```
>> type Project_Code.m

%Nathan Lutes
%Project main script
%ME6481
%5/7/2019
%%
clc
clear
try
    close all
catch
end
%%
%numerical integration
%Define numerical integration values
%for this, we will need some constant values
J2 = 0.0010826; %constant multiplying the J2 effect
u = 3.98600442E5; %KM3/s2 Earth's gravitational parameter
rEarth = 6378.14; %kilometers    Radius of the Earth
initial_distance = 100; %kilometers
initial_lat = 45; %degrees
initial_long = 45; %degrees
V = 7.843;  %KM/s   Orbit Velocity
%initial state values
X0 = [(rEarth + initial_distance)*cos(initial_lat * (pi/180))*cos(initial_long *↙
(pi/180));
    (rEarth + initial_distance)*cos(initial_lat * (pi/180))*sin(initial_long * (pi/180));
    (rEarth + initial_distance)*sin(initial_lat * (pi/180));
    -V * cos(initial_lat * (pi/180))*cos(initial_long * (pi/180));
    -V * cos(initial_lat * (pi/180))*sin(initial_long * (pi/180));
    V * sin(initial_lat * (pi/180))];
%time constant
T = 0.1; %seconds
%time duration
t = 720; %seconds
%now define f, the function used to numerically integrate the actual states
syms x y z xdot ydot zdot
%states
X = [x y z xdot ydot zdot]';
r = sqrt(X(1)^2 + X(2)^2 + X(3)^2);
%matrices
A_X = [0 0 0 1 0 0;
    0 0 0 0 1 0;
    0 0 0 0 0 1;
    -u/(r^3) 0 0 0 0 0;
    0 -u/(r^3) 0 0 0 0;
    0 0 -u/(r^3) 0 0 0];
B = [0 0 0;
    0 0 0;
    0 0 0;
```

```
    1 0 0;
    0 1 0;
    0 0 1];
C = [1 0 0 0 0 0;
    0 1 0 0 0 0;
    0 0 1 0 0 0];
%J2 effect for each axis
f_x = J2*((3 * u * (rEarth^2) * X(1) * (5*(X(3)^2)-(r^2)))/(2*r^7));
f_y = J2*((3 * u * (rEarth^2) * X(2) * (5*(X(3)^2)-(r^2)))/(2*r^7));
f_z = J2*((3 * u * (rEarth^2) * X(3) * (5*(X(3)^2)-(3*r^2)))/(2*r^7));
%concatenate all into one matrix
f_X = [f_x; f_y; f_z];
%final equation
f = A_X*X + B*f_X;
%pass to integration function
[x_hist,y_hist,y0,f_X_hist] = num_int(f,f_X,X0,T,t,C);
%%
%Define RMSO values
syms xhat yhat zhat xdothat ydothat zdothat
initial_error = [.005; .005; .005; .003; .003; .003];
%define initial estimates
Xhat0 = X0 - initial_error;
Xhat1 = [x y z xdothat ydothat zdothat]';
r_1 = sqrt(Xhat1(1)^2 + Xhat1(2)^2 + Xhat1(3)^2);
%define matrices for RMSO
A_RMSO = [0 0 0 1 0 0;
    0 0 0 0 1 0;
    0 0 0 0 0 1;
    -u/(r_1^3) 0 0 0 0 0;
    0 -u/(r_1^3) 0 0 0 0;
    0 0 -u/(r_1^3) 0 0 0];

%basis function vector
phi = [1 xhat/rEarth yhat/rEarth zhat/rEarth xhat*yhat/(rEarth^2)...
     xhat*zhat/(rEarth^2) yhat*zhat/(rEarth^2)]';

K = [0.0012 0 0;
    0 0.0012 0;
    0 0 0.0012;
    0.000608 0 0;
    0 0.000608 0;
    0 0 0.000608];
%original F values, the following are learning rates that I have tuned
% F = diag([0.0000129 0.0000301 0.0000172 0.0000301 0.0000215 ...
%     0.0000129 0.0000129]);
F = diag([0.0000060 0.0000150 0.0000085 0.0000150 0.0000112 ...
    0.0000060 0.0000060]);
%other constants for RMSO
D = 1e-3;
emax = 1e-7;
%original sigma
```

```
% sigma = .02;
sigma = 2;

%implement RMSO
[xhat_hist,est_hist,W_hist] = RMSO(x_hist,y_hist,Xhat0,X0,y0,T,A_RMSO,B,K,C,D,F,sigma,↙
emax,phi);
%%
t1=t/3600;
T1=t1/length(0:T:t-T);
%Plots
%Plot of just the truth model
%x
figure()
subplot(3,1,1)
plot(0:T1:t1,x_hist(1,:))
title('Truth Model X estimate vs. Time')
xlabel('Time (hours)')
ylabel('Position in X axis (KM)')
%y
subplot(3,1,2)
plot(0:T1:t1,x_hist(2,:))
title('Truth Model Y estimate vs. Time')
xlabel('Time (hours)')
ylabel('Position in Y axis (KM)')
%z
subplot(3,1,3)
plot(0:T1:t1,x_hist(3,:))
title('Truth Model Z estimate vs. Time')
xlabel('Time (hours)')
ylabel('Position in Z axis (KM)')
%velocities
%Vx
figure()
subplot(3,1,1)
plot(0:T1:t1,x_hist(4,:))
title('Truth Model X vel estimate vs. Time')
xlabel('Time (hours)')
ylabel('Velocity in X axis (KM/s)')
%Vy
subplot(3,1,2)
plot(0:T1:t1,x_hist(5,:))
title('Truth Model Y vel estimate vs. Time')
xlabel('Time (hours)')
ylabel('Velocity in Y axis (KM/s)')
%Vz
subplot(3,1,3)
plot(0:T1:t1,x_hist(6,:))
title('Truth Model Z vel estimate vs. Time')
xlabel('Time (hours)')
ylabel('Velocity in Z axis (KM/s)')
%truth model J2 acceleration
```

```
figure()
%J2 accel in x
subplot(3,1,1)
plot(0:T1:t1-T1,f_X_hist(1,:))
title('J2 Truth X vs. Time')
xlabel('Time (hours)')
ylabel('J2 acceleration (Km/s^2)')
%J2 accel in y
subplot(3,1,2)
plot(0:T1:t1-T1,f_X_hist(2,:))
title('J2 Truth Y vs. Time')
xlabel('Time (hours)')
ylabel('J2 acceleration (Km/s^2)')
%J2 accel in z
subplot(3,1,3)
plot(0:T1:t1-T1,f_X_hist(3,:))
title('J2 Truth Z vs. Time')
xlabel('Time (hours)')
ylabel('J2 acceleration (Km/s^2)')
%%
%plot of the true and estimated positions
figure()
%x
subplot(3,1,1)
hold on
plot(0:T1:t1,x_hist(1,:))
plot(0:T1:t1,xhat_hist(1,:))
hold off
title('Actual X position and RMSO estimate vs. Time')
xlabel('Time (hours)')
ylabel('Position in X axis (KM)')
legend('Actual','Estimate')
%y
subplot(3,1,2)
hold on
plot(0:T1:t1,x_hist(2,:))
plot(0:T1:t1,xhat_hist(2,:))
hold off
title('Actual Y position and RMSO estimate vs. Time')
xlabel('Time (hours)')
ylabel('Position in Y axis (KM)')
legend('Actual','Estimate')
%z
subplot(3,1,3)
hold on
plot(0:T1:t1,x_hist(3,:))
plot(0:T1:t1,xhat_hist(3,:))
hold off
title('Actual Z position and RMSO estimate vs. Time')
xlabel('Time (hours)')
ylabel('Position in Z axis (KM)')
```

```
legend('Actual','Estimate')
%plots of the true and estimate velocities
figure()
%Vx
subplot(3,1,1)
hold on
plot(0:T1:t1,x_hist(4,:))
plot(0:T1:t1,xhat_hist(4,:))
hold off
title('Actual X Velocity and RMSO estimate vs. Time')
xlabel('Time (hours)')
ylabel('Velocity in X axis (KM/hr)')
legend('Actual','Estimate')
%Vy
subplot(3,1,2)
hold on
plot(0:T1:t1,x_hist(5,:))
plot(0:T1:t1,xhat_hist(5,:))
hold off
title('Actual Y Velocity and RMSO estimate vs. Time')
xlabel('Time (hours)')
ylabel('Velocity in Y axis (KM/hr)')
legend('Actual','Estimate')
%Vz
subplot(3,1,3)
hold on
plot(0:T1:t1,x_hist(6,:))
plot(0:T1:t1,xhat_hist(6,:))
hold off
title('Actual Z Velocity and RMSO estimate vs. Time')
xlabel('Time (hours)')
ylabel('Velocity in Z axis (KM/hr)')
legend('Actual','Estimate')
%%
%State estimation error
est_error = x_hist-xhat_hist;
%plot position error
figure()
%x
subplot(3,1,1)
plot(0:T1:t1,est_error(1,:));
title('Estimation error in X position vs Time')
xlabel('Time (hours)')
ylabel('Estimation error (KM)')
%y
subplot(3,1,2)
plot(0:T1:t1,est_error(2,:));
title('Estimation error in Y position vs Time')
xlabel('Time (hours)')
ylabel('Estimation error (KM)')
%z
```

```
subplot(3,1,3)
plot(0:T1:t1,est_error(3,:));
title('Estimation error in Z position vs Time')
xlabel('Time (hours)')
ylabel('Estimation error (KM)')
%velocity estimation error
figure()
%Vx
subplot(3,1,1)
plot(0:T1:t1,est_error(4,:))
title('Estimation error in X velocity vs. Time')
xlabel('Time (hours)')
ylabel('Estimation Error (KM/hr)')
%Vy
subplot(3,1,2)
plot(0:T1:t1,est_error(5,:))
title('Estimation error in Y velocity vs. Time')
xlabel('Time (hours)')
ylabel('Estimation Error (KM/hr)')
%Vz
subplot(3,1,3)
plot(0:T1:t1,est_error(6,:))
title('Estimation error in Z velocity vs. Time')
xlabel('Time (hours)')
ylabel('Estimation Error (KM/hr)')
%%
%J2 acceleration plots
figure()
%J2 accel in x
subplot(3,1,1)
hold on
plot(0:T1:t1-T1,f_X_hist(1,:))
plot(0:T1:t1-T1,est_hist(1,:))
hold off
title('J2 acceleration in X Actual and Estimate vs. Time')
xlabel('Time (hours)')
ylabel('J2 acceleration (Km/s^2)')
legend('Actual','Estimate')
%J2 accel in y
subplot(3,1,2)
hold on
plot(0:T1:t1-T1,f_X_hist(2,:))
plot(0:T1:t1-T1,est_hist(2,:))
hold off
title('J2 acceleration in Y Actual and Estimate vs. Time')
xlabel('Time (hours)')
ylabel('J2 acceleration (Km/s^2)')
legend('Actual','Estimate')
%J2 accel in z
subplot(3,1,3)
hold on
```

```
plot(0:T1:t1-T1,f_X_hist(3,:))
plot(0:T1:t1-T1,est_hist(3,:))
hold off
title('J2 acceleration in Z Actual and Estimate vs. Time')
xlabel('Time (hours)')
ylabel('J2 acceleration (Km/s^2)')
legend('Actual','Estimate')
%J2 acceleration estimation error
unc_est_error=f_X_hist-est_hist;
figure()
%Accel error in x
subplot(3,1,1)
plot(0:T1:t1-T1,unc_est_error(1,:))
title('J2 acceleration in X estimation error vs. Time')
xlabel('Time (hours)')
ylabel('Estimation Error (Km/s^2)')
%Accel error in y
subplot(3,1,2)
plot(0:T1:t1-T1,unc_est_error(2,:))
title('J2 acceleration in Y estimation error vs. Time')
xlabel('Time (hours)')
ylabel('Estimation Error (Km/s^2)')
%Accel error in z
subplot(3,1,3)
plot(0:T1:t1-T1,unc_est_error(3,:))
title('J2 acceleration in Z estimation error vs. Time')
xlabel('Time (hours)')
ylabel('Estimation Error (Km/s^2)')
>>
```

```
>> type RMSO.m

function [Xhat_hist,est_hist,W_hist] = RMSO(X_hist,Y_hist,Xhat0,X0,Y0,T,A,B,...
    K,C,D,F,sigma,emax,phi)
%This matlab function approximates a function or state history using a reduced
%order modified state observer.The following are the required inputs
%state or function value history (with noise included)
%measurement history (with noise included)
%initial conditions
%time step
%A and B matrices, use xhat1 as the symbolic variable
%MSO gain
%C matrix, assumed scalar
%D, tuning parameter
%F tuning matrix
%sigma, tuning parameter
%emax, tuning parameter

%get number of steps
[r,c]=size(X_hist);  %assuming each state is a column vector
num_steps=c;

%create storage for important variable histories
Xhat_hist=zeros(r,num_steps);
est_hist=zeros(3,num_steps-1);
W_hist = zeros(7,3,num_steps);

%RMSO loop
for i = 1:num_steps
    %initialization
    if i==1
        %Update states and estimates
        %Get measurement at time t=i-1
        Y=Y0;
        %initial weights = 0
        W0=zeros(7,3);
        %assign measured states to variables
        x = Y(1);
        y = Y(2);
        z = Y(3);
        %assign initial estimates to variables
        Xhat=Xhat0;
        xhat = Xhat(1);
        yhat = Xhat(2);
        zhat = Xhat(3);
        xhatdot = Xhat(4);
        yhatdot = Xhat(5);
        zhatdot = Xhat(6);
        %define values at t=i-1
        %A matrix
        X1hat=[x; y; z; xhatdot; yhatdot; zhatdot];
```

```
        A_val=eval(A);
        %robustifying term
        denom = sqrt((Y(1)-Xhat(1))^2 + (Y(2)-Xhat(2))^2 + ...
            (Y(3)-Xhat(3))^2);
        v = -D*((Y-C*Xhat) / denom) - emax*(Y-C*Xhat);
        %evaluate basis function vector
        phi_val=eval(phi);
        %calculate uncertainty estimate at time t=i-1
        f=W0'*phi_val;
        %update values at t=i
        %update weights
        W = W0 + T*F*(phi_val*(Y - C*Xhat)' - sigma*W0);
        %update estimates (note that f: t=i-t, v: t=i-1)
        Xhat = Xhat + T*(A_val*X1hat + B*(f-v) + K*(Y-C*Xhat));
        %store values at t=i
        Xhat_hist(:,i)=Xhat;
        W_hist(:,:,i) = W;
    else
        %general case
        %update states and estimates
        %get measurement at t=i-1
        Y=Y_hist(:,i-1);
        %assign measured states to variables
        x = Y(1);
        y = Y(2);
        z = Y(3);
        %assign estimated states to variables
        xhat = Xhat(1);
        yhat = Xhat(2);
        zhat = Xhat(3);
        xhatdot = Xhat(4);
        yhatdot = Xhat(5);
        zhatdot = Xhat(6);
        %define values at t=i-1
        %A matrix
        X1hat=[x; y; z; xhatdot; yhatdot; zhatdot];
        A_val=eval(A);  %calculate A using previously defined variables
        %robustifying term
        denom = sqrt((Y(1)-Xhat(1))^2 + (Y(2)-Xhat(2))^2 + ...
            (Y(3)-Xhat(3))^2);
        v= -D*((Y-C*Xhat) / denom) - emax*(Y-C*Xhat);
        %calculate basis function values
        phi_val=eval(phi);
        %calculate uncertainty estimate term
        f=W'*phi_val;
        %update values at t=i
        W = W + T*F*(phi_val*(Y - C*Xhat)' - sigma*W);
        Xhat = Xhat + T*(A_val*X1hat + B*(f-v) + K*(Y-C*Xhat));
        %store values
        Xhat_hist(:,i)=Xhat;
        est_hist(:,i-1)=f;
```

```
    end
end
end

>>
```

```
>> type num_int.m

function [X_hist,Y_hist,Y0,f_X_hist] = num_int(f,f_X,X0,T,t,C)
%This is a numerical integration function designed to receive a symbolic
%function f, an initial condition x0, a step size T, and a time value t.
%This function integrates the symbolic function f from 0 to t given the
%initial conditions and step size value
%This function is designed to be used with filters or other estimation
%algorithms and thus adds random noise to the state. It also returns a
%measurement with added noise

%determine number of integration steps
num_steps=length(0:T:t);

%get dimensions of c
[meas, ~]=size(C);

%create storage for important variables
%state history
X_hist=zeros(length(X0),num_steps);
%measurement history
Y_hist=zeros(meas,num_steps);
%J2 effect matrix
f_X_hist=zeros(3,num_steps-1);

%initial measurement
Y0=C*X0;

%perform integration
for i =1:num_steps
    %initialization
    if i==1
        %generate values at t=i-1
        %assign state values
        x = X0(1);
        y = X0(2);
        z = X0(3);
        xdot = X0(4);
        ydot = X0(5);
        zdot = X0(6);
        %calculate equation values
        f_val=eval(f);
        %update variables at time t=i
        X=X0 + T*(f_val);
        Y=C*X0;
        %store variables
        X_hist(:,i)=X;
        Y_hist(:,i)=Y;
    else
        %general
        %generate values at t=i-1
```

```
        %assign state values
        x = X(1);
        y = X(2);
        z = X(3);
        xdot = X(4);
        ydot = X(5);
        zdot = X(6);
        %calculate values
        f_val=eval(f);
        f_X_val=eval(f_X);
        %update values at t=i
        X=X + T*(f_val);
        Y=C*X;
        %store variables
        X_hist(:,i)=X;
        Y_hist(:,i)=Y;
        f_X_hist(:,i-1)=f_X_val;
    end
end
end

>>
```