```
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
7 # constants
 8 params = \{'mu \ E': 398600.0,
             '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 \emptyset = 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 \text{ 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 \times 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
```

```
File - /Users/iancooke/Dropbox/CUBoulder/Grad Year 2/FormationFlying/Homeworks_PyCharm/HW1/Prob8.py
47 , atol=1.0e-12)
48 print(x_sol.shape)
49 print(tspan[-1])
50
51 # plot first
52 font = {'family' : 'arial',
            'weight' : 'normal',
53
                    : 12}
54
            'size'
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.
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]):
        for j in range(params['n objs']):
87
            r = x_sol[i, 6*j:3+6*j]
88
            v = x_sol[i, 3+6*j:6+6*j]
89
            momentum[i, j] = np.linalg.norm(np.cross(r, v))
90
91
            energy[i, j] = np.power(np.linalg.norm(v), 2.0) / 2
   .0 - params['mu_E'] / np.linalg.norm(r)
```

93 # print out the momentum at the right times

92

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

```
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']):
            r_{vec} = x_{sol}[i*n_{steps}, 6*j:3+6*j]
97
            v_{vec} = x_{sol}[i*n_{steps}, 3 + 6 * j:6 + 6 * j]
 98
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)
            h = np.linalg.norm(h_vec)
104
105
            energy = np_power(v, 2.0) / 2.0 - params['mu_E'] /
     r
106
            #print(h)
107
            #print(energy)
108
```

```
1 import numpy as np
 2 from HW1 import convert_rv_kep
 4 delta_t = 3600.0
 5 # case 1
 6 a = 8000.0
 7 e = 0.1
 8 \text{ inc} = 30.0
9 0 \text{mega} = 145.0
10 omega = 120.0
11 M 0 = 10.0
12
13 elems = np.array([a, e, inc, Omega, omega, M_0])
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\} \text{ 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 0 \text{mega} = 145.0
32 \text{ omega} = 120.0
33 M 0 = 10.0
34
35 elems = np.array([a, e, inc, Omega, omega, M_0])
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 v = \{0\} km/s'.format(state vec[4]))
44 print('v_z = {0} km/s'.format(state_vec[5]))
45 print('')
46
```

```
File - /Users/iancooke/Dropbox/CUBoulder/Grad Year 2/FormationFlying/Homeworks_PyCharm/HW1/Prob7_test.py
 47 # case 3
 48 state = np.array([-1264.61, 8013.81, -3371.25, -6.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.03962, -0.0
           .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]))

```
1 import numpy as np
 2
 3 def kepler J2 ODE(t, x, params):
 4
       # get params
       mu_E = params['mu_E']
 5
 6
       J2 = params['J2']
 7
       R E = params['R E']
       n objs = params['n objs']
 8
9
10
       dxdt = np_zeros((n_objs * 6))
11
12
       for k in range(n_objs):
13
           r_{vec} = x[6*k:3+6*k]
14
           X = r_vec[0]
15
           Y = r_vec[1]
16
           Z = r_vec[2]
17
           rdot_vec = x[3+6*k:6+6*k]
18
           r = np.linalg.norm(r_vec)
19
           Z_r_2 = np_power(Z / r, 2.0)
20
           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*
21
   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
22
23
           dxdt[6*k:6*k+6] = np.concatenate((rdot_vec,
   rddot_vec))
24
25
       return dxdt
26
27
```

```
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]
            ecc = x_vec[1] # [none]
25
            inc = np_deg2rad(x_vec[2]) # [deg]
26
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)
            else:
34
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
```

 $h = np_sqrt(mu_E * p)$ 

r = p / (1.0 + ecc\*np\*cos(f))

43

44 45

```
File - /Users/iancooke/Dropbox/CUBoulder/Grad Year 2/FormationFlying/Homeworks_PyCharm/HW1/convert_rv_kep.py
46
47
            r_x = r * (np_cos(0mega)*np_cos(theta) - np_sin(
   Omega)*np.sin(theta)*np.cos(inc))
            r y = r * (np.sin(0mega)*np.cos(theta) + np.cos(
48
   Omega)*np.sin(theta)*np.cos(inc))
49
            r_z = r * (np.sin(theta)*np.sin(inc))
50
            v x = -mu E / h * (np.cos(Omega)*(np.sin(theta) +
51
   ecc*np.sin(omega)) + np.sin(Omega)*(np.cos(theta) + ecc*np.
   cos(omega))*np.cos(inc))
            v y = -mu E / h * (np.sin(0mega)*(np.sin(theta) +
52
   ecc*np.sin(omega)) - np.cos(Omega)*(np.cos(theta) + ecc*np.
   cos(omega))*np.cos(inc))
53
            vz = 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]
            z = x_vec[2]
64
65
            xd = x_vec[3]
            yd = x_vec[4]
66
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.linalq.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.linalq.norm(h vec)
80
            ecc_vec = np.cross(v_vec, h_vec) / mu_E - r_vec / r
81
82
            ecc = np.linalg.norm(ecc vec)
83
84
            ihat_e = ecc_vec / ecc
85
            ihat h = h \text{ vec } / h
86
            ihat p = np.cross(ihat h, ihat e)
```

```
File - /Users/iancooke/Dropbox/CUBoulder/Grad Year 2/FormationFlying/Homeworks PyCharm/HW1/convert rv kep.py
 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:
                 H = 2.0*np.arctanh(np.tan(f/2) / np.sqrt((ecc
100
    + 1.0) / (ecc - 1.0)))
101
                 M = ecc*np.sinh(H) - H
                 n = np.sqrt(mu_E / np.power(-a, 3.0))
102
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:
             raise ValueError('Incorrect input or output flags'
116
    )
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
```

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

```
for k in range(n_iter):
130
131
            # elliptic case
132
            if ecc < 1.0:
                x_k = x_k - (x_0 - (x_k - ecc*np_sin(x_k)))/-(
133
    1.0 - ecc*np.cos(x_k)
134
            # hyperbolic case
135
            else:
                x_k = x_k - (x_0 - (ecc*np_sinh(x_k) - x_k))/-
136
    (ecc*np*cosh(x_k) - 1)
137
        # elliptic case
138
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
```