```
import numpy as np
from scipy import integrate
from mpl_toolkits.mplot3d import Axes3D
from HW1 import convert_rv_kep, kepler_J2_ODE
import matplotlib.pyplot as plt

# constants
params = {'mu_E': 398600.0,
          'J2': 1.082626925638815e-03,
          'R_E': 6378.1363,
          'n_objs': 2}

# orbit elems of first satellite
a = 10000.0  # [km]
e = 0.001  # [km]
inc = 40.0  # [deg]
Omega = 80.0  # [deg]
omega = 40.0  # [deg]
M_0 = 0.0  # [deg]
elems_1 = np.array([a, e, inc, Omega, omega, M_0])
# convert to state vector
x_1 = convert_rv_kep.convert_rv_kep('keplerian', elems_1, 0.0, 'state').reshape(6)

# orbit elems of second satellite
a = 10000.0  # [km]
e = 0.8  # [km]
inc = 90.0  # [deg]
Omega = 80.0  # [deg]
omega = 40.0  # [deg]
M_0 = 0.0  # [deg]
elems_2 = np.array([a, e, inc, Omega, omega, M_0])
# convert to state vector
x_2 = convert_rv_kep.convert_rv_kep('keplerian', elems_2, 0.0, 'state').reshape(6)


# # #
# setup the simulation
# calc the orbital period
T = 2 * np.pi * np.power(a, 1.5) / np.sqrt(params['mu_E'])
dt = 10.0  # [sec]
n_orbits = 15.0
tspan = np.arange(0.0, n_orbits*T, dt)

x_0 = np.concatenate((x_1, x_2))

# solve
x_sol = integrate.odeint(func=kepler_J2_ODE.kepler_J2_ODE, t=tspan, y0=x_0, tfirst=True, args=(params, ), rtol=1.0e-12
```

```python
, atol=1.0e-12)
print(x_sol.shape)
print(tspan[-1])

# plot first
font = {'family' : 'arial',
        'weight' : 'normal',
        'size'   : 12}
fig1 = plt.figure(1, figsize=(9, 4.5))
plt.rc('font', **font)
ax = fig1.gca(projection='3d')
ax.plot(x_sol[:, 0], x_sol[:, 1], x_sol[:, 2], linewidth=0.5)
ax.scatter(0.0, 0.0, 0.0, s=50, color='green')
ax.set_xlabel('X [km]')
ax.set_ylabel('Y [km]')
ax.set_zlabel('Z [km]')
#ax.set_title('First Satellite 3D Position')
plt.title('Satellite 1 Trajectory')
plt.legend(('Orbit', 'Earth'))
plt.show()


# plot second
fig2 = plt.figure(2, figsize=(9, 4.5))
ax = fig2.gca(projection='3d')
plt.rc('font', **font)
ax.plot(x_sol[:, 6], x_sol[:, 7], x_sol[:, 8], linewidth=0.5)
ax.scatter(0.0, 0.0, 0.0, s=50, color='green')
ax.set_xlabel('X [km]')
ax.set_ylabel('Y [km]')
ax.set_zlabel('Z [km]')
#ax.set_title('Second Satellite 3D Position')
plt.title('Satellite 2 Trajectory')
plt.legend(('Orbit', 'Earth'))
plt.show()

# calc & plot angular momentum and energy
momentum = np.zeros((tspan.shape[0], params['n_objs']))
energy = np.zeros((tspan.shape[0], params['n_objs']))
for i in range(tspan.shape[0]):
    for j in range(params['n_objs']):
        r = x_sol[i, 6*j:3+6*j]
        v = x_sol[i, 3+6*j:6+6*j]
        momentum[i, j] = np.linalg.norm(np.cross(r,  v))
        energy[i, j] = np.power(np.linalg.norm(v), 2.0) / 2.0 - params['mu_E'] / np.linalg.norm(r)

# print out the momentum at the right times
```

```
n_steps = np.shape(np.arange(0.0, T, dt))[0]
for i in range(int(n_orbits)):
    for j in range(params['n_objs']):
        r_vec = x_sol[i*n_steps, 6 * j:3 + 6 * j]
        v_vec = x_sol[i*n_steps, 3 + 6 * j:6 + 6 * j]
        r = np.linalg.norm(r_vec)
        v = np.linalg.norm(v_vec)
        print(r)
        print(v)
        h_vec = np.cross(r_vec, v_vec)
        h = np.linalg.norm(h_vec)
        energy = np.power(v, 2.0) / 2.0 - params['mu_E'] / r
        #print(h)
        #print(energy)

```

```python
import numpy as np
from HW1 import convert_rv_kep

delta_t = 3600.0
# case 1
a = 8000.0
e = 0.1
inc = 30.0
Omega = 145.0
omega = 120.0
M_0 = 10.0

elems = np.array([a, e, inc, Omega, omega, M_0])

state_vec = convert_rv_kep.convert_rv_kep('keplerian', elems, delta_t, 'state')
print('Convert from first set of elements given to state vector:')
print('r_x = {0} km'.format(state_vec[0]))
print('r_y = {0} km'.format(state_vec[1]))
print('r_z = {0} km'.format(state_vec[2]))
print('v_x = {0} km/s'.format(state_vec[3]))
print('v_y = {0} km/s'.format(state_vec[4]))
print('v_z = {0} km/s'.format(state_vec[5]))
print('')

# case 2
delta_t = 3600.0
# case 1
a = -8000.0
e = 1.1
inc = 30.0
Omega = 145.0
omega = 120.0
M_0 = 10.0

elems = np.array([a, e, inc, Omega, omega, M_0])

state_vec = convert_rv_kep.convert_rv_kep('keplerian', elems, delta_t, 'state')
print('Convert from second set of elements given to state vector:')
print('r_x = {0} km'.format(state_vec[0]))
print('r_y = {0} km'.format(state_vec[1]))
print('r_z = {0} km'.format(state_vec[2]))
print('v_x = {0} km/s'.format(state_vec[3]))
print('v_y = {0} km/s'.format(state_vec[4]))
print('v_z = {0} km/s'.format(state_vec[5]))
print('')

```

```python
# case 3
state = np.array([-1264.61, 8013.81, -3371.25, -6.03962, -0.204398, 2.09672])

elems_out = convert_rv_kep.convert_rv_kep('state', state, delta_t, 'keplerian')
print('Convert from state vector back to first set of elements')
print('a = {0} km'.format(elems_out[0]))
print('e = {0}'.format(elems_out[1]))
print('i = {0} deg'.format(elems_out[2]))
print('Omega = {0} deg'.format(elems_out[3]))
print('omega = {0} deg'.format(elems_out[4]))
print('M_0 = {0} deg'.format(elems_out[5]))
print('')


# case 4
state = np.array([18877, 27406.6, -19212.8, 3.55968, 6.35532, -4.18447])

elems_out = convert_rv_kep.convert_rv_kep('state', state, delta_t, 'keplerian')
print('Convert from state vector back to second set of elements')
print('a = {0}'.format(elems_out[0]))
print('e = {0}'.format(elems_out[1]))
print('i = {0}'.format(elems_out[2]))
print('Omega = {0}'.format(elems_out[3]))
print('omega = {0}'.format(elems_out[4]))
print('M_0 = {0}'.format(elems_out[5]))
```

```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


```

```python
# convert_rv_kep
# translate between the state vector and classical keplerian orbit elements
# Inputs:
#   input_flag - type of state being inputted
#   x - the numbers
#   delta_t - time elapsed since t_0
#   output_flag - type of state being outputted


# imports
import numpy as np


# # #
def convert_rv_kep(input_flag, x_vec, delta_t, output_flag):

    # define some constants
    mu_E = 398600.0  # [km^3/s^2] standard gravitational parameter of the Earth

    # Handle various cases, if none then it just returns
    # keplerian to state vector
    if input_flag == 'keplerian' and output_flag == 'state':
        # parse
        a = x_vec[0]  # [km]
        ecc = x_vec[1]  # [none]
        inc = np.deg2rad(x_vec[2])  # [deg]
        Omega = np.deg2rad(x_vec[3])  # [deg]
        omega = np.deg2rad(x_vec[4])  # [deg]
        M_0 = np.deg2rad(x_vec[5])  # [deg]

        if ecc < 1.0:
            M = M_0 + np.sqrt(mu_E / np.power(a, 3.0)) * delta_t
            f = convert_M_to_f(M, 6, ecc)
        else:
            n = np.sqrt(mu_E / np.power(-a, 3.0))
            N = M_0 + n*delta_t
            f = convert_M_to_f(N, 6, ecc)

        theta = omega + f

        p = a * (1.0 - np.power(ecc, 2.0))

        h = np.sqrt(mu_E * p)

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

```python

        r_x = r * (np.cos(Omega)*np.cos(theta) - np.sin(Omega)*np.sin(theta)*np.cos(inc))
        r_y = r * (np.sin(Omega)*np.cos(theta) + np.cos(Omega)*np.sin(theta)*np.cos(inc))
        r_z = r * (np.sin(theta)*np.sin(inc))

        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))
        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))
        v_z = mu_E / h * (np.cos(theta) + ecc*np.cos(omega))*np.sin(inc)

        x_out = np.array([[r_x], [r_y], [r_z], [v_x], [v_y], [v_z]])

        return x_out

    # state vector to keplerian
    elif input_flag == 'state' and output_flag == 'keplerian':
        # parse
        x = x_vec[0]
        y = x_vec[1]
        z = x_vec[2]
        xd = x_vec[3]
        yd = x_vec[4]
        zd = x_vec[5]

        r_vec = np.array([x, y, z])
        v_vec = np.array([xd, yd, zd])

        r = np.linalg.norm(r_vec)
        v = np.linalg.norm(v_vec)

        one_over_a = 2.0 / r - np.power(v, 2.0) / mu_E
        a = 1.0 / one_over_a

        h_vec = np.cross(r_vec, v_vec)
        h = np.linalg.norm(h_vec)

        ecc_vec = np.cross(v_vec, h_vec) / mu_E - r_vec / r
        ecc = np.linalg.norm(ecc_vec)

        ihat_e = ecc_vec / ecc
        ihat_h = h_vec / h
        ihat_p = np.cross(ihat_h, ihat_e)
```

```

        PN = np.array([ihat_e.T, ihat_p.T, ihat_h.T])

        Omega = np.arctan2(PN[2, 0], -PN[2, 1])
        inc = np.arccos(PN[2, 2])
        omega = np.arctan2(PN[0, 2], PN[1, 2])
        ihat_r = r_vec / r
        f = np.arctan2(np.dot(np.cross(ihat_e, ihat_r), ihat_h), np.dot(ihat_e, ihat_r))
        if ecc < 1.0:
            E = 2.0*np.arctan(np.tan(f/2.0) / np.sqrt((1.0 + ecc)/(1.0 - ecc)))
            M = E - ecc*np.sin(E)
            n = np.sqrt(mu_E / np.power(a, 3.0))
        else:
            H = 2.0*np.arctanh(np.tan(f/2) / np.sqrt((ecc + 1.0) / (ecc - 1.0)))
            M = ecc*np.sinh(H) - H
            n = np.sqrt(mu_E / np.power(-a, 3.0))

        M_0 = M - n * delta_t
        if M_0 < 0:
            M_0 = M_0 + 2 * np.pi

        return np.array([[a], [ecc], [np.rad2deg(inc)], [np.rad2deg(Omega)], [np.rad2deg(omega)], [np.rad2deg(M_0)]])


    # flags are the same
    elif input_flag == output_flag:
        return x
    # inputs are wrong
    else:
        raise ValueError('Incorrect input or output flags')


# subroutine for newton's method to solve keplers equation for E (eccentric anomaly)
# Inputs:
#   x_0 - [deg] initial guess
#   n_iter - [none] number of iterations to be completed
#   ecc - eccentricity of orbit
# Outputs:
#   x_k - final solution
def convert_M_to_f(x_0, n_iter, ecc):

    # iterate
    x_k = x_0
```

```python
    for k in range(n_iter):
        # elliptic case
        if ecc < 1.0:
            x_k = x_k - (x_0 - (x_k - ecc*np.sin(x_k)))/-(1.0 - ecc*np.cos(x_k))
        # hyperbolic case
        else:
            x_k = x_k - (x_0 - (ecc*np.sinh(x_k) - x_k))/-(ecc*np.cosh(x_k) - 1)

    # elliptic case
    if ecc < 1.0:
        f = 2.0 * np.arctan(np.sqrt((ecc + 1.0)/(1.0 - ecc)) * np.tan(x_k / 2.0))
    # hyperbolic case
    else:
        f = 2.0 * np.arctan(np.sqrt((ecc + 1.0) / (ecc - 1.0)) * np.tanh(x_k / 2.0))

    return f

```