```
"""
    File problem_1_hw4.py created on 04/12/2021 by sebrimmer at 11:44:26

    Current Directory: HW4

    Problem 1

    Given:
    ð\235\234\207 = 398600 km & /s ’
    ð\235\221\216 = 7000 km
    ð\235\221\222 = 0.05
    ð\235\221\226 = 35Â°
    Î© = 100Â°
    ð\235\234\224 = 30Â°
    ð\235\221\200 = 0Â°

    (a) Use your code from HW 2, convert the orbit elements above into to Cartesian position and
        velocity.

    (b) Then use a numerical integrator (e.g. ode45 in MATLAB with tolerances set to 1 Ã\227 10 ()* ) to
        propagate the Cartesian initial conditions (computed above) for 10 orbit periods around the
        Earth using the perturbed two-body equations of motion where the perturbation is due to ð\235\220½ ’ .

    (c) Plot the resulting orbit.

    (d) At each time step compute and plot the corresponding orbital elements (i.e. ð\235\221\216, ð\235\221\222,
ð\235\221\226, Î©, ð\235\234\224). See
        lecture 21, slide 16.

    (e) Which elements exhibit secular drift, which elements exhibit short period variations.

    (f) Approximately what value of inclination causes Î© to precess (opposite of regress) at about
        ð\235\237\217Â°/ð\235\220\235ð\235\220\232ð\235\220² ? This is known as a sun synchronous orbit (4 credit p
roblem).

"""
# importing sys
import sys

# adding HW2 to the system path and importing necessary parts for this HW
sys.path.insert(0, '/home/sebrimmer/Documents/Uni/Year_3/AE_402/HW/HW2')
import HW2.problem_2 as p2
import HW2.problem_1 as p1

from scipy.integrate import solve_ivp
from math import pi, sqrt
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import ticker
import json


def function_j2(t, x0, mu, R, J_2=None):

    # If a J_2 value is specified, add in the p_vector calculation
    if J_2:
        i, j, k = np.array([1, 0, 0]), np.array([0, 1, 0]), np.array([0, 0, 1])
        x, y, z = x0[0], x0[1], x0[2]
        r = sqrt(x ** 2 + y ** 2 + z ** 2)

        p_constant = 1.5 * (J_2 * mu * R ** 2) / (r ** 4)

        p_vector = p_constant * ((x / r * (5 * (z / r) ** 2 - 1) * i) + \
                                 (y / r * (5 * (z / r) ** 2 - 1) * j) + \
                                 (z / r * (5 * (z / r) ** 2 - 3) * k))
    else:
        p_vector = [0] * 3

    xdot = [0] * 6

    # values 3-5 of x array are cartesian coords VELOCITY
    xdot[0], xdot[1], xdot[2] = x0[3], x0[4], x0[5]

    xdot[3] = -mu * x0[0] / (np.linalg.norm(x0[0:3]) ** 3) + p_vector[0]
    xdot[4] = -mu * x0[1] / (np.linalg.norm(x0[0:3]) ** 3) + p_vector[1]
    xdot[5] = -mu * x0[2] / (np.linalg.norm(x0[0:3]) ** 3) + p_vector[2]

    return xdot


def main():
    # Q1 Orbit Propagation
```

```
    # Orbital Elements and Earth Parameters
    mu = 398600
    R = 6378
    J_2 = 0.00108263
    orb_elmes_dict = {"a": 7000,
                      "r_p": 0,
                      "r_a": 0,
                      "e": 0.05,
                      "i": 35,
                      "omega": 100,
                      "w": 30,
                      "M": 0
                      }

    # Converting orb elems to r and v vectors:
    orb_elmes_dict['E'] = p2.kepler_E_solution_iteration(eccentricity=orb_elmes_dict['e'],
                                                          n=None,
                                                          delta_t=None,
                                                          mean_anom=(orb_elmes_dict['M'] * pi / 180),
                                                          E_0=0)

    orb_elmes_dict['f'] = p2.true_anom(orb_elmes_dict)
    orb_elmes_dict['theta'] = (orb_elmes_dict['w'] * pi / 180 + orb_elmes_dict['f'],
                               orb_elmes_dict['w'] + orb_elmes_dict['f'] * 180 / pi)

    orb_elmes_dict['r_mag'] = p2.radius_magnitude(orb_elmes_dict)

    # Q1.1 - Position vector (km)
    orb_elmes_dict['r_vector'] = p2.radius_vector(orb_elmes_dict, orb_elmes_dict['theta'][0])

    # Q1.2 - Velocity vector (km/s)
    orb_elmes_dict['h_mag'] = p2.h_value_from_elements(orb_elmes_dict, mu)
    orb_elmes_dict['v_vector'] = p2.velocity_vector(orb_elmes_dict, mu)

    # Some debug to help with checking the vectors have been created correctly
    verbose_string = f"Orbital elements to r and v vectors:\n" \
                     f"a :                          {orb_elmes_dict['a']:.2f} km\n" \
                     f"r_periapse :                 {orb_elmes_dict['r_p']}\n" \
                     f"e :                          {orb_elmes_dict['e']}\n" \
                     f"i (deg):                     {orb_elmes_dict['i']}\n" \
                     f"RAoAN (deg):                 {orb_elmes_dict['omega']}\n" \
                     f"Longit. of AN (deg):         {orb_elmes_dict['w']}\n" \
                     f"Mean Anom (deg) :            {orb_elmes_dict['M']}\n" \
                     f"Eccentric Anom (rad, deg):   {orb_elmes_dict['E'][0]:.3f}, {orb_elmes_dict['E'][1]:.3f}\n"
 \
                     f"Theta (rad):                 {orb_elmes_dict['theta'][0]:.3f}\n" \
                     f"h vector :                   {orb_elmes_dict['h_mag']:.3f}\n" \
                     f"True Anom:                   {orb_elmes_dict['f']:.3f} radians\n" \
                     f"Radius Mag:                  {orb_elmes_dict['r_mag']:.3f} km\n" \
                     f"Radius vector:               {p1.sf_vector(orb_elmes_dict['r_vector'][0], 3)} km\n" \
                     f"Radius mag (2):              {orb_elmes_dict['r_vector'][1]:.3f} km\n" \
                     f"Velocity vector:             {p1.sf_vector(orb_elmes_dict['v_vector'][0], 3)} km/s\n" \
                     f"Velocity mag (2):            {orb_elmes_dict['v_vector'][1]:.3f} km/s\n" \
                     f"Return of check function:    {p2.check_r_and_v_with_h(orb_elmes_dict)}\n" \
                     f"---------------------------------------------\n"

    # Q1.3 - J2 Perturbing acceleration vector at t=0 due to J2 perturbation only (km/s^2).
    #          Enter numbers to 8 decimal places.

    t0_r_vector, r, = orb_elmes_dict['r_vector'][0], orb_elmes_dict['r_vector'][1]
    t0_v_vector, t0_v_vector_mag, = orb_elmes_dict['v_vector'][0], orb_elmes_dict['v_vector'][1]

    i, j, k = np.array([1, 0, 0]), np.array([0, 1, 0]), np.array([0, 0, 1])
    x, y, z = t0_r_vector[0], t0_r_vector[1], t0_r_vector[2]

    p_constant = 1.5 * (J_2 * mu * R ** 2) / (r ** 4)

    p_vector = p_constant * ((x / r * (5 * (z / r) ** 2 - 1) * i) + \
                             (y / r * (5 * (z / r) ** 2 - 1) * j) + \
                             (z / r * (5 * (z / r) ** 2 - 3) * k))

    # From lecture 21 slide 13:
    # r_double_dot = -mu * (t0_r_vector / r ** 3) + p_vector

    # One orbital period (in seconds) given by:

    T = sqrt((4 * pi ** 2 * orb_elmes_dict['a'] ** 3) / mu)
    dt = 10 * T
    # TA -  I'm using 1e-13 and the integrator runs in 0.2s

    # First doing propagation without J2, just on its own
    x0 = [t0_r_vector[0], t0_r_vector[1], t0_r_vector[2],
          t0_v_vector[0], t0_v_vector[1], t0_v_vector[2]]
```

```
    # Executing propagation for the unperturbed orbit, ie J_2 value is None / 0
    sol_unperturb = solve_ivp(function_j2, [0, dt], x0, t_eval=None, args=[mu, R, None], rtol=1e-13)

    # Executing propagation for orbit WITH J_2 pertubation, constant specified
    sol_j2 = solve_ivp(function_j2, [0, dt], x0, t_eval=None, args=[mu, R, J_2], rtol=1e-13)

    # Q1.4 - r after 10 orbits,  Position vector (km) after propagation for 10 orbit periods.
    r_t_10T_unperturb = np.array([sol_unperturb.y[0][-1], sol_unperturb.y[1][-1], sol_unperturb.y[2][-1]])
    r_t_10T_j2 = np.array([sol_j2.y[0][-1], sol_j2.y[1][-1], sol_j2.y[2][-1]])

    # Q1.5 - v after 10 orbits, Velocity vector  (km/s) after propagation for 10 orbit periods.
    v_t_10T_unperturb = np.array([sol_unperturb.y[3][-1], sol_unperturb.y[4][-1], sol_unperturb.y[5][-1]])
    v_t_10T_j2 = np.array([sol_j2.y[3][-1], sol_j2.y[4][-1], sol_j2.y[5][-1]])

    # Q1.6 - Figure Upload your orbit figure. Include title, axes labels and axes units.
    fig = plt.figure()
    ax1 = plt.axes(projection='3d')

    ax1.scatter3D(sol_unperturb.y[0], sol_unperturb.y[1], sol_unperturb.y[2], s=1, c='b',
                  label="Plot of UNPERTURBED r(t) vector")
    ax1.scatter3D(sol_j2.y[0], sol_j2.y[1], sol_j2.y[2], s=1, c='orange',
                  label="Plot of r(t) vector with J2 pertubations")

    ax1.scatter3D([0], [0], [0], c='g', s=6378 * 2)
    ax1.scatter3D([0], [0], [0], c='g', s=10, label="Representation of Earth (radius=6378km)")

    ax1.set(title="Position Vector Plot from t=0 to t=10*T (with and without\n J2 orbit perturbations)")
    ax1.set_xlabel("i vector direction (x, km)")
    ax1.set_ylabel("j vector direction (y, km)")
    ax1.set_zlabel("k vector direction (z, km)")

    ax1.legend()
    fig.savefig("figures/orbit_propagation_plot_in_space.png")

    # We now need to convert the r and v vectors at each point into orbital element list
    # Q1.7  -  Semimajor axis (km) after 10 orbits.
    # Q1.8  - Eccentricity after 10 orbits.
    # Q1.9  - Inclination (deg) after 10 orbits.
    # Q1.10 - Omega, Right Ascension of ascending node (deg) after 10 orbit periods.
    # Q1.11 - Argument of periapse (deg) after 10 orbit periods.

    orb_elems_list = []

    time_vals = sol_j2.t
    sol = sol_j2.y

    # Calculating orbital elements for each point in orbit
    for count in range(len(sol_j2.y[0])):
        r_vector = {"vector": np.array([sol[0][count], sol[1][count], sol[2][count]]),
                    "mag": float(np.linalg.norm([sol[0][count], sol[1][count], sol[2][count]]))}
        v_vector = {"vector": np.array([sol[3][count], sol[4][count], sol[5][count]]),
                    "mag": float(np.linalg.norm([sol[3][count], sol[4][count], sol[5][count]]))}

        e = p1.eccentricity_from_vectors(r_vector, v_vector, mu)
        a = p1.a_from_vectors(r_vector, v_vector, mu)
        h = p1.h_value_from_vectors(r_vector, v_vector)
        n = p1.n_value(h, k)
        inclin = p1.inclination(h, k)
        omega = p1.ra_o_an(n, i)
        arg_p = p1.arg_of_periapse(e, n)

        orb_elems_list.append([a, e[1], inclin[0], omega[0], arg_p[0]])

    # Q1.12 - Upload figures showing the variation in the orbital elements.
    #         Include title, axes labels and axes units.

    fig_a, ax_a = plt.subplots()
    fig_e, ax_e = plt.subplots()
    fig_i, ax_i = plt.subplots()
    fig_omg, ax_omg = plt.subplots()
    fig_argp, ax_argp = plt.subplots()

    # list of elems to plot
    elems = [[elems[i] for elems in orb_elems_list] for i in range(5)]

    with open("orb_elems_figs_config.json", "r") as config_json_obj:
        figures_json_config = json.load(config_json_obj)

    for fig, ax, y_data, labels in zip([fig_a, fig_e, fig_i, fig_omg, fig_argp],
                                       [ax_a, ax_e, ax_i, ax_omg, ax_argp],
                                       elems,
                                       figures_json_config):
```

```python
        ax.plot(time_vals, y_data, linewidth=0.5)

        # Labels
        ax.set_ylabel(labels['title'])
        ax.set_xlabel("Time along orbits (hrs)")
        ax.set(title=labels['y_axis_label'])
        # ax.legend()

        # Turning gridlines and legend on, setting ticks fontsize
        ax.minorticks_on()
        ax.grid(b=True, which='minor', color='k', linestyle='-', alpha=0.05)
        ax.grid(b=True, which='major', color='k', linestyle='-', alpha=0.3)

        ticks_x = ticker.FuncFormatter(lambda x, pos: '{0:g}'.format(round(x / 3600, 0)))
        ax.xaxis.set_major_formatter(ticks_x)

        plt.xticks(fontsize=11)
        plt.yticks(fontsize=11)
        plt.tight_layout()

        fig.savefig(f"figures/{labels['filename']}", bbox_inches="tight")

    out_string = f"Q1.1 - Radius vector:\n" \
                 f"       {p1.sf_vector(orb_elmes_dict['r_vector'][0], 3)} km\n" \
                 f"     - Radius Magnitude : {orb_elmes_dict['r_vector'][1]:.3f} km, " \
                 f"altitude of {orb_elmes_dict['r_vector'][1] - 6378:.0f} km\n" \
                 f"Q1.2 - Velocity vector (km/s):\n" \
                 f"       {p1.sf_vector(orb_elmes_dict['v_vector'][0], 3)} km/s\n" \
                 f"     - Velocity Magnitude : {orb_elmes_dict['v_vector'][1]:.3f} km/s\n" \
                 f"Q1.3 - J2 Perturbing acceleration vector at t=0 due to J2 perturbation only (km/s^2):\n" \
                 f"       {p1.sf_vector(p_vector, 8)} km^2/s\n" \
                 f"Q1.4 - r after 10 orbits,  Position vector (km) after propagation for 10 orbit periods\n" \
                 f"     - No pertubation (J2=0): {p1.sf_vector(r_t_10T_unperturb, 3)} km, " \
                 f"{np.linalg.norm(r_t_10T_unperturb):.3f} km\n" \
                 f"     - WITH PERTUBATION (J2!=0): {p1.sf_vector(r_t_10T_j2, 3)} km, " \
                 f"{np.linalg.norm(r_t_10T_j2):.3f} km\n" \
                 f"Q1.5 - v after 10 orbits, Velocity vector (km/s) after propagation for 10 orbit periods.\n" \
                 f"     - No pertubation (J2=0): {p1.sf_vector(v_t_10T_unperturb, 3)} km/s, " \
                 f"{np.linalg.norm(v_t_10T_unperturb):.3f} km/s\n" \
                 f"     - WITH PERTUBATION (J2!=0): {p1.sf_vector(v_t_10T_j2, 3)} km/s, " \
                 f"{np.linalg.norm(v_t_10T_j2):.3f} km/s\n" \
                 f"Q1.7 - Semimajor axis (km) after 10 orbits: {orb_elems_list[-1][0]:.3f}\n" \
                 f"Q1.8 - Eccentricity after 10 orbits: {orb_elems_list[-1][1]:.3f}\n" \
                 f"Q1.9 - Inclination (deg) after 10 orbits: {orb_elems_list[-1][2]:.3f}\n" \
                 f"Q1.10 - Omega, Right Ascension of ascending node (deg) after 10 orbit periods: " \
                 f"{orb_elems_list[-1][3]:.3f}\n" \
                 f"Q1.11 - Argument of periapse (deg) after 10 orbit periods: {orb_elems_list[-1][4]:.3f}\n" \

    # print(verbose_string)
    print(out_string)
    # plt.show()

    with open('output/problem_1_output.txt', 'w') as output:
        output.write(out_string)

    return 0


if __name__ == '__main__':
    main()
```