Homework #7 – ASEN 5050

Due: Thursday, 11/5/2015

Name: John Clouse

This problem set examines the difference between propagating a state using your analytic two-body equations (i.e., convert to orbital elements, advance time, and output Cartesian coordinates for each time) *versus* using a numerical integrator. Assume two-body dynamics only, with Earth as the central body. Answer the questions given below.

Note 1: This homework requires numerical integration, but you do not need to write your own numerical integrator. If you're using Matlab you can use the function "ode45" (example code at the bottom of this homework). If all else fails, you are welcome to code up an RK4 integrator or download one in your language of choice. This may take some time to work out so start early!

Note 2: If you're using a variable time-step integrator (like ode45), set the tolerance to 1e-12 or better (unless otherwise noted). If you're using a fixed time-step integrator (like a very basic RK4), then set the time-step to 0.1 sec (unless otherwise noted).

Problem Statement

The ECI position and velocity of the Cygnus vehicle has been determined to be:

X = 5492.000 km
Y = 3984.001 km
Z = 2.955 km
VX = -3.931 km/sec
VY = 5.498 km/sec
VZ = 3.665 km/sec

Use a GM value for the Earth of 398,600.4415 km^3/s^2

1. Use your analytic two-body propagator (not the integrator, but the code we developed in previous homeworks!) to determine the X, Y, and Z positions of Cygnus at a time 100 seconds into the future. Do this again at a time 1,000,000 seconds into the future.

	100 seconds later	**1,000,000 seconds later**
X (km)	5064.753117 km	1407.037338 km
Y (km)	4507.257422 km	6270.084687 km
Z (km)	368.658030 km	2306.266073 km

2. Now use a numerical integrator to do the same (ode45 or the like). Populate the table given below, as was done for part (1). You should not find the answers to be identical with part (1), and you should expect the 1,000,000 sec state to be more different than the 100 sec state.

	100 seconds later	**1,000,000 seconds later**
X (km)	5064.753168 km	1407.037860 km
Y (km)	4507.257367 km	6270.084634 km
Z (km)	368.657989 km	2306.265884 km

3. Use the analytic results in part (1) as **truth** and compare the *magnitude of the position difference* for different numerical integrations **experiments**. Run your integrator on the 1,000,000-second integration several times for different tolerance values. If you're using a variable time-step integrator like ode45, then edit the tolerance of the integration to be between values of 1e-4 and 1e-12. If you're using a fixed time-step integrator, like RK4, then change the fixed time-step to be values between 0.01 sec and 100 sec. If you're using something else, then stop that and go use either an integrator like ode45 or a fixed time-step integrator! ☺ (just for now)

For each case, compute the vector position difference between the truth and the experiment; then take the magnitude of that position difference and record that. I.e, compute: $\Delta R = \sqrt{\left[\left(x_{experiment} - x_{conic}\right)^2 + \left(y_{exp} - y_{conic}\right)^2 + \left(z_{exp} - z_{conic}\right)^2\right]}$

Fill in the appropriate values in the table below. Please use scientific notation so that we can compare the exponents in the difference magnitudes. You only have to complete one column:

	1,000,000-second integration **ode45 or other variable time-step integrator**	**1,000,000-second integration** **RK4 or other fixed time-step integrator**
ΔR (km)	Tol = 1e-12 5.566436e-04 km	Δt = 0.01 sec
ΔR (km)	Tol = 1e-10 0.043311 km	Δt = 0.1 sec
ΔR (km)	Tol = 1e-8 3.948961 km	Δt = 1 sec
ΔR (km)	Tol = 1e-6 31.540622 km	Δt = 10 sec
ΔR (km)	Tol = 1e-4 1.204741e+04 km	Δt = 100 sec

Numerical Integration in Matlab

Numerical integration of the two-body problem in Matlab can be accomplished by defining a function "two-body" as:

```
function xdot=two_body(t,X)
Mu   = 3.986004415e5;
r    = norm(X(1:3));
xdot = [ X(4); X(5); X(6); -Mu*X(1)/r^3; -Mu*X(2)/r^3;
-Mu*X(3)/r^3];
```

and then your main program would look like:

```
%HW 7
% Script to Numerically Integrate the two-body problem
%
clear
close all

%Solve set of differential equations in the ECI frame

time=[0:10:14400];
RO=[5492.0 3984.001 2.955];
VO=[-3.931 5.498 3.665];
XO=[RO VO];

tol=1e-12;
options=odeset('RelTol',tol,'AbsTol',[tol tol tol tol tol
tol]);

%ode45 matlab integrator - type "help ode45"

[t,X]=ode45('two_body',time,XO,options);
figure
plot(t,X(:,1),'X');
title('X vs Time');
ylabel('X (km)');
```

HW7 Problem 1

```
fprintf('\n');
clearvars -except function_list hw_pub toolsPath
close all
CelestialConstants; % import useful constants

X0 = [5492.000;%km
 3984.001 ;%km
 2.955 ;%km
 -3.931 ;%km/sec
 5.498 ;%km/sec
 3.665 ];%km/sec

% Anon fcn to calculate specific energy. It shouldn't change!
spec_energy = @(X) norm(X(4:6))^2/2 - Earth.mu/norm(X(1:3));

% Classical orbit elements
[a,e,i,RAAN,w,f] = cart2OE(X0(1:3),X0(4:6),Earth.mu);

% Get the stuff that's propagated
n = sqrt(Earth.mu/a/a/a);
M0 = E2M(f2E(f,e),e);

for t = [100 1e6]; %s
    % Final mean anom is easy...
    Mf = M0 + n*t;
    % Unwinde the mean anom
    while Mf > 2*pi
        Mf = Mf - 2*pi;
    end
    % Final true anom
    ff = E2f(M2E(Mf,e),e);
    % Back to ECI!
    [r_f, v_f] = OE2cart(a,e,i,RAAN,w,ff,Earth.mu);
    fprintf('r_f(t=%d):\n',t)
    disp(r_f);
    fprintf('delta Energy(t=%d):\n',t)
    disp(spec_energy([r_f;v_f]) - spec_energy(X0));
end

% Anonymous function to calculate 2-body accel
two_body = @(t,X) [X(4);X(5);X(6);...
    -Earth.mu*X(1)/norm(X(1:3))^3;...
    -Earth.mu*X(2)/norm(X(1:3))^3;...
    -Earth.mu*X(3)/norm(X(1:3))^3];

% Anon fcn to calculateposition difference.
calc_dr = @(X_exp, r_f) sqrt((X_exp(1)-r_f(1))^2 ...
        +(X_exp(2)-r_f(2))^2 ...
        +(X_exp(3)-r_f(3))^2);
```

```
tol=1e-12;
options=odeset('RelTol',tol,'AbsTol',[tol tol tol tol tol tol]);
for t = [100 1e6]
    [t_array,X_array]=ode45(two_body,[0 t],X0,options);
    fprintf('r_f(t=%d)(integrated):\n',t_array(end))
    disp(X_array(end,1:3)');
    fprintf('delta Energy(t=%d)(integrated):\n',t)
    disp(spec_energy(X_array(end,1:6)) - spec_energy(X0));
end

for tol = [1e-12 1e-10 1e-8 1e-6 1e-4]
    options=odeset('RelTol',tol,'AbsTol',[tol tol tol tol tol tol]);
    [t_array,X_array]=ode45(two_body,[0 1e6],X0,options);
    fprintf('Position Diff @ tol = %e:\n',tol)
    disp(calc_dr(X_array(end,1:3)',r_f));
    fprintf('delta Energy @ tol = %f:\n',tol)
    disp(spec_energy(X_array(end,1:6)) - spec_energy(X0));
end
```

```
r_f(t=100):
   1.0e+03 *

   5.064753117135560
   4.507257422053243
   0.368658029818315

delta Energy(t=100):
     3.552713678800501e-15

r_f(t=1000000):
   1.0e+03 *

   1.407037337984632
   6.270084686694161
   2.306266072702379

delta Energy(t=1000000):
     1.421085471520200e-14

r_f(t=100)(integrated):
   1.0e+03 *

   5.064753168465420
   4.507257366803862
   0.368657989226936

delta Energy(t=100)(integrated):
     4.973799150320701e-14

r_f(t=1000000)(integrated):
   1.0e+03 *

   1.407037859051284
```

```
   6.270084634401733
   2.306265884003925

delta Energy(t=1000000)(integrated):
     2.198543569420508e-09

Position Diff @ tol = 1.000000e-12:
     5.566435667007229e-04

delta Energy @ tol = 0.000000:
     2.198543569420508e-09

Position Diff @ tol = 1.000000e-10:
   0.043311061412548

delta Energy @ tol = 0.000000:
     2.203521596300107e-07

Position Diff @ tol = 1.000000e-08:
   3.948961410452488

delta Energy @ tol = 0.000000:
     2.020949409953232e-05

Position Diff @ tol = 1.000000e-06:
  31.540622301006042

delta Energy @ tol = 0.000001:
    -1.331167062197380e-04

Position Diff @ tol = 1.000000e-04:
     1.204741406084945e+04

delta Energy @ tol = 0.000100:
  -2.587559250330802
```

Published with MATLAB® R2013b

CelestialConstants

Table of Contents

Description

All sorts of constants for orbital mechanics purposes

```
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app
```

Earth

```
Earth.mu = 3.986004415e5; %km3/s2
Earth.R = 6378; %km
Earth.a = 149598023; %km
Earth.spin_rate = 7.2921158553e-05; %rad/s
Earth.flattening = 1/298.25722; %WGS-84

%%Sun
Sun.mu = 1.32712428e11; %km3/s2
```

Venus

```
Venus.a = 108208601; %km
```

Mars

```
Mars.a = 227939186; %km
```

Jupiter

```
Jupiter.a = 778298361; %km
```

Celestial units

```
au2km = 149597870.7;
```

Physical constants

```
day2sec = 86400; % sec/day
```

Published with MATLAB® R2013b

```
function [a,e,i,RAAN,w,f] = cart2OE( r, v ,mu)
%cart2OE return classical orbital elements from cartesian coords
% Only valid for e < 1
% units in radians
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

h = cross(r,v);
n = cross([0;0;1],h);
ecc_vec = ((norm(v)*norm(v)-mu/norm(r))*r - dot(r,v)*v)/mu;

e = norm(ecc_vec);
a = 0;
if e < 1.0
    specific_energy = norm(v)*norm(v)/2-mu/norm(r);
    a = -mu/2/specific_energy;
end

i = acos(h(3)/norm(h));
RAAN = acos(n(1)/norm(n));
if n(2) < 0
    RAAN = 2*pi-RAAN;
end
w = acos(dot(n,ecc_vec)/(norm(n)*norm(ecc_vec)));
if ecc_vec(3) < 0
    w = 2*pi-w;
end
f = acos(dot(ecc_vec,r)/(norm(ecc_vec)*norm(r)));
if dot(r,v) < 0
    f = 2*pi-f;
end
```

Published with MATLAB® R2013b

```
function E = f2E( f, e )
%f2E True anomaly (f) to eccentric anomaly (E)
% Only valid for e < 1
% units in radians
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

% Vallado eqn 2-9
E = acos((e + cos(f))/(1+e*cos(f)));
if f > pi
    E = 2*pi - E;
end
```

Published with MATLAB® R2013b

```
function M = E2M( E, e )
%E2M Eccentric anomaly (E) to mean anomaly (M)
% units in radians
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

% Vallado eqn 2-4
M = E-e*sin(E);
```

Published with MATLAB® R2013b

```
function E = M2E( M, e )
%M2E Mean anom (M) to eccentric anom (E)
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

tol = 1e-5;

if (M < 0 && M > -pi) || M > pi
    E_1 = M - e;
else
    E_1 = M + e;
end

E = E_1 + tol + 1;
while abs(E_1-E) > tol
    E = E_1;
    E_1 = E - (E - e*sin(E) - M)/(1 - e*cos(E));
end
```

Published with MATLAB® R2013b

```matlab
function f = E2f( E, e )
%E2f Eccentric anomaly (E) to true anomaly (f)
% Only valid for e < 1
% units in radians
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

% Vallado eqn 2-10
f = acos((cos(E) - e)/(1-e*cos(E)));
if E > pi
    f = 2*pi - f;
end
```

Published with MATLAB® R2013b

```
function [r, v ] = OE2cart( a,e,i,RAAN,w,f,mu)
%cart2OE return classical orbital elements from cartesian coords
% Only valid for e < 1
% units in radians
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

% First find r,v in the perifocal coord system.
p = a*(1-e*e);
r_pqw = [p*cos(f);p*sin(f);0]/(1+e*cos(f));
v_pqw = [-sqrt(mu/p)*sin(f); sqrt(mu/p)*(e+cos(f));0];

r = Euler2DCM('313', -[w,i,RAAN])*r_pqw;
v = Euler2DCM('313', -[w,i,RAAN])*v_pqw;
```

Published with MATLAB® R2013b

```
function DCM = Euler2DCM( seq_string, angle_vector )
%Euler2DCM Turn an Euler Angle set into a DCM
%   Angle vector in radians
fcnPrintQueue(mfilename('fullpath'))

DCM = eye(3);
%get the trig functions
num_rot = length(seq_string);
c = zeros(num_rot,1);
s = zeros(num_rot,1);

for idx = 1:num_rot
c(idx) = cos(angle_vector(idx));
s(idx) = sin(angle_vector(idx));
end


for idx = num_rot:-1:1
    if strcmp(seq_string(idx),'1')
        M = [1 0 0; 0 c(idx) s(idx); 0 -s(idx) c(idx)];
        DCM = DCM*M;
    elseif strcmp(seq_string(idx),'2')
        M = [c(idx) 0 -s(idx); 0 1 0; s(idx) 0 c(idx)];
        DCM = DCM*M;
    elseif strcmp(seq_string(idx),'3')
        M = [c(idx) s(idx) 0; -s(idx) c(idx) 0; 0 0 1];
        DCM = DCM*M;
    else
        fprintf('%s is not a valid axis\n', seq_string(idx))
    end
end


end
```

Published with MATLAB® R2013b

```
function fcnPrintQueue( filename )
global function_list;
if exist('function_list', 'var')
    file_in_list = 0;
    for idx = 1:length(function_list)
        if strcmp(function_list(idx), filename);
            file_in_list = 1;
            break
        end
    end
    if ~file_in_list
        function_list = [function_list, filename];
    end
end
end
```

Published with MATLAB® R2013b