ASEN 5070

Homework 5

Zach Dischner

10/2/2012

1.0 Single Plane Orbit Problem

In a planar orbit system, the equations of motion governing the system dynamics are:

$$\ddot{x} = = -\frac{x}{r^3} \quad (1)$$

$$\ddot{y} = = -\frac{y}{r^3} \quad (2)$$

$$r = x^2 + y^2 \quad (3)$$

a)

First, I generated a 'true' solution for this planar orbit system by integrating the above equations of motion in MATLAB using the initial conditions provided in the assignment. This 'true' solution, **X**, was saved once every 10 seconds, for 100 seconds. See *Appendix A* for a copy of the code used for this integration. This integration was performed with MATLAB's *ode45* with a tolerance set to **1e-15**.

b)

Next, I perturbed the initial conditions for this integration, resulting in a new trajectory (**X***), along with a state transition matrix (Φ) for each time step. After perturbing the initial conditions, I found a new trajectory for each time step. Below is a comparison between three terms between what I found and what was given as a solution.

Table 1: Solution Comparison

Term	My solution	Given Solution
$X(t_{100})$	0.862318872280193 -0.5063656411229 0.506365641122773 0.862318872279933	0.862318872290 -0.506365641106 0.506365641106 0.862318872290
$X^*(t_{100})$	0.862623359655056 -0.505843963218956 0.505845689229311 0.862623303039941	0.862623359658 -0.505843963213 0.505845689224 0.862623303043
$X(t_{100}) - X^*(t_{100})$	-0.000304487374862772 -0.000521677903944417 0.000519951893462101 -0.000304430760008412	0.000000158340 -0.000000088813 0.000000091051 0.000000158087

Seen in the table above, the differences in my solutions and the given ones are minute. Mostly, they can be attributed to numerical precision of the computer used and the algorithmic methods used for integration.

I also compared my state transition matrix at t=100 to that given by the solutions. For convenience, I provide only the difference between the two below.

Table 2: Φ *Comparison*

Φ_determined - Φ_solution			
-1.46664547173714e-09	8.49521841761458e-12	-2.40141240226421e-13	-1.57288582158799e-09
1.28801502796705e-09	6.13054051967765e-12	3.98484023556023e-12	1.11793951873551e-09
-1.34463107315241e-09	-2.49300580179579e-12	-7.94253551816837e-13	-1.16415321826935e-09
-1.42415501613868e-09	5.07704989161084e-13	-7.24881266123134e-12	-1.53451651385694e-09

Again, the computed and given state transition matrix are very close. At most, they differ by around 1e-9.

c)

Next, I had to verify that Φ was symplectic. Even though this could be done by simply multiplying it by its inverse, I followed equation 4.2.22 in the text. See *MATLAB* code included in Appendix A. The result is shown in the table below.

Table 3: Φ Symplecticness

	$-(\boldsymbol{J} * \boldsymbol{\Phi}(\boldsymbol{t}, \boldsymbol{t_{100}}) * \boldsymbol{J})^{\boldsymbol{T}}$		**EQ(4.2.22)**
0.999999999981895	4.48139303443895e-11	-5.32907051820075e-15	-6.80415723763872e-11
7.95239429862704e-11	1.00000000001214	6.80522305174236e-11	0
0	-5.51239054402686e-11	0.999999999981906	7.95239429862704e-11
5.5116800012911e-11	0	4.48139303443895e-11	1.00000000001214

The result is very close to the identity matrix, as equation the problem statement stipulates. Any discrepancies can again be attributed to numerical integration errors.

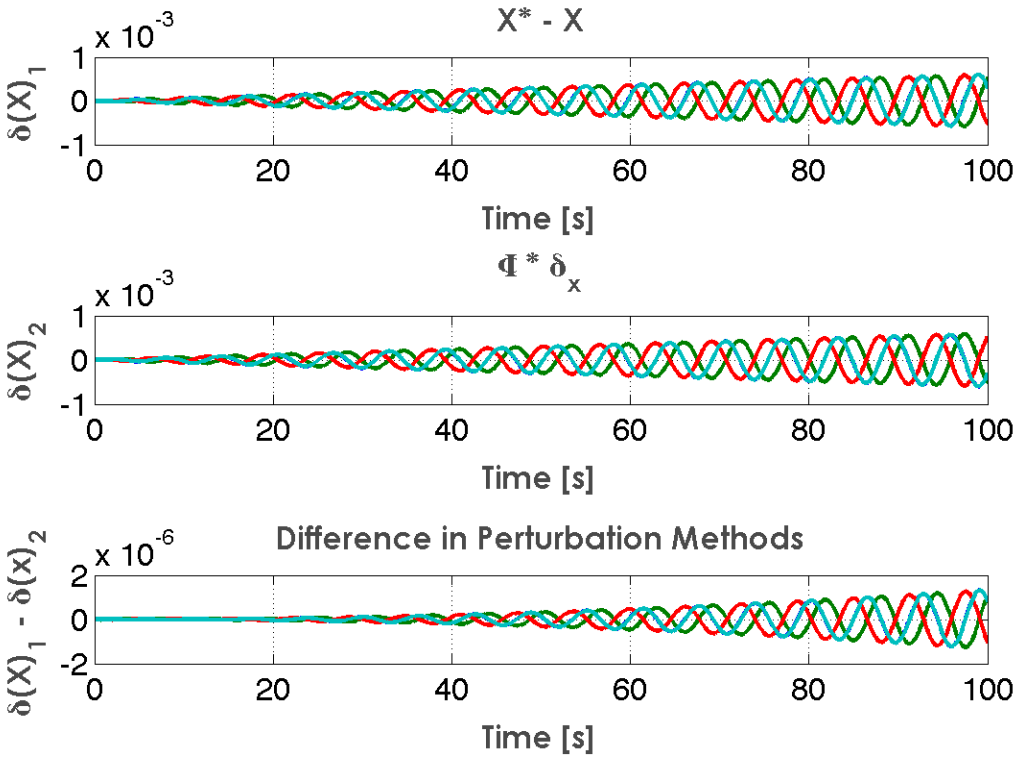
d)

Next, the perturbation vector (δX) was calculated via two methods, shown below.

$$\delta X_1(t_i) = X(t_i) - X^*(t_i) \quad (4)$$

$$\delta X_2(t_i) = \Phi(t_i, t_0)\delta X(t_0) \quad (5)$$

To best compare the two methods, I chose to plot them over the entire integration period, with more than just 10 instances. See Figure 1.



The two perturbation calculation methods produced very similar results. Both the shape and magnitude are comparable, both being around 1e-3. As such, the differences in perturbations were very small.

2.0 **Simple Least Squares Problem**

In this problem, I was given the state relation $y = Hx + \epsilon$ and given:

$$y = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix},$$

$$W = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{and}$$

$$H = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

With priori information $\bar{x} = 2$ and $\overline{W} = \ 2$.

a/b) Find $\hat{x}$

This was a simple calculation, using EQ(4.3.25) in the book. The actual calculation was performed in MATLAB, and the reference code can be found in *Appendix A.* At the same time, a calculation for the best estimate for the observation error was performed. This equation can also be found in the text.

$$\hat{x} = (H^T * W * H + \overline{W})^{-1}(H^T * W * y + \overline{W} * \bar{x}) \quad 6$$

$$\hat{\epsilon} = y - H * \hat{x} \quad 7$$

The results for the above equations are provided in the following table.

Table 4: Problem 2 Solution

$\hat{x}$	**1.500**
$\hat{\epsilon}$	**-0.5** **0.5** **-0.5**

Problem 3-The Batch Processor

This problem asked for basically a re-write of example 4.2.8 using the different observation data, provided with the problem and online. Again, this was all performed in *MATLAB*, so see the attached code. Basically, I followed the example and the batch processor algorithm laid out in the text. The only difference with the outline and my code is that I waited until the end of each iteration's processing to add the covariance matrix, where the outline suggests doing it before hand. I also performed the entire integration, across all time steps, in one single step. I felt that this would be faster, and would yield no less accuracy than doing separate integrations due to the inner workings of *MATLAB*'s numerical integrators.

At the end of the batch processor, I ended up with the exact same solutions as was provided in the book, which I took to be a good sign. See the solutions table below for the values I determined. The statistical values were determined using Appendix A.14 from the text, which discussed various inferences of the variance covariance matrix.

Table 5: Problem 3 Solutions

$[\hat{x}_0 \ \hat{v}_0]$	[2.9571 -0.12602]'
$\rho\ (rms)$	0.24745
$\dot{\rho}\ (rms)$	0.0875
σ_{x_0}	0.044956
σ_{v_0}	0.079449
ρ_{xy}	0.042673

Appendix A – MATLAB code for this assignment

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
% Zach Dischner-9/28/2012
%
% ASEN 5070-Statistical Orbit Determination
%
% Homework 5
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% addpath 'Utils'

clc;clear all;close all; format compact;format long g;
% set(0,'defaultLineLineWidth',
2);set(0,'DefaultAxesXGrid','on','DefaultAxesYGrid','on')
tic

%% Problem 1a-Simple Orbit Solver
%1a - Get true solution
RV_Init = [1.0 0 0 1.0]';
tol     = 1e-9;
tol_mat = ones(size(RV_Init)) .* tol;
time    = [0:0.01:100];

options = odeset('RelTol',tol,'AbsTol',tol_mat);


[t,X]  = ode45('RV_Deriv1a',time,RV_Init,options);

RV_true = X(mod(t,10)==0,:);

%% 1b Perturb previous initial conditions

perturb_init = 1e-6*[1 -1 1 1]';
RV_Init2    = RV_Init - perturb_init;

Phi_Init_mat= eye(size(RV_Init2,1));
[m n]       = size(Phi_Init_mat);

% Reform initial vector of phi.
Phi_Init    = reshape(Phi_Init_mat,m*n,1);

State_Init  = [RV_Init2;Phi_Init];

tol_mat = ones(size(State_Init)) .* tol;
options = odeset('RelTol',tol,'AbsTol',tol_mat);
```

```
[t,Y]   = ode45('RV_Deriv1b',time,State_Init,options);

Xstar    = Y(:,1:length(RV_Init2));
R        = sqrt(Xstar(:,1).^2 + Xstar(:,2).^2);

for ii = 1:length(Y)
    Phi{ii} = reshape(Y(ii,5:20),size(Phi_Init_mat));
end


fprintf('Tolerance used was %3.3e\n',tol)
fprintf('\n\n#^#^#^#^#^#^#^#^#^#^#^#^# Problem 1b
#^#^#^#^#^#^#^#^#^#^#^#^#^#\n\n')
disp(['X(t_100) :                    [',num2str(X(end,:)),']'])
disp(['X*(t_100) :                   [',num2str(Xstar(end,:)),']'])
disp(['X*(t_100) - X*(t_100) :    [',num2str(X(end,:) -
Xstar(end,:)),']'])
disp('\Phi(t_100) :')
Phi{end}


%% 1c - Verify PHI Symplecticness from EQ 4.2.22

Phi_inv = [Phi{end}(3:end,3:end)',  -Phi{end}(1:2,3:end)';...
            -Phi{end}(3:end,1:2)',    Phi{end}(1:2,1:2)'];
fprintf('\n\n#^#^#^#^#^#^#^#^#^#^#^#^# Problem 1c
#^#^#^#^#^#^#^#^#^#^#^#^#^#\n\n')
fprintf('Phi(t_100,t_0) * Phi(t_100,t_0)^-1 is nearly the identity
matrix. Close enough for numerical precision\n')
Phi{end}*Phi_inv


%% 1d - Calculate Perturbation Vector
% Method 1: DELTA(X)i = Xi - X*i

Perturb1 = Xstar - X;

% Method 2: DELTA(X)i = PHI(ti,t0)*(delta??)Xt0
for ii = 1:length(Phi)
    Perturb2(ii,:) = Phi{ii}*perturb_init;
end

figure; grid on

subplot(3,1,1)
plot(time,Perturb1); xlabel('Time [s]');ylabel('\delta(X)_1');title('X*
- X')
subplot(3,1,2)
plot(time,Perturb2); xlabel('Time
[s]');ylabel('\delta(X)_2');title('\Phi * \delta_x')
subplot(3,1,3)
plot(time,Perturb1 + Perturb2); xlabel('Time [s]');ylabel('\delta(X)_1
- \delta(x)_2');title('Difference in Perturbation Methods')
```

```
%% Problem 2     given  y = H*x + e,   x is a scalar
% Observations
y       = [1 2 1]';
% Weighting
W       = [2 0 0;
           0 1 0;
           0 0 1];
% Map between observations and state
H       = [1 1 1]';
% Priori
x_bar   = 2;
W_bar   = 2;

%% 2a - Find "x" Using Batch Processor Algorithm?? Just single point>>

% Eq 4.3.25 in book.
x_caret = inv(H'*W*H + W_bar)*(H'*W*y + W_bar*x_bar);

fprintf('\n\n\n\n#^#^#^#^#^#^#^#^#^#^#^# Problem 2a
#^#^#^#^#^#^#^#^#^#^#^#^#^#^#\n\n')
fprintf('x^    : %3.5f\n\n',x_caret)
%% 2b - Find best estimate for observation error e^

% Eq 4.3.7
e_caret = y - H*x_caret;
fprintf('\n\n#^#^#^#^#^#^#^#^#^#^#^# Problem 2b
#^#^#^#^#^#^#^#^#^#^#^#^#^#^#\n\n')
disp(['e^    :   [',num2str(e_caret'),']'''])



%% Problem 3 - Problem 4.15 From Book
rho = [     6.37687486186586
            5.50318198665912
            5.94513302809067
            6.30210798411686
            5.19084347133671
            6.31368240334678
            5.80399842220377
            5.45115048359871
            5.91089305965839
            5.67697312013520
            5.25263404969825];

rho_dot = [ -0.00317546143535849
             1.17587430814596
            -1.47058865193489
             0.489030779000695
             0.993054430595876
            -1.40470245576321
             0.939807575607138
```

```
     0.425908088320457
    -1.47604467619908
     1.42173765213734
    -0.12082311844776];
sigma_rho       = 0.25;
sigma_rhodot    = 0.1;

R       = [ 0.0625   0;    % R = E(e*e') = R = W^-1
            0   0.01;
    ];



Phi_Init= eye(2,2);

DelXhat0 = [0 0];

DelXbar0 = [0 ; 0];


% Priori Reference Trajectory
Xstar0  = [4.0 ; 0.2];

XV_Init = Xstar0;       % [ m ; m/s ]   [x v]

for jj = 1:4




    State_Init  = [XV_Init ; reshape(Phi_Init,4,1)];

    % Priori <something>
    Pbar0   = [1000 0 ; 0 100];

    % System dynamics
    k1          = 2.5;  % N/m
    k2          = 3.7;  % N/m
    m           = 1.5;  % Kg
    w_squared   = (k1 + k2)/m;
    h           = 5.4;  % m


    %% Perform dynamical integration
    tol     = 1e-14;
    time    = [0:1:10];
    tol_mat = ones(size(State_Init)) .* tol;
    options = odeset('RelTol',tol,'AbsTol',tol_mat);

    [t,X]      = ode45('SpringDeriv',time,State_Init,options);

    Xstar = X(:,1:length(XV_Init));
```

```
            for ii = 1:length(X)
                Phi3{ii} = reshape(X(ii,3:6),size(Phi_Init));
            end

            x = X(:,1); v = X(:,2);
            sumHH = [0 0;0 0];
            sumHy = [0;0];



            for ii = 1:length(rho)


                RhoStar(ii)  = sqrt(x(ii)^2 + h^2);

                Htilde   = [                 x(ii)/RhoStar(ii)
0;...
                    (v(ii)/RhoStar(ii) - x(ii)^2*v(ii)/(RhoStar(ii)^3))
x(ii)/RhoStar(ii) ...
                    ];

                H3{ii}    = Htilde*Phi3{ii};

                sumHH    = sumHH + H3{ii}'*inv(R)*H3{ii};

                G_Star   = [RhoStar(ii) ; x(ii)*v(ii)/RhoStar(ii)];

                y3{ii}    = [rho(ii) rho_dot(ii)]' - G_Star;
                sumHy    = sumHy + H3{ii}'*inv(R)*y3{ii};

            end

            DelXhat0 = inv(sumHH + inv(Pbar0))*(sumHy +
inv(Pbar0)*DelXbar0);

            DelXbar0 = DelXbar0 - DelXhat0;

            XV_Init = XV_Init + DelXhat0;

        end
        format long G

        fprintf('\n\n\n\n#^#^#^#^#^#^#^#^#^#^#^# Problem 3
#^#^#^#^#^#^#^#^#^#^#^#^#^#\n\n')
disp(['X*0    :   [',num2str(XV_Init'),']'''])


rho_rms = rms(RhoStar' - rho);
rho_dot_rms = rms(rho_dot - x.*v./RhoStar');


disp(['rho_rms    :',num2str(rho_rms)])
```

```
disp(['rho_dot_rms    :',num2str(rho_dot_rms)])


% Final Covariance Matrix
P = inv(sumHH + inv(Pbar0));

sigmaX = P(1,1)^0.5;
sigmaY = P(2,2)^0.5;

disp(['Sigma_X0   :',num2str(sigmaX)])
disp(['Sigma_Y0   :',num2str(sigmaY)])

disp(['rho_xy   :',num2str(P(1,2)/(sigmaX*sigmaY))])

fprintf('\n\n\n\nAssignement took %3.3f seconds to run\n\n\n\n\n',toc)
```

```
function [yprime] = RV_Deriv1a(time,y)

% Simple function to pass to integrator for HW5, problem 1.
% yprime = d(y)/dt =

% where y = [x y x' y']
%
% so yprime = [x' y' x'' y'']
%
% where x'' = -x/r^3;  y'' = -y/r^3;

u = 1;

r = sqrt(y(1).^2 + y(2).^2);

yprime = [y(3) y(4) -y(1)/r^3 -y(2)/r^3 ]';

end
```

```
function [yprime] = RV_Deriv1b(time,y)

% Simple function to pass to integrator for HW5, problem 1b.
% yprime = d(y)/dt =

% where y = [x y x' y' PHI]
%
% so yprime = [x' y' x'' y'' PHI']
%
% where x'' = -x/r^3;  y'' = -y/r^3; phi' = A*phi

u = 1;



r = sqrt(y(1).^2 + y(2).^2);

A = [ 1     0       0       0
      0     1       0       0
      0     0    -1./r.^3   0;
      0     0       0    -1./r.^3;
    ];

A = [               0                         0                         1
0;
                    0                         0                         0
1;
    -u./(r.^3)+3*u.*y(1).^2./(r.^5)   3*u*y(1)*y(2)/(r^5)           0
0;
            3*u*y(1)*y(2)/(r^5)    -u./(r.^3)+3*u.*y(2).^2./(r.^5)  0
0;
    ];


% lumping together RV' = A*y     and PHI' = A*PHI

% PHI = reshape(y(5:20),4,4)

yprime = [[y(3) y(4) -y(1)/r^3 -y(2)/r^3 ]'; reshape( A *
reshape(y(5:20),4,4), 16,1)];

end
```

```
function [yprime] = SpringDeriv(time,y)

% Simple function to pass to integrator for HW5, problem 3  => 4.15 in
book.
% yprime = d(y)/dt =

% where y = [x v PHI]
%
% so yprime = [    0    1       [x
%                -w^2   0]       v]     ; A*Phi
%
%
% where w^2 = (k1 + k2)/m

k1          = 2.5;  % N/m
k2          = 3.7;  % N/m
m           = 1.5;  % Kg
w_squared   = (k1 + k2)/m;

A           = [ 0 1; -w_squared 0];

yprime =[ [0 1; -w_squared 0]*y(1:2); reshape( A * reshape(y(3:6),2,2),
4,1)];




end
```