^(-5))  (3*mu*x_j(1)*x_j(3)*r_j^(-5))  0  0  0;
                        (3*mu*x_j(1)*x_j(2)*r_j^(-5))  (3*mu*r_j^(-5)*x_j(2)^2-mu*
                            r_j^(-3))  (3*mu*x_j(2)*x_j(3)*r_j^(-5))  0  0  0;
                        (3*mu*x_j(1)*x_j(3)*r_j^(-5))  (3*mu*x_j(2)*x_j(3)*r_j^(-5
                            )  (3*mu*r_j^(-5)*x_j(3)^2-mu*r_j^(-3))  0  0  0];

                        % Generate phi
```

```matlab
                        if epochno > 1
                            phi_kj = phi_kj + (dFdx * phi_kj * dt);
                        else
                            phi_kj = phi_kj_0 + (dFdx * phi_kj * dt);
                        end

                        % Use Euler intgeration
                        x_k = x_j + x_dot_jk * dt;

                        % Eq 10.38
                        P_k = phi_kj * P_j * phi_kj'+Gamma * Q * Gamma';

                        % compute rho with and without cc.
                        for satno = 1:8
                            rho_cc(epochno,satno) = sqrt((x_k(1) - x_gps_interpolated(epochno,satno)*1000)^2 + (x_k(2) - y_gps_interpolated(epochno,satno)*1000)^2 + (x_k(3)-z_gps_interpolated(epochno,satno)*1000)^2)+(CCSA(epochno,1) - c*cc_gps_interpolated(epochno,satno));
                            rho(epochno,satno) = sqrt((x_k(1) - x_gps_interpolated(epochno,satno)*1000)^2 + (x_k(2) - y_gps_interpolated(epochno,satno)*1000)^2 + (x_k(3)-z_gps_interpolated(epochno,satno)*1000)^2);
                        end

                        % fill H
                        for satno = 1:8
                            H(satno,:) =  [(x_k(1)-x_gps_interpolated(epochno,satno)*1000)/rho(epochno,satno) (x_k(2)-y_gps_interpolated(epochno,satno)*1000)/rho(epochno,satno) (x_k(3)-z_gps_interpolated(epochno,satno)*1000)/rho(epochno,satno) 0 0 0];
                        end

                        % Include Receiver - Transmitter CC
                        for satno = 1:8
                            y_k(satno,epochno) = C1(epochno,satno)  - rho_cc(epochno,satno);
                        end

                        % Kalman gain
                        K_k = P_k * (H)' * inv(H * P_k * H' + R_j);

                        % Updated x using Kalman gain
                        x_kalman = K_k * y_k;
                        x_final = x_k + K_k * (y_k-H*x_k);

                        % Updated covariance matrix estimation
                        P_k = (I - K_k * H) * P_k;

                        x_j = x_final;

                        % Store result
                        Radii(it_Gamma,it_P,it_Q,it_R,it_X,epochno) = sqrt(x_k(1,epochno)^2+x_k(2,epochno)^2+x_k(3,epochno)^2);
                    end
                end
            end
        end
    end
end
```

```
%% Plots

for i = 1:1
    % Compute radius
    r_estim_stored = Radii(1,1,1,1,1,:);
    r_exact_stored = r_exact(10, x_precise, y_precise, z_precise);

    % Plot to latex example
    dataseries_1 = r_estim_stored
    dataseries_2 = r_exact_stored;

    obj_mult = PlotMultipleLines;
    filename = "plot_a";
    plot_altitudes(obj_mult,dataseries_1,dataseries_2,filename);
end
```