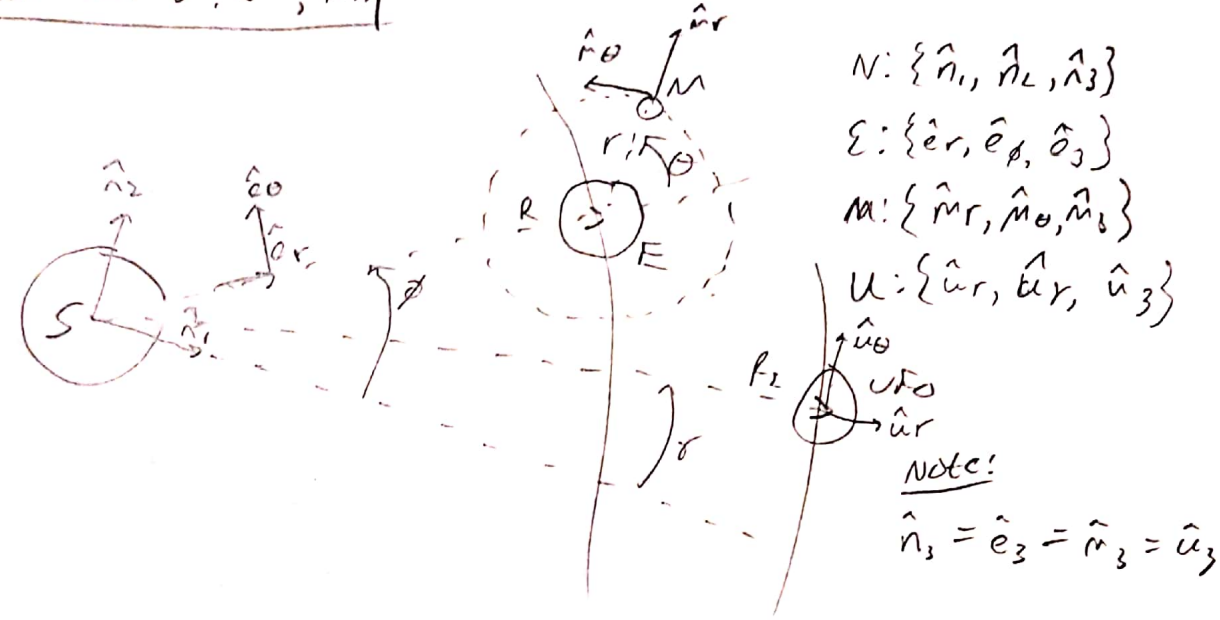
Problem 1: 5.J, 1.11



$N: \{\hat{n}_1, \hat{n}_2, \hat{n}_3\}$

$\mathcal{E}: \{\hat{e}_r, \hat{e}_\theta, \hat{e}_3\}$

$M: \{\hat{m}_r, \hat{m}_\theta, \hat{m}_3\}$

$U: \{\hat{u}_r, \hat{u}_r, \hat{u}_3\}$

Note:

$\hat{n}_3 = \hat{e}_3 = \hat{m}_3 = \hat{u}_3$

a) $\dot{\vec{r}}_{M/S}$, $\ddot{\vec{r}}_{M/S}$

$\vec{r}_{M/S} = R\hat{e}_r + r\hat{m}_r$, $\vec{\omega}_{E/S} = \dot{\phi}\hat{e}_3$, $\vec{\omega}_{M/E} = \dot{\theta}\hat{m}_3$

$\vec{\omega}_{M/S} = \vec{\omega}_{E/S} + \vec{\omega}_{M/E} = (\dot{\phi}+\dot{\theta})\hat{e}_3$

$$\dot{\vec{r}}_{M/S} = {}^N\frac{d}{dt}(\vec{r}_{M/E}) = {}^E\frac{d}{dt}(R\hat{e}_r) + \vec{\omega}_{E/S}\times R\hat{e}_r + {}^M\frac{d}{dt}(r\hat{m}_r) + \vec{\omega}_{M/S}\times r\hat{m}_r$$

$$= \dot{\phi}\hat{e}_3 \times R\hat{e}_r + (\dot{\phi}+\dot{\theta})\hat{m}_3 \times r\hat{m}_r$$

$$\boxed{\dot{\vec{r}}_{M/S} = \dot{\phi}R\hat{e}_\theta + (\dot{\phi}+\dot{\theta})r\hat{m}_\theta}$$

$$\ddot{\vec{r}}_{M/S} = {}^N\frac{d}{dt}(\dot{\vec{r}}_{M/S}) = {}^E\frac{d}{dt}(\dot{\phi}R\hat{e}_\theta) + \vec{\omega}_{E/S}\times(\dot{\phi}R\hat{e}_\theta)$$

$$+ {}^M\frac{d}{dt}((\dot{\phi}+\dot{\theta})r\hat{m}_\theta) + \vec{\omega}_{M/S}\times[(\dot{\phi}+\dot{\theta})r\hat{m}_\theta]$$

$$= \dot{\phi}\hat{e}_3 \times (\dot{\phi}R\hat{e}_\theta) + (\dot{\phi}+\dot{\theta})\hat{m}_3 \times (\dot{\phi}+\dot{\theta})r\hat{m}_\theta$$

$$= \boxed{-\dot{\phi}^2 R\hat{e}_r - (\dot{\phi}+\dot{\theta})^2 r\hat{m}_r = \ddot{\vec{r}}_{M/S}}$$

→

b) Find $\vec{r}_{M/u}$

$\vec{r}_{M/S} = R\hat{e}_r + r\hat{m}_r$, $\vec{r}_{u/S} = R_2\hat{u}_r$

$$\vec{r}_{M/u} = \vec{r}_{M/S} - \vec{r}_{u/S} = \boxed{R\hat{e}_r + r\hat{m}_r - R_2\hat{u}_r}$$

c) $\vec{\omega}_{\ell/u}$, $\vec{\omega}_{M/u}$

$\vec{\omega}_{\ell/S} = \dot{\phi}\hat{e}_3 = \dot{\phi}\hat{n}_3$, $\vec{\omega}_{M/S} = (\dot{\theta}+\dot{\phi})\hat{m}_3 = (\dot{\theta}+\dot{\phi})\hat{n}_3$, $\vec{\omega}_{u/S} = \dot{\gamma}\hat{u}_3 = \dot{\gamma}\hat{n}_3$

$$\vec{\omega}_{\ell/u} = \vec{\omega}_{\ell/S} - \vec{\omega}_{u/S} = \boxed{\dot{\phi}\hat{e}_3 - \dot{\gamma}\hat{u}_3 = \vec{\omega}_{\ell/u}}$$

$$\vec{\omega}_{M/u} = \vec{\omega}_{M/S} - \vec{\omega}_{u/S} = \boxed{(\dot{\theta}+\dot{\phi})\hat{m}_3 - \dot{\gamma}\hat{u}_3 = \vec{\omega}_{M/u}}$$

d) $\ddot{\vec{r}}_{M/u}$?

$\vec{r}_{M/u} = R\hat{e}_r + r\hat{m}_r - R_2\hat{u}_r$, $\vec{\omega}_{\ell/u} = (\dot{\phi}-\dot{\gamma})\hat{e}_3$, $\vec{\omega}_{M/u} = (\dot{\theta}+\dot{\phi}-\dot{\gamma})\hat{m}_3$

$$\dot{\vec{r}}_{M/u} = \overset{\ell}{\frac{d}{dt}}(R\hat{e}_r) + (\dot{\phi}-\dot{\gamma})\hat{e}_3 \times (R\hat{e}_r) + \overset{M}{\frac{d}{dt}}(r\hat{m}_r) + (\dot{\theta}+\dot{\phi}-\dot{\gamma})\hat{m}_3 \times (r\hat{m}_r) + \overset{u}{\frac{d}{dt}}(R_2\hat{u}_r)$$

$$= \boxed{R(\dot{\phi}-\dot{\gamma})\hat{e}_\theta + r(\dot{\theta}+\dot{\phi}-\dot{\gamma})\hat{m}_\theta}$$

$$\ddot{\vec{r}}_{M/u} = \overset{\ell}{\frac{d}{dt}}\left(R(\dot{\phi}-\dot{\gamma})\hat{e}_\theta\right) + (\dot{\phi}-\dot{\gamma})\hat{e}_3 \times \left(R(\dot{\phi}-\dot{\gamma})\hat{e}_\theta\right) + \overset{M}{\frac{d}{dt}}\left(r(\dot{\theta}+\dot{\phi}-\dot{\gamma})\hat{m}_\theta\right) + (\dot{\theta}+\dot{\phi}-\dot{\gamma})\hat{m}_3 \times \left(r(\dot{\theta}+\dot{\phi}-\dot{\gamma})\hat{m}_\theta\right)$$

$$= \boxed{-R(\dot{\phi}-\dot{\gamma})^2\hat{e}_r - r(\dot{\theta}+\dot{\phi}-\dot{\gamma})^2\hat{m}_r}$$

→

Problem 2:

$V(r) = -\frac{G m_e M}{r}$, Linearize about r_e, develop expression for potential e.n. w/ height h, for $r = r_e + h$

We do a 1st-order Taylor expansion

$$V(r) \approx V(r_e) + \left.\frac{\partial V}{\partial r}\right|_{r_e} (r - r_e) \quad \text{... (neglecting H.O.T.)}$$

$$= -\frac{G m_e M}{r_e} - \frac{G m_e M}{r_e^2}(r - r_e)$$

$$= -\frac{G m_e M}{r_e}\left(1 - \frac{1}{r_e}(r - r_e)\right)$$

$$= -\frac{G m_e M}{r_e}\left(1 - \left(\frac{r}{r_e} - 1\right)\right)$$

$$\boxed{V(r) \approx -\frac{G m_e M}{r_e}\left(2 - \frac{r}{r_e}\right)}$$

$r = r_e + h$

$$\Downarrow$$

$$V(h) = -\frac{G m_e M}{r_e}\left(2 - \frac{r_e + h}{r_e}\right)$$

$$= -\frac{G m_e M}{r_e}\left(2 - \left(1 + \frac{h}{r_e}\right)\right)$$

$$\boxed{V(h) = -\frac{G m_e M}{r_e}\left(1 - \frac{h}{r_e}\right)}$$

Problem 3: 9.9

$V(r) = -\frac{\mu}{r}$, $\vec{r} = (x, y, z)^T$, $r = \sqrt{x^2+y^2+z^2}$

$$\therefore \ddot{\vec{r}} = -\begin{pmatrix} \frac{\partial V}{\partial x} \\ \frac{\partial V}{\partial y} \\ \frac{\partial V}{\partial z} \end{pmatrix}$$

$$\frac{\partial V}{\partial x} = -\mu \frac{\partial}{\partial x}\left(\frac{1}{\sqrt{x^2+y^2+z^2}}\right) = \frac{\mu x}{(x^2+y^2+z^2)^{3/2}} = \frac{\mu x}{r^3}$$

$$\Rightarrow \frac{\partial V}{\partial y} = \frac{\mu y}{r^3} , \quad \frac{\partial V}{\partial z} = \frac{\mu z}{r^3}$$

$$\Rightarrow \begin{pmatrix} \frac{\partial V}{\partial x} \\ \frac{\partial V}{\partial y} \\ \frac{\partial V}{\partial z} \end{pmatrix} = \frac{\mu}{r^3}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \frac{\mu \vec{r}}{r^3}$$

and then, $$-\frac{\mu \vec{r}}{r^3} = \ddot{\vec{r}} = -\nabla_r V = -\begin{pmatrix} \frac{\partial V}{\partial x} \\ \frac{\partial V}{\partial y} \\ \frac{\partial V}{\partial z} \end{pmatrix}$$

I.C.

Problem 4 9.11

$$\vec{r} = \begin{bmatrix} -4069.503 \\ 2861.986 \\ 4483.608 \end{bmatrix} \text{ km} \qquad \vec{v} = \begin{bmatrix} -5.114 \\ -5.691 \\ -1.000 \end{bmatrix} \text{ km/s}$$

$r = 6.6973e3$ km , $v = 7.7162$ km/s

Assume $\mu = 398600 \text{ km}^3/\text{s}^2$

a) $\varepsilon = \frac{v^2}{2} - \frac{\mu}{r}$

$$= \frac{(7.7162 \text{ km/s})^2}{2} - \frac{398600 \text{ km}^3/\text{s}^2}{6.6973e3 \text{ km}} = -29.7466 \text{ km}^2/\text{s}^2$$

Since total energy < 0, the orbit is **hyperbolic**

b) $\vec{e}$

$$\vec{e} = \frac{\vec{v} \times (\vec{r} \times \vec{v})}{\mu} - \frac{\vec{r}}{r} = \text{*matlab*} = \begin{bmatrix} 2.8946 \\ 7.6120 \\ 3.7026 \end{bmatrix} \times 10^{-4}$$

$$e = |\vec{e}| = 8.9460 \times 10^{-4}$$

c) $r_p = a(1-e)$

$$\varepsilon = -\frac{\mu}{2a} = -29.7466 \Rightarrow a = 6699.9 \text{ km}$$

$$r_p = 6699.9(1 - 8.9460 \times 10^{-4}) = 6693.6 \text{ km}$$

Since $r_e = 6378$ km and $r_p = 6693.6$ km,

$r_p > r_e$ and will not collide with the earth.

Problem 5: 9.17

$y = r\sin f \quad , \dot{y} = \frac{h}{p}(e + \cos f)$

$\dot{y} = r\cos f\, \dot{f} + \dot{r}\sin f \quad , \dot{r} = \frac{he\sin f}{p}$

$= r\cos f\, \dot{f} + \frac{he\sin^2 f}{p}$

$= \frac{r^2 \dot{f}\cos f}{r} + \frac{he\sin^2 f}{p} \quad , h = r^2\dot{f}$

$= \frac{h\cos f}{r} + \frac{he\sin^2 f}{p} \quad , r = \frac{p}{1+e\cos f}$

$= \frac{h\cos f}{p}(1+e\cos f) + \frac{he\sin^2 f}{p}$

$= \frac{h\cos f}{p} + \frac{he\cos^2 f}{p} + \frac{he\sin^2 f}{p} \quad , he\sin^2 f + he\cos^2 f = he$

$= \frac{h\cos f}{p} + \frac{he}{p} = \boxed{\frac{h}{p}(e+\cos f)}$

Problem 6 : 9.18

$$r(f) = \frac{p}{1+e\cos f}$$

$$\dot{r} = \frac{d}{df}\left(\frac{p}{1+e\cos f}\right) = -\frac{p}{(1+e\cos f)^2} \cdot -e\sin f \cdot \dot{f}$$

$$\therefore 1+e\cos f = \frac{p}{r}$$

$$\Rightarrow \dot{r} = \frac{\dot{f}\, p e \sin f}{\left(\frac{p}{r}\right)^2} = \frac{r^2 e \sin f \cdot \dot{f}}{p}$$

$$h = r^2 \dot{f}$$

$$\Rightarrow \boxed{\dot{r} = \frac{h e \sin f}{p}}$$

Problem 7 - conversion code

File - /Users/iancooke/Dropbox/CUBoulder/Grad Year 2/FormationFlying/Homeworks_PyCharm/HW1/convert_rv_kep.py

```
1  # convert_rv_kep
2  # translate between the state vector and classical
   keplerian orbit elements
3  # Inputs:
4  #   input_flag - type of state being inputted
5  #   x - the numbers
6  #   delta_t - time elapsed since t_0
7  #   output_flag - type of state being outputted
8
9
10 # imports
11 import numpy as np
12
13
14 # # #
15 def convert_rv_kep(input_flag, x_vec, delta_t, output_flag)
   :
16
17     # define some constants
18     mu_E = 398600.0  # [km^3/s^2] standard gravitational
   parameter of the Earth
19
20     # Handle various cases, if none then it just returns
21     # keplerian to state vector
22     if input_flag == 'keplerian' and output_flag == 'state'
   :
23         # parse
24         a = x_vec[0]  # [km]
25         ecc = x_vec[1]  # [none]
26         inc = np.deg2rad(x_vec[2])  # [deg]
27         Omega = np.deg2rad(x_vec[3])  # [deg]
28         omega = np.deg2rad(x_vec[4])  # [deg]
29         M_0 = np.deg2rad(x_vec[5])  # [deg]
30
31         if ecc < 1.0:
32             M = M_0 + np.sqrt(mu_E / np.power(a, 3.0)) *
   delta_t
33             f = convert_M_to_f(M, 6, ecc)
34         else:
35             n = np.sqrt(mu_E / np.power(-a, 3.0))
36             N = M_0 + n*delta_t
37             f = convert_M_to_f(N, 6, ecc)
38
39         theta = omega + f
40
41         p = a * (1.0 - np.power(ecc, 2.0))
42
43         h = np.sqrt(mu_E * p)
44
45         r = p / (1.0 + ecc*np.cos(f))
```

```
46
47          r_x = r * (np.cos(Omega)*np.cos(theta) - np.sin(
    Omega)*np.sin(theta)*np.cos(inc))
48          r_y = r * (np.sin(Omega)*np.cos(theta) + np.cos(
    Omega)*np.sin(theta)*np.cos(inc))
49          r_z = r * (np.sin(theta)*np.sin(inc))
50
51          v_x = -mu_E / h * (np.cos(Omega)*(np.sin(theta) +
    ecc*np.sin(omega)) + np.sin(Omega)*(np.cos(theta) + ecc*np.
    cos(omega))*np.cos(inc))
52          v_y = -mu_E / h * (np.sin(Omega)*(np.sin(theta) +
    ecc*np.sin(omega)) - np.cos(Omega)*(np.cos(theta) + ecc*np.
    cos(omega))*np.cos(inc))
53          v_z = mu_E / h * (np.cos(theta) + ecc*np.cos(omega)
    )*np.sin(inc)
54
55          x_out = np.array([[r_x], [r_y], [r_z], [v_x], [v_y]
    , [v_z]])
56
57          return x_out
58
59      # state vector to keplerian
60      elif input_flag == 'state' and output_flag == '
    keplerian':
61          # parse
62          x = x_vec[0]
63          y = x_vec[1]
64          z = x_vec[2]
65          xd = x_vec[3]
66          yd = x_vec[4]
67          zd = x_vec[5]
68
69          r_vec = np.array([x, y, z])
70          v_vec = np.array([xd, yd, zd])
71
72          r = np.linalg.norm(r_vec)
73          v = np.linalg.norm(v_vec)
74
75          one_over_a = 2.0 / r - np.power(v, 2.0) / mu_E
76          a = 1.0 / one_over_a
77
78          h_vec = np.cross(r_vec, v_vec)
79          h = np.linalg.norm(h_vec)
80
81          ecc_vec = np.cross(v_vec, h_vec) / mu_E - r_vec / r
82          ecc = np.linalg.norm(ecc_vec)
83
84          ihat_e = ecc_vec / ecc
85          ihat_h = h_vec / h
86          ihat_p = np.cross(ihat_h, ihat_e)
```

```
 87
 88         PN = np.array([ihat_e.T, ihat_p.T, ihat_h.T])
 89
 90         Omega = np.arctan2(PN[2, 0], -PN[2, 1])
 91         inc = np.arccos(PN[2, 2])
 92         omega = np.arctan2(PN[0, 2], PN[1, 2])
 93         ihat_r = r_vec / r
 94         f = np.arctan2(np.dot(np.cross(ihat_e, ihat_r),
    ihat_h), np.dot(ihat_e, ihat_r))
 95         if ecc < 1.0:
 96             E = 2.0*np.arctan(np.tan(f/2.0) / np.sqrt((1.0
     + ecc)/(1.0 - ecc)))
 97             M = E - ecc*np.sin(E)
 98             n = np.sqrt(mu_E / np.power(a, 3.0))
 99         else:
100             H = 2.0*np.arctanh(np.tan(f/2) / np.sqrt((ecc
    + 1.0) / (ecc - 1.0)))
101             M = ecc*np.sinh(H) - H
102             n = np.sqrt(mu_E / np.power(-a, 3.0))
103
104         M_0 = M - n * delta_t
105         if M_0 < 0:
106             M_0 = M_0 + 2 * np.pi
107
108         return np.array([[a], [ecc], [np.rad2deg(inc)], [
    np.rad2deg(Omega)], [np.rad2deg(omega)], [np.rad2deg(M_0)]
    ])
109
110
111     # flags are the same
112     elif input_flag == output_flag:
113         return x
114     # inputs are wrong
115     else:
116         raise ValueError('Incorrect input or output flags'
    )
117
118
119 # subroutine for newton's method to solve keplers equation
     for E (eccentric anomaly)
120 # Inputs:
121 #   x_0 - [deg] initial guess
122 #   n_iter - [none] number of iterations to be completed
123 #   ecc - eccentricity of orbit
124 # Outputs:
125 #   x_k - final solution
126 def convert_M_to_f(x_0, n_iter, ecc):
127
128     # iterate
129     x_k = x_0
```

```
130     for k in range(n_iter):
131         # elliptic case
132         if ecc < 1.0:
133             x_k = x_k - (x_0 - (x_k - ecc*np.sin(x_k)))/-(
    1.0 - ecc*np.cos(x_k))
134         # hyperbolic case
135         else:
136             x_k = x_k - (x_0 - (ecc*np.sinh(x_k) - x_k))/-
    (ecc*np.cosh(x_k) - 1)
137 
138     # elliptic case
139     if ecc < 1.0:
140         f = 2.0 * np.arctan(np.sqrt((ecc + 1.0)/(1.0 - ecc
    )) * np.tan(x_k / 2.0))
141     # hyperbolic case
142     else:
143         f = 2.0 * np.arctan(np.sqrt((ecc + 1.0) / (ecc - 1
    .0)) * np.tanh(x_k / 2.0))
144 
145     return f
146 
```

Problem 7 - test code

File - /Users/iancooke/Dropbox/CUBoulder/Grad Year 2/FormationFlying/Homeworks_PyCharm/HW1/Prob7_test.py

```
 1 import numpy as np
 2 from HW1 import convert_rv_kep
 3
 4 delta_t = 3600.0
 5 # case 1
 6 a = 8000.0
 7 e = 0.1
 8 inc = 30.0
 9 Omega = 145.0
10 omega = 120.0
11 M_0 = 10.0
12
13 elems = np.array([a, e, inc, Omega, omega, M_0])
14
15 state_vec = convert_rv_kep.convert_rv_kep('keplerian',
   elems, delta_t, 'state')
16 print('Convert from first set of elements given to state
   vector:')
17 print('r_x = {0} km'.format(state_vec[0]))
18 print('r_y = {0} km'.format(state_vec[1]))
19 print('r_z = {0} km'.format(state_vec[2]))
20 print('v_x = {0} km/s'.format(state_vec[3]))
21 print('v_y = {0} km/s'.format(state_vec[4]))
22 print('v_z = {0} km/s'.format(state_vec[5]))
23 print('')
24
25 # case 2
26 delta_t = 3600.0
27 # case 1
28 a = -8000.0
29 e = 1.1
30 inc = 30.0
31 Omega = 145.0
32 omega = 120.0
33 M_0 = 10.0
34
35 elems = np.array([a, e, inc, Omega, omega, M_0])
36
37 state_vec = convert_rv_kep.convert_rv_kep('keplerian',
   elems, delta_t, 'state')
38 print('Convert from second set of elements given to state
   vector:')
39 print('r_x = {0} km'.format(state_vec[0]))
40 print('r_y = {0} km'.format(state_vec[1]))
41 print('r_z = {0} km'.format(state_vec[2]))
42 print('v_x = {0} km/s'.format(state_vec[3]))
43 print('v_y = {0} km/s'.format(state_vec[4]))
44 print('v_z = {0} km/s'.format(state_vec[5]))
45 print('')
46
```

```
47 # case 3
48 state = np.array([-1264.61, 8013.81, -3371.25, -6.03962, -0
   .204398, 2.09672])
49
50 elems_out = convert_rv_kep.convert_rv_kep('state', state,
   delta_t, 'keplerian')
51 print('Convert from state vector back to first set of
   elements')
52 print('a = {0} km'.format(elems_out[0]))
53 print('e = {0}'.format(elems_out[1]))
54 print('i = {0} deg'.format(elems_out[2]))
55 print('Omega = {0} deg'.format(elems_out[3]))
56 print('omega = {0} deg'.format(elems_out[4]))
57 print('M_0 = {0} deg'.format(elems_out[5]))
58 print('')
59
60
61 # case 4
62 state = np.array([18877, 27406.6, -19212.8, 3.55968, 6.
   35532, -4.18447])
63
64 elems_out = convert_rv_kep.convert_rv_kep('state', state,
   delta_t, 'keplerian')
65 print('Convert from state vector back to second set of
   elements')
66 print('a = {0}'.format(elems_out[0]))
67 print('e = {0}'.format(elems_out[1]))
68 print('i = {0}'.format(elems_out[2]))
69 print('Omega = {0}'.format(elems_out[3]))
70 print('omega = {0}'.format(elems_out[4]))
71 print('M_0 = {0}'.format(elems_out[5]))
```

Problem 7 - test code output

File - Prob7_test

```
/usr/local/bin/python3.7 "/Users/iancooke/Dropbox/CUBoulder
/Grad Year 2/FormationFlying/Homeworks_PyCharm/HW1/
Prob7_test.py"
Convert from first set of elements given to state vector:
r_x = [-1264.60766618] km
r_y = [8013.80931615] km
r_z = [-3371.25163985] km
v_x = [-6.03962093] km/s
v_y = [-0.2043976] km/s
v_z = [2.09671503] km/s

Convert from second set of elements given to state vector:
r_x = [18876.96930193] km
r_y = [27406.55484551] km
r_z = [-19212.78525117] km
v_x = [3.55967689] km/s
v_y = [6.35531587] km/s
v_z = [-4.18447123] km/s

Convert from state vector back to first set of elements
a = [8000.00211438] km
e = [0.09999977]
i = [30.00002023] deg
Omega = [144.99993059] deg
omega = [120.00001905] deg
M_0 = [10.00013815] deg

Convert from state vector back to second set of elements
a = [-7999.98492425]
e = [1.10000037]
i = [29.99997862]
Omega = [144.99991754]
omega = [120.00010867]
M_0 = [10.00026831]

Process finished with exit code 0

```

Problem 8 - ODE pass function

File - /Users/iancooke/Dropbox/CUBoulder/Grad Year 2/FormationFlying/Homeworks_PyCharm/HW1/kepler_J2_ODE.py

```python
import numpy as np

def kepler_J2_ODE(t, x, params):
    # get params
    mu_E = params['mu_E']
    J2 = params['J2']
    R_E = params['R_E']
    n_objs = params['n_objs']

    dxdt = np.zeros((n_objs * 6))

    for k in range(n_objs):
        r_vec = x[6*k:3+6*k]
        X = r_vec[0]
        Y = r_vec[1]
        Z = r_vec[2]
        rdot_vec = x[3+6*k:6+6*k]
        r = np.linalg.norm(r_vec)
        Z_r_2 = np.power(Z / r, 2.0)
        p_J2 = 1.5 * J2 * mu_E / np.power(r, 2.0) * np.power(R_E / r, 2.0) * \
               np.array([X/r*(5.0*Z_r_2 - 1.0), Y/r*(5.0*Z_r_2 - 1.0), Z/r*(5.0*Z_r_2 - 3.0)])
        rddot_vec = -mu_E / np.power(r, 3.0)*r_vec + p_J2
        dxdt[6*k:6*k+6] = np.concatenate((rdot_vec, rddot_vec))

    return dxdt


```

Problem 8 - test code

File - /Users/iancooke/Dropbox/CUBoulder/Grad Year 2/FormationFlying/Homeworks_PyCharm/HW1/Prob8.py

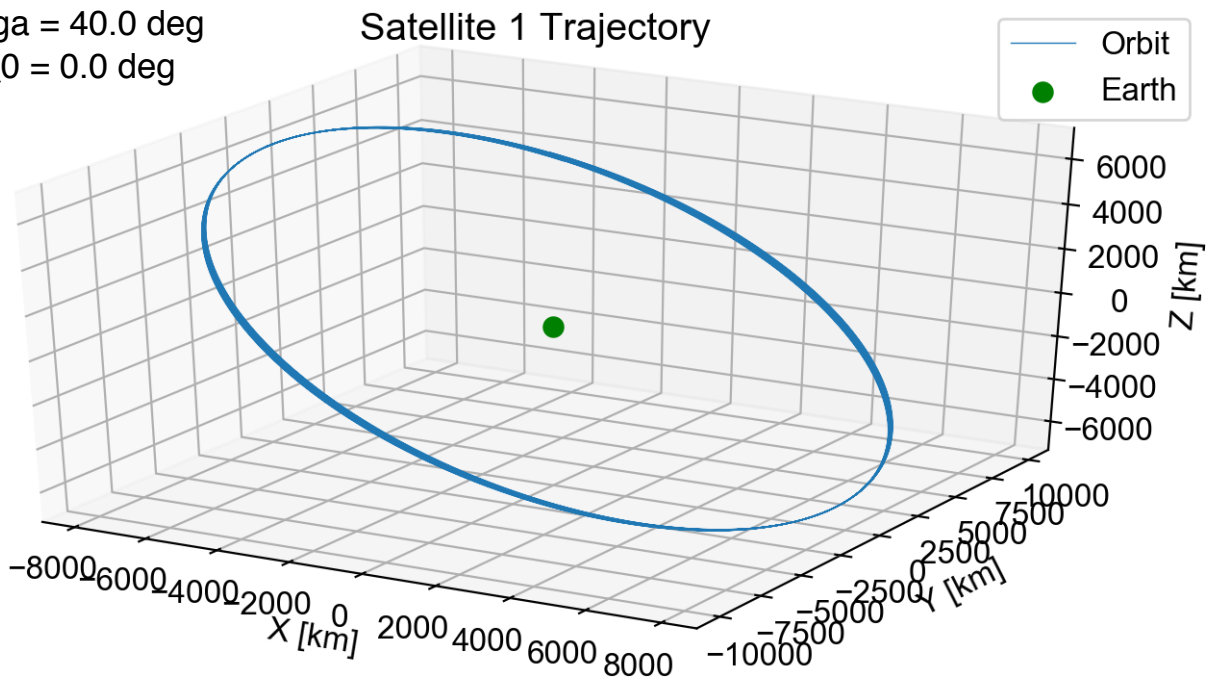
```
 1 import numpy as np
 2 from scipy import integrate
 3 from mpl_toolkits.mplot3d import Axes3D
 4 from HW1 import convert_rv_kep, kepler_J2_ODE
 5 import matplotlib.pyplot as plt
 6
 7 # constants
 8 params = {'mu_E': 398600.0,
 9           'J2': 1.082626925638815e-03,
10           'R_E': 6378.1363,
11           'n_objs': 2}
12
13 # orbit elems of first satellite
14 a = 10000.0  # [km]
15 e = 0.001  # [km]
16 inc = 40.0  # [deg]
17 Omega = 80.0  # [deg]
18 omega = 40.0  # [deg]
19 M_0 = 0.0  # [deg]
20 elems_1 = np.array([a, e, inc, Omega, omega, M_0])
21 # convert to state vector
22 x_1 = convert_rv_kep.convert_rv_kep('keplerian', elems_1, 0
   .0, 'state').reshape(6)
23
24 # orbit elems of second satellite
25 a = 10000.0  # [km]
26 e = 0.8  # [km]
27 inc = 90.0  # [deg]
28 Omega = 80.0  # [deg]
29 omega = 40.0  # [deg]
30 M_0 = 0.0  # [deg]
31 elems_2 = np.array([a, e, inc, Omega, omega, M_0])
32 # convert to state vector
33 x_2 = convert_rv_kep.convert_rv_kep('keplerian', elems_2, 0
   .0, 'state').reshape(6)
34
35
36 # # #
37 # setup the simulation
38 # calc the orbital period
39 T = 2 * np.pi * np.power(a, 1.5) / np.sqrt(params['mu_E'])
40 dt = 10.0  # [sec]
41 n_orbits = 15.0
42 tspan = np.arange(0.0, n_orbits*T, dt)
43
44 x_0 = np.concatenate((x_1, x_2))
45
46 # solve
47 x_sol = integrate.odeint(func=kepler_J2_ODE.kepler_J2_ODE,
   t=tspan, y0=x_0, tfirst=True, args=(params, ), rtol=1.0e-12
```

```
47 , atol=1.0e-12)
48 print(x_sol.shape)
49 print(tspan[-1])
50
51 # plot first
52 font = {'family' : 'arial',
53         'weight' : 'normal',
54         'size'   : 12}
55 fig1 = plt.figure(1, figsize=(9, 4.5))
56 plt.rc('font', **font)
57 ax = fig1.gca(projection='3d')
58 ax.plot(x_sol[:, 0], x_sol[:, 1], x_sol[:, 2], linewidth=0.
   5)
59 ax.scatter(0.0, 0.0, 0.0, s=50, color='green')
60 ax.set_xlabel('X [km]')
61 ax.set_ylabel('Y [km]')
62 ax.set_zlabel('Z [km]')
63 #ax.set_title('First Satellite 3D Position')
64 plt.title('Satellite 1 Trajectory')
65 plt.legend(('Orbit', 'Earth'))
66 plt.show()
67
68
69 # plot second
70 fig2 = plt.figure(2, figsize=(9, 4.5))
71 ax = fig2.gca(projection='3d')
72 plt.rc('font', **font)
73 ax.plot(x_sol[:, 6], x_sol[:, 7], x_sol[:, 8], linewidth=0.
   5)
74 ax.scatter(0.0, 0.0, 0.0, s=50, color='green')
75 ax.set_xlabel('X [km]')
76 ax.set_ylabel('Y [km]')
77 ax.set_zlabel('Z [km]')
78 #ax.set_title('Second Satellite 3D Position')
79 plt.title('Satellite 2 Trajectory')
80 plt.legend(('Orbit', 'Earth'))
81 plt.show()
82
83 # calc & plot angular momentum and energy
84 momentum = np.zeros((tspan.shape[0], params['n_objs']))
85 energy = np.zeros((tspan.shape[0], params['n_objs']))
86 for i in range(tspan.shape[0]):
87     for j in range(params['n_objs']):
88         r = x_sol[i, 6*j:3+6*j]
89         v = x_sol[i, 3+6*j:6+6*j]
90         momentum[i, j] = np.linalg.norm(np.cross(r,  v))
91         energy[i, j] = np.power(np.linalg.norm(v), 2.0) / 2
   .0 - params['mu_E'] / np.linalg.norm(r)
92
93 # print out the momentum at the right times
```

```
 94 n_steps = np.shape(np.arange(0.0, T, dt))[0]
 95 for i in range(int(n_orbits)):
 96     for j in range(params['n_objs']):
 97         r_vec = x_sol[i*n_steps, 6 * j:3 + 6 * j]
 98         v_vec = x_sol[i*n_steps, 3 + 6 * j:6 + 6 * j]
 99         r = np.linalg.norm(r_vec)
100         v = np.linalg.norm(v_vec)
101         print(r)
102         print(v)
103         h_vec = np.cross(r_vec, v_vec)
104         h = np.linalg.norm(h_vec)
105         energy = np.power(v, 2.0) / 2.0 - params['mu_E'] /
    r
106         #print(h)
107         #print(energy)
108
```

Problem 8 - test code output

a = 1000 km
e = 0.001
i = 40.0 deg
Omega = 80.0 deg
omega = 40.0 deg
M_0 = 0.0 deg



a = 1000 km
e = 0.8
i = 90.0 deg
Omega = 80.0 deg
omega = 40.0 deg
M_0 = 0.0 deg