HW1 Problem 1: Cartesian Coordinates to Keplerian Orbital Elements

```
fprintf('\n');
clearvars -except function_list pub_opt
close all

r = [-2436.45; -2436.45; 6891.037]; % km
v = [5.088611; -5.088611; 0.0]; % km/s
state = [r;v];
oe = cart2oe(state);
fprintf('a = %f km\n', oe(1))
fprintf('e = %f\n', oe(2))
fprintf('i = %f degrees\n', oe(3)*180/pi)
fprintf('RAAN = %f degrees\n', oe(4)*180/pi)
fprintf('Arg of Periapse = %f degrees\n', oe(5)*180/pi)
fprintf('True Anomaly = %f degrees\n', oe(6)*180/pi)
```

```
a = 7712.194677 km
e = 0.001001
i = 63.434003 degrees
RAAN = 135.000000 degrees
Arg of Periapse = 90.000000 degrees
True Anomaly = 0.000000 degrees
```

Published with MATLAB® R2013b

HW1 Problem 2: Keplerian Orbital Elements to Cartesian Coordinates

```
fprintf('\n');
clearvars -except function_list pub_opt
close all

r = [-2436.45; -2436.45; 6891.037]; % km
v = [5.088611; -5.088611; 0.0]; % km/s
state = [r;v];
oe = cart2oe(state);
new_state = oe2cart(oe);
state_diff = new_state - state;

fprintf('Computed State Vector\n')
new_state
fprintf('Delta between original state vector and computed vector')
state_diff
```

```
Computed State Vector

new_state =

   1.0e+03 *

   -2.4365
   -2.4365
    6.8910
    0.0051
   -0.0051
    0.0000

Delta between original state vector and computed vector
state_diff =

   1.0e-09 *

   -0.1710
   -0.1723
    0.4866
   -0.0004
    0.0004
    0.0000
```

Published with MATLAB® R2013b

John Clouse
ASEN 5070
HW1

Problem 3: given $U = \mu/R$, solve for 2-body accel due to gravity where $R = \sqrt{x^2+y^2+z^2}$

$$\ddot{\vec{r}} = \nabla U = \frac{\partial U}{\partial x}\hat{\imath} + \frac{\partial U}{\partial y}\hat{\jmath} + \frac{\partial U}{\partial z}\hat{k}$$

$$\frac{\partial U}{\partial x} = \frac{\partial(\mu/R)}{\partial x} = \frac{\partial\left(\mu(x^2+y^2+z^2)^{-\frac{1}{2}}\right)}{\partial x}$$

$$= \frac{-\mu(x^2+y^2+z^2)^{-3/2}}{2} \cdot \frac{\partial}{\partial x}(x^2+y^2+z^2)$$

$$= \frac{-\mu(x^2+y^2+z^2)^{-3/2}}{\cancel{2}} \cdot \cancel{2}x = \frac{-\mu x}{R^3}$$

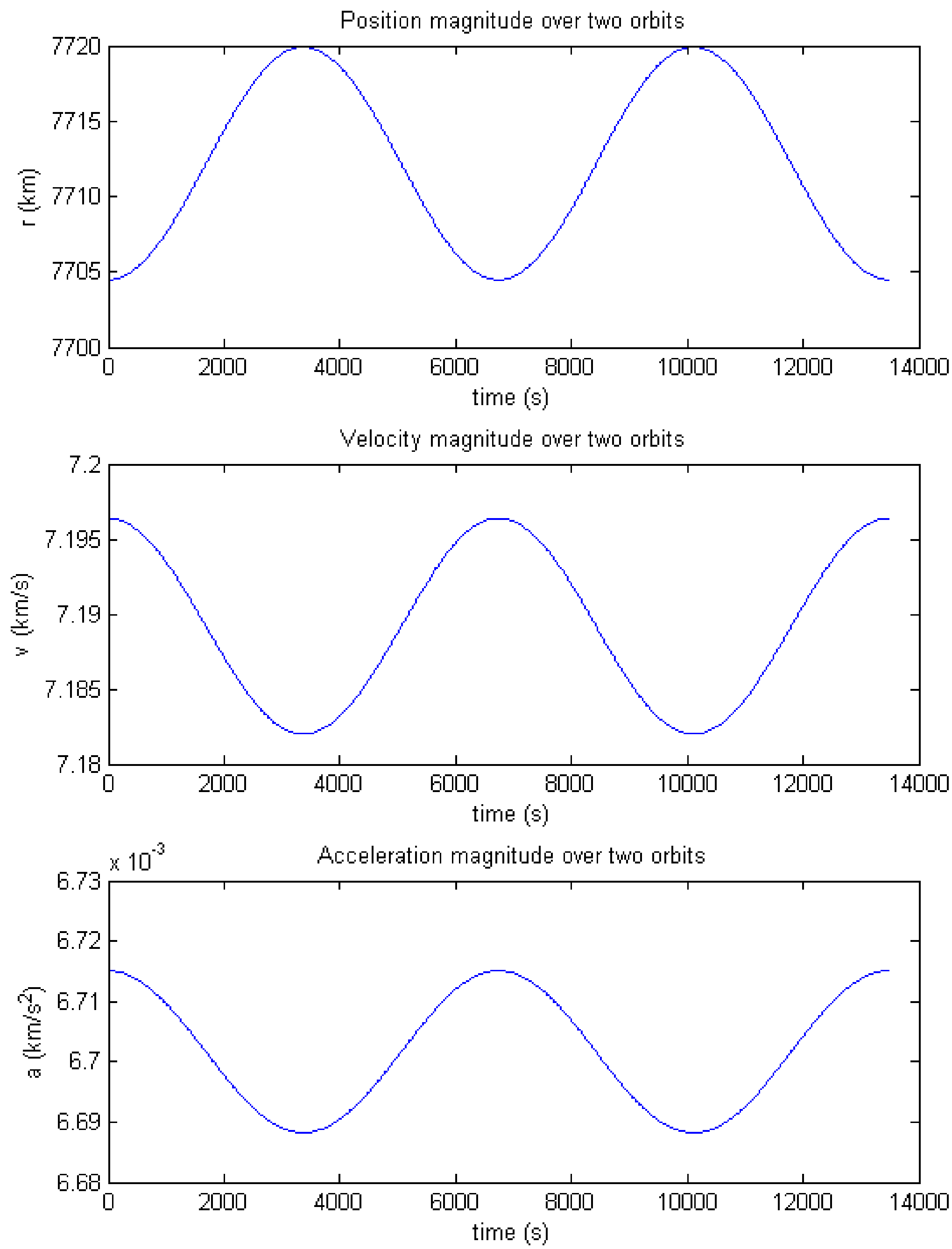
similarly, $\frac{\partial U}{\partial y} = \frac{-\mu y}{R^3}$; $\frac{\partial U}{\partial z} = \frac{-\mu z}{R^3}$

$$\ddot{\vec{r}} = \nabla U = -\frac{\mu}{R^3}(x\hat{\imath} + y\hat{\jmath} + z\hat{k})$$

$$\boxed{\ddot{\vec{r}} = -\frac{\mu}{r^3}\vec{r}}$$ where $r = \|\vec{r}\| = R$

HW1 Problem 4: Orbit Numerical Integration

```
fprintf('\n');
clearvars -except function_list pub_opt
close all

ode_opts = odeset('RelTol', 1e-12, 'AbsTol', 1e-20);
r = [-2436.45; -2436.45; 6891.037]; % km
v = [5.088611; -5.088611; 0.0]; % km/s
state = [r;v];

%Find the period
OE = cart2oe(state);
a = OE(1);
period = 2*pi*sqrt(a*a*a/3.986e5);

times = 0:20:period*2;

[T,X] = ode45(@two_body_state_dot, times, state, ode_opts);

%Get the magnitues for plotting
r_mag = zeros(1,length(times));
v_mag = zeros(1,length(times));
a_mag = zeros(1,length(times));

for i = 1:length(times)
    r_mag(i) = norm(X(i,1:3));
    v_mag(i) = norm(X(i,4:6));
    s_dot = two_body_state_dot(0,X(i,:));
    a_mag(i) = norm(s_dot(4:6));
end

%Plot the result
figHandle = figure;
set(figHandle, 'Position', [100, 100, 600, 800])
subplot(3,1,1);
plot(times, r_mag)
title('Position magnitude over two orbits')
ylabel('r (km)');
xlabel('time (s)');
subplot(3,1,2)
plot(times, v_mag)
title('Velocity magnitude over two orbits')
ylabel('v (km/s)');
xlabel('time (s)');
subplot(3,1,3)
plot(times, a_mag)
title('Acceleration magnitude over two orbits')
ylabel('a (km/s^2)');
xlabel('time (s)');
```

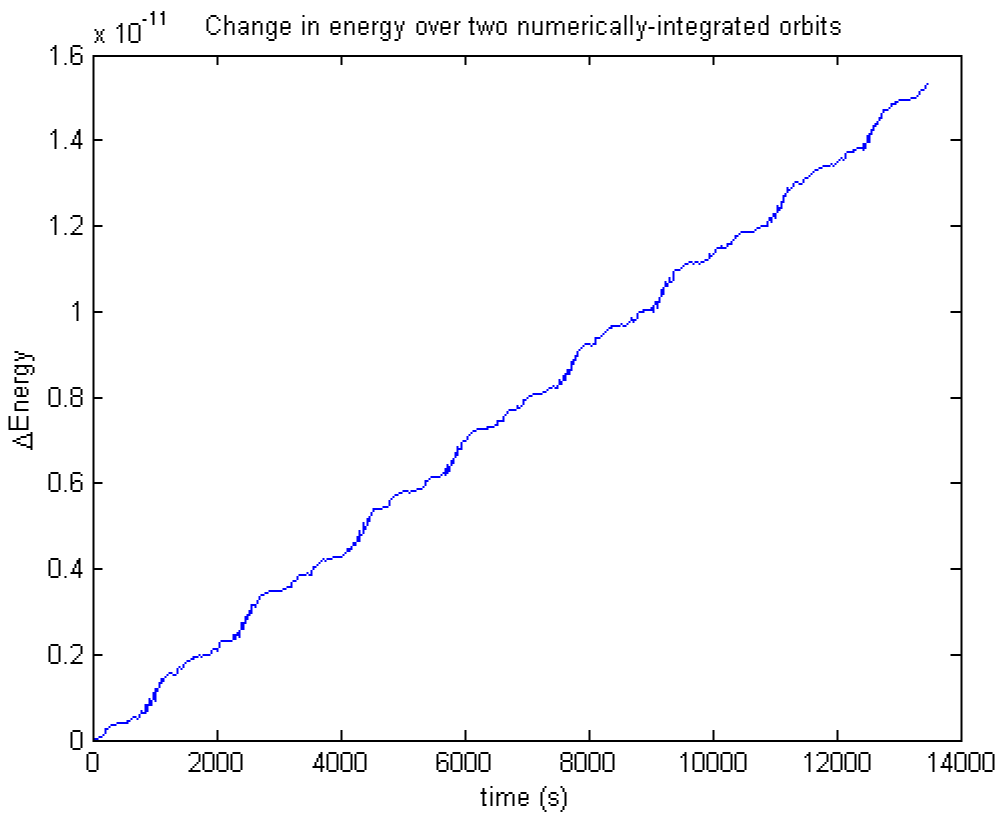


HW1 Problem 5: Orbit Numerical Integration Energy

Why is the change in total specific energy not constant?

The computational precision does not allow for exact calculations. The small error that results is built upon as the simulation runs.

```
KE = v_mag.*v_mag/2;
PE = -3.986e5./r_mag;

deltaE = KE + PE - (KE(1)+PE(1));
figure
plot(times, deltaE)

title('Change in energy over two numerically-integrated orbits')
ylabel('\DeltaEnergy');
xlabel('time (s)');
```



Published with MATLAB® R2013b

HW1 Problem 6: Numerical estimation of initial state

```
fprintf('\n');
clearvars -except function_list pub_opt
close all

state = [1.5
    10
    2.2
    0.5
    0.3];

station_loc = [1.0; 1.0;];

obs = [7.0, 0
    8.00390597, 1
    8.94427191, 2
    9.801147892, 3
    10.630145813, 4];

tol = 1e-6;

delta = 1;
while delta > tol
    jac = compute_cost_fcn_jacobian(obs, state, station_loc);
    cost_fnc = compute_obs_2d(obs, state, station_loc);
    state_est = state - jac\cost_fnc;
    delta = norm(state_est - state);
    state = state_est;
end

fprintf('True Initial State:\n');
state
```

```
True Initial State:

state =

    1.0001
    8.0000
    2.0000
    1.0000
    0.5000
```

Published with MATLAB® R2013b

John Clouse
ASEN 5070
HW1

Problem 6 Supplemental: Finding the partials of the computed range

$$\rho_j = \sqrt{(X_0 - X_s + \dot{X}_0 t)^2 + (Y_0 - Y_s + \dot{Y}_0 t - g\tfrac{t^2}{2})^2}$$

$$\frac{\delta \rho_j}{\delta X_0} = \frac{1}{2} \cdot \frac{1}{\rho_j} \cdot \left(\frac{\delta}{\delta X_0}(X_0 - X_s + \dot{X}_0 t)^2 + \cancelto{0}{\frac{\delta}{\delta X_0}(Y_0 - Y_s + \dot{Y}_0 t - g\tfrac{t^2}{2})^2} \right)$$

$$= \frac{1}{\rho_j} \cdot (X_0 - X_s + \dot{X}_0 t) \cdot \cancelto{1}{\frac{\delta}{\delta X_0}(X_0)}$$

$$\frac{\delta \rho_j}{\delta Y_0} = \frac{1}{\rho_j} \cdot (Y_0 - Y_s + \dot{Y}_0 t - g\tfrac{t^2}{2}) \cancelto{1}{\frac{\delta}{\delta Y_0}(Y_0)}$$

$$\frac{\delta \rho_j}{\delta \dot{X}_0} = \frac{1}{\rho_j} \cdot (X_0 - X_s + \dot{X}_0 t) \cancelto{t}{\frac{\delta}{\delta \dot{X}_0}(\dot{X}_0 t)}$$

$$\frac{\delta \rho_j}{\delta \dot{Y}_0} = \frac{1}{\rho_j} \cdot (Y_0 - Y_s + \dot{Y}_0 t - g\tfrac{t^2}{2}) \cancelto{t}{\frac{\delta}{\delta \dot{Y}_0}(\dot{Y}_0 t)}$$

$$\frac{\delta \rho_j}{\delta g} = \frac{1}{\rho_j} (Y_0 - Y_s + \dot{Y}_0 t - g\tfrac{t^2}{2}) \cancelto{-\frac{t^2}{2}}{\frac{\delta}{\delta g}(-g\tfrac{t^2}{2})}$$

HW 1 Master Script

Table of Contents

Initialize

```
if ispc
    addpath('C:\Users\John\Documents\ASEN5070_SOD\tools')
end
clear all
clc

% Cell array to track what functions are used, so they can be published
% later
global function_list;
function_list = {};

% publishing options
pub_opt.format = 'pdf';
pub_opt.outputDir = '.\html';
pub_opt.imageFormat = 'bmp';
pub_opt.figureSnapMethod = 'entireGUIWindow';
pub_opt.useNewFigure = true ;
pub_opt.maxHeight = Inf;
pub_opt.maxWidth = Inf;
pub_opt.showCode = true;
pub_opt.evalCode = true;
pub_opt.catchError = true;
pub_opt.createThumbnail = true;
pub_opt.maxOutputLines = Inf;
```

Run Problem scripts and publish them

```
% Problem 1
publish('HW1_P1', pub_opt);

% Problem 2
publish('HW1_P2', pub_opt);

% Problem 4-5
publish('HW1_P4', pub_opt);

% Problem 6
publish('HW1_P6', pub_opt);
```

Publishing tools and support code

```
pub_opt.outputDir = '.\tools';
pub_opt.evalCode = false;

%Publish all used functions
function_list = ...
    [function_list; 'C:\Users\John\Documents\ASEN5070_SOD\tools\fcnPrintQueue'];
for idx = 1:length(function_list)
    publish(function_list{idx}, pub_opt);
end
```

Published with MATLAB® R2013b

```
function OE = cart2oe(state)
%cart2oe   Return classical orbital elements given state vector. Earth
%   orbits only. State given in km, km/s.
%   OE = [a; e; i; RAAN; w; f] = cart2oe(state)
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

mu = 3.986e5; % km3/s2
r_vec = state(1:3);
r = norm(r_vec);
v_vec = state(4:6);
v = norm(v_vec);
h = cross(r_vec,v_vec);

% Specific Energy:
E = v*v/2 - mu/r;

a = -mu/2/E;
e = sqrt(1-dot(h,h)/a/mu);
i = acos(h(3)/norm(h));
RAAN = atan2(h(1), -h(2));

arg_lat = atan2(r_vec(3)/sin(i), (r_vec(1)*cos(RAAN)+r_vec(2)*sin(RAAN)));
cosf = (a*(1-e*e)-r)/e/r;
f = acos(max([min([1, cosf]), -1]));
if dot(r,v) < 0
    f = 2*pi - f;
end
w = arg_lat - f;

OE = [a; e; i; RAAN; w; f];
```

Published with MATLAB® R2013b

```
function jac = compute_cost_fcn_jacobian(obs, state, station_loc)
%compute_cost_fcn_jacobian   Compute the cost function jacobian for a 2D
%flat-earth problem.
fcnPrintQueue(mfilename('fullpath')) % Add this code to code appendix

num_unks = length(state);
jac = zeros(num_unks, num_unks);
for ii = 1:num_unks
    jac(ii, :) = compute_range_partials(obs(ii,:), state, station_loc);
end

end

function row = compute_range_partials(observation, state, station_loc)
%compute_range_partials   Compute a row of the flat-earth-problem jacobian.
X0 = state(1);
Y0 = state(2);
X0_dot = state(3);
Y0_dot = state(4);
g = state(5);
Xs = station_loc(1);
Ys = station_loc(2);

rho = observation(1);
t = observation(2);

xterm = -(X0 - Xs + X0_dot*t)/rho;
yterm = -(Y0 - Ys + Y0_dot*t - g*t*t/2)/rho;

row = [xterm, yterm, xterm*t, yterm*t, -yterm*t*t/2];

end
```

Published with MATLAB® R2013b

```
function cost_fnc = compute_obs_2d(obs, state, station_loc)
%compute_obs_2d   Compute the cost function given the est. state and
%   observations.  Currently 2D, range observations, num measurements are
%   exactly the num unks
fcnPrintQueue(mfilename('fullpath')) % Add this code to code appendix

num_unks = length(state);
cost_fnc = zeros(num_unks, 1);
for ii = 1:num_unks
    cost_fnc(ii) = obs(ii,1) ...
        - compute_range(state, station_loc, obs(ii,2));
end

end

function rho = compute_range(state, station_loc, t)
%compute_range   Compute the calculated range, given state.
X0 = state(1);
Y0 = state(2);
X0_dot = state(3);
Y0_dot = state(4);
g = state(5);
Xs = station_loc(1);
Ys = station_loc(2);

xterm = (X0-Xs+X0_dot*t)*(X0-Xs+X0_dot*t);
yterm = (Y0-Ys+Y0_dot*t-g*t*t/2)*(Y0-Ys+Y0_dot*t-g*t*t/2);
rho = sqrt(xterm + yterm);

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
%          fprintf('%s\n', filename);
        function_list = [function_list; filename];
    end
end
end
```

Published with MATLAB® R2013b

```
function state = oe2cart(OE)
%cart2oe   Return classical orbital elements given state vector. Earth
%   orbits only. State given in km, km/s.
%   OE = [a; e; i; RAAN; w; f] = cart2oe(state)
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

mu = 3.986e5; % km3/s2

a = OE(1);
e = OE(2);
i = OE(3);
RAAN = OE(4);
w = OE(5);
f = OE(6);

p = a*(1-e*e);
r = p/(1+e*cos(f));
h = sqrt(mu*a*(1-e*e));
heSinf_rp = h*e/r/p*sin(f);

r_vec = zeros(3,1);
r_vec(1) = r*(cos(RAAN)*cos(w+f) - sin(RAAN)*sin(w+f)*cos(i));
r_vec(2) = r*(sin(RAAN)*cos(w+f) + cos(RAAN)*sin(w+f)*cos(i));
r_vec(3) = r*(sin(i)*sin(w+f));

v_vec = zeros(3,1);
v_vec(1) = r_vec(1)*heSinf_rp ...
    - h/r*(cos(RAAN)*sin(w+f) + sin(RAAN)*cos(w+f)*cos(i));
v_vec(2) = r_vec(2)*heSinf_rp ...
    - h/r*(sin(RAAN)*sin(w+f) - cos(RAAN)*cos(w+f)*cos(i));
v_vec(3) = r_vec(3)*heSinf_rp ...
    - h/r*(sin(i)*cos(w+f));

state = [r_vec; v_vec];
```

Published with MATLAB® R2013b

```
function state_dot = two_body_state_dot(t, state)
%two_body_state_dot   Return state_dot given state. Used for numerical
%integration
fcnPrintQueue(mfilename('fullpath')) % Add this code to code app

state_dot = zeros(6,1);
state_dot(1:3) = state(4:6);
mu = 3.986e5; % km3/s2

r_vec = (state(1:3));
r = norm(r_vec);
state_dot(4:6) = -mu * r_vec/(r*r*r);
```

Published with MATLAB® R2013b